DASSAULT SYSTEMES

SIMULIA

ABAQUS 2016

ANALYSIS USER'S GUIDE
VOLUME II: ANALYSIS



3DEXPERIENCE®

Abaqus Analysis

User's Guide

Volume II

Legal Notices

Abaqus, the 3DS logo, and SIMULIA are commercial trademarks or registered trademarks of Dassault Systèmes or its subsidiaries in the United States and/or other countries. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

Abaqus and this documentation may be used or reproduced only in accordance with the terms of the software license agreement signed by the customer, or, absent such an agreement, the then current software license agreement to which the documentation relates.

This documentation and the software described in this documentation are subject to change without prior notice.

Dassault Systèmes and its subsidiaries shall not be responsible for the consequences of any errors or omissions that may appear in this documentation.

© Dassault Systèmes, 2015

Other company, product, and service names may be trademarks or service marks of their respective owners. For additional information concerning trademarks, copyrights, and licenses, see the Legal Notices in the Abaqus 2016 Installation and Licensing Guide.

Preface

This section lists various resources that are available for help with using Abaqus Unified FEA software.

Support

Both technical software support (for problems with creating a model or performing an analysis) and systems support (for installation, licensing, and hardware-related problems) for Abaqus are offered through a global network of support offices, as well as through our online support system. Contact information for our regional offices is accessible from **SIMULIA→Locations** at www.3ds.com/simulia. The online support system is accessible by selecting the **SUBMIT A REQUEST** link at **Support - Dassault Systèmes** (http://www.3ds.com/support).

Online support

Dassault Systèmes provides a knowledge base of questions and answers, solutions to questions that we have answered, and guidelines on how to use Abaqus, Engineering Process Composer, Isight, Tosca, fe-safe, and other SIMULIA products. The knowledge base is available by using the **Search our Knowledge** option on www.3ds.com/support (http://www.3ds.com/support).

By using the online support system, you can also submit new requests for support. All support/service requests are tracked. If you contact us by means outside the system to discuss an existing support problem and you know the support request number, please mention it so that we can query the support system to see what the latest action has been.

Training

All SIMULIA regional offices offer regularly scheduled public training classes. The courses are offered in a traditional classroom form and via the Web. We also provide training seminars at customer sites. All training classes and seminars include workshops to provide as much practical experience with Abaqus as possible. For a schedule and descriptions of available classes, see the **Training** link at www.3ds.com/products-services/simulia (www.3ds.com/products-services/simulia) or call your support office.

Feedback

We welcome any suggestions for improvements to Abaqus software, the support tool, or documentation. We will ensure that any enhancement requests you make are considered for future releases. If you wish to make a suggestion about the service or products, refer to www.3ds.com/simulia. Complaints should be made by contacting your support office or by visiting **SIMULIA→Quality Assurance** at www.3ds.com/simulia (www.3ds.com/simulia).

Contents

Volume I

PART I INTRODUCTION, SPATIAL MODELING, AND EXECUTION

1. Introduction

Introduction

Introduction: general 1.1.1

Abaqus syntax and conventions

Input syntax rules 1.2.1
Conventions 1.2.2

Abaqus model definition

Defining a model in Abaqus 1.3.1

Parametric modeling

Parametric input 1.4.1

2. Spatial Modeling

Node definition

Node definition 2.1.1
Parametric shape variation 2.1.2
Nodal thicknesses 2.1.3
Normal definitions at nodes 2.1.4
Transformed coordinate systems 2.1.5
Adjusting nodal coordinates 2.1.6

Element definition

Element definition 2.2.1
Element foundations 2.2.2
Defining reinforcement 2.2.3
Defining rebar as an element property 2.2.4
Orientations 2.2.5

Surface definition

Surfaces: overview 2.3.1
Element-based surface definition 2.3.2
Node-based surface definition 2.3.3

Analytical rigid surface definition 2.3.4
Eulerian surface definition 2.3.5
Operating on surfaces 2.3.6

Rigid body definition

Rigid body definition 2.4.1

Integrated output section definition

Integrated output section definition 2.5.1

Mass adjustment

Adjust and/or redistribute mass of an element set 2.6.1

Nonstructural mass definition

Nonstructural mass definition 2.7.1

Distribution definition

Distribution definition 2.8.1

Display body definition

Display body definition 2.9.1

Assembly definition

Defining an assembly 2.10.1

Matrix definition

Defining matrices 2.11.1

Probability density function

Probability density function 2.12.1

3. Job Execution

Execution procedures: overview

Execution procedure for Abaqus: overview 3.1.1

Execution procedures

Obtaining information 3.2.1
Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution 3.2.2
SIMULIA Co-Simulation Engine director execution 3.2.3
Abaqus/Standard, Abaqus/Explicit, Abaqus/CFD, AND FMU co-simulation execution 3.2.4
Dymola model execution 3.2.5
SIMULIA Co-Simulation Engine FMU execution 3.2.6
Abaqus/CAE execution 3.2.7
Abaqus/Viewer execution 3.2.8
Topology optimization execution 3.2.9

Python execution 3.2.10
Parametric studies 3.2.11
Abaqus documentation 3.2.12
Licensing utilities 3.2.13
ASCII translation of results (`.fil`) files 3.2.14
Joining results (`.fil`) files 3.2.15
Querying the keyword/problem database 3.2.16
Fetching sample input files 3.2.17
Making user-defined executables and subroutines 3.2.18
Output database upgrade utility 3.2.19
SIM database utilities 3.2.20
Translating ODB output database files to SIM format 3.2.21
Generating output database reports 3.2.22
Joining output database (`.odb`) files from restarted analyses 3.2.23
Combining output from substructures 3.2.24
Combining data from multiple output databases 3.2.25
Network output database file connector 3.2.26
Mapping thermal and magnetic loads 3.2.27
Element matrix assembly utility 3.2.28
Fixed format conversion utility 3.2.29
Translating Nastran bulk data files to Abaqus input files 3.2.30
Translating Abaqus files to Nastran bulk data files 3.2.31
Translating ANSYS input files to partial Abaqus input files 3.2.32
Translating PAM-CRASH input files to partial Abaqus input files 3.2.33
Translating RADIOSS input files to partial Abaqus input files 3.2.34
Translating Abaqus output database files to Nastran Output2 results files 3.2.35
Translating LS-DYNA data files to Abaqus input files 3.2.36
Exchanging Abaqus data with ZAERO 3.2.37
Translating Abaqus data to MSC.ADAMS modal neutral files 3.2.38
Translating an Abaqus substructure to a SIMPACK flexible body 3.2.39
Translating an Abaqus substructure to an EXCITE flexible body 3.2.40
Translating Moldflow data to Abaqus input files 3.2.41
Encrypting and decrypting Abaqus input data 3.2.42
Job execution control 3.2.43
Hardware system verification process 3.2.44

Environment file settings

Using the Abaqus environment settings 3.3.1

Managing memory and disk resources

Managing memory and disk use in Abaqus 3.4.1

Parallel execution

Parallel execution: overview 3.5.1
Parallel execution in Abaqus/Standard 3.5.2
Parallel execution in Abaqus/Explicit 3.5.3
Parallel execution in Abaqus/CFD 3.5.4

File extension definitions

File extensions used by Abaqus 3.6.1

Fortran unit numbers

Fortran unit numbers used by Abaqus 3.7.1

PART II OUTPUT

4. Output

Output

Output 4.1.1
Output to the data and results files 4.1.2
Output to the output database 4.1.3
Error indicator output 4.1.4

Output variables

Abaqus/Standard output variable identifiers 4.2.1
Abaqus/Explicit output variable identifiers 4.2.2
Abaqus/CFD output variable identifiers 4.2.3

The postprocessing calculator

The postprocessing calculator 4.3.1

5. File Output Format

Accessing the results file

Accessing the results file: overview 5.1.1
Results file output format 5.1.2
Accessing the results file information 5.1.3
Utility routines for accessing the results file 5.1.4

OI.1 Abaqus/Standard Output Variable Index

OI.2 Abaqus/Explicit Output Variable Index

OI.3 Abaqus/CFD Output Variable Index

Volume II

PART III ANALYSIS PROCEDURES, SOLUTION, AND CONTROL

6. Analysis Procedures

Introduction

Solving analysis problems: overview 6.1.1
Defining an analysis 6.1.2
General and linear perturbation procedures 6.1.3
Multiple load case analysis 6.1.4
Direct linear equation solver 6.1.5
Iterative linear equation solver 6.1.6

Static stress/displacement analysis

Static stress analysis procedures: overview 6.2.1
Static stress analysis 6.2.2
Eigenvalue buckling prediction 6.2.3
Unstable collapse and postbuckling analysis 6.2.4
Quasi-static analysis 6.2.5
Direct cyclic analysis 6.2.6
Low-cycle fatigue analysis using the direct cyclic approach 6.2.7

Dynamic stress/displacement analysis

Dynamic analysis procedures: overview 6.3.1
Implicit dynamic analysis using direct integration 6.3.2
Explicit dynamic analysis 6.3.3
Direct-solution steady-state dynamic analysis 6.3.4
Natural frequency extraction 6.3.5
Complex eigenvalue extraction 6.3.6
Transient modal dynamic analysis 6.3.7
Mode-based steady-state dynamic analysis 6.3.8
Subspace-based steady-state dynamic analysis 6.3.9
Response spectrum analysis 6.3.10
Random response analysis 6.3.11

Steady-state transport analysis

Steady-state transport analysis 6.4.1

Heat transfer and thermal-stress analysis

Heat transfer analysis procedures: overview 6.5.1
Uncoupled heat transfer analysis 6.5.2
Fully coupled thermal-stress analysis 6.5.3

Adiabatic analysis 6.5.4

Fluid dynamic analysis

Fluid dynamic analysis procedures: overview 6.6.1
Incompressible fluid dynamic analysis 6.6.2

Electromagnetic analysis

Electromagnetic analysis procedures 6.7.1
Piezoelectric analysis 6.7.2
Coupled thermal-electrical analysis 6.7.3
Fully coupled thermal-electrical-structural analysis 6.7.4
Eddy current analysis 6.7.5
Magnetostatic analysis 6.7.6

Coupled pore fluid flow and stress analysis

Coupled pore fluid diffusion and stress analysis 6.8.1
Geostatic stress state 6.8.2

Mass diffusion analysis

Mass diffusion analysis 6.9.1

Acoustic and shock analysis

Acoustic, shock, and coupled acoustic-structural analysis 6.10.1

Abaqus/Aqua analysis

Abaqus/Aqua analysis 6.11.1

Annealing

Annealing procedure 6.12.1

7. Analysis Solution and Control

Solving nonlinear problems

Solving nonlinear problems 7.1.1

Analysis convergence controls

Convergence and time integration criteria: overview 7.2.1
Commonly used control parameters 7.2.2
Convergence criteria for nonlinear problems 7.2.3
Time integration accuracy in transient problems 7.2.4

PART IV ANALYSIS TECHNIQUES

8. Analysis Techniques: Introduction

Introduction

Analysis techniques: overview 8.1.1

9. Analysis Continuation Techniques

Restarting an analysis

Restarting an analysis 9.1.1

Importing and transferring results

Transferring results between Abaqus analyses: overview 9.2.1
Transferring results between Abaqus/Explicit and Abaqus/Standard 9.2.2
Transferring results from one Abaqus/Standard analysis to another 9.2.3
Transferring results from one Abaqus/Explicit analysis to another 9.2.4

10. Modeling Abstractions

Substructuring

Using substructures 10.1.1
Defining substructures 10.1.2

Submodeling

Submodeling: overview 10.2.1
Node-based submodeling 10.2.2
Surface-based submodeling 10.2.3

Generating matrices

Generating matrices 10.3.1
Generating thermal matrices 10.3.2

Symmetric model generation, results transfer, and analysis of cyclic symmetry models

Symmetric model generation 10.4.1
Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh 10.4.2
Analysis of models that exhibit cyclic symmetry 10.4.3

Periodic media analysis

Periodic media analysis 10.5.1

Meshed beam cross-sections

Meshed beam cross-sections 10.6.1

Modeling discontinuities as an enriched feature using the extended finite element method

Modeling discontinuities as an enriched feature using the extended finite element method 10.7.1

11. Special-Purpose Techniques

Inertia relief

Inertia relief 11.1.1

Mesh modification or replacement

Element and contact pair removal and reactivation 11.2.1

Geometric imperfections

Introducing a geometric imperfection into a model 11.3.1

Fracture mechanics

Fracture mechanics: overview 11.4.1
Contour integral evaluation 11.4.2
Crack propagation analysis 11.4.3

Surface-based fluid modeling

Surface-based fluid cavities: overview 11.5.1
Fluid cavity definition 11.5.2
Fluid exchange definition 11.5.3
Inflator definition 11.5.4

Mass scaling

Mass scaling 11.6.1

Selective subcycling

Selective subcycling 11.7.1

Steady-state detection

Steady-state detection 11.8.1

12. Adaptivity Techniques

Adaptivity techniques: overview

Adaptivity techniques 12.1.1

ALE adaptive meshing

ALE adaptive meshing: overview 12.2.1
Defining ALE adaptive mesh domains in Abaqus/Explicit 12.2.2

ALE adaptive meshing and remapping in Abaqus/Explicit 12.2.3
Modeling techniques for Eulerian adaptive mesh domains in Abaqus/Explicit 12.2.4
Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit 12.2.5
Defining ALE adaptive mesh domains in Abaqus/Standard 12.2.6
ALE adaptive meshing and remapping in Abaqus/Standard 12.2.7

Adaptive remeshing

Adaptive remeshing: overview 12.3.1
Selection of error indicators influencing adaptive remeshing 12.3.2
Solution-based mesh sizing 12.3.3

Analysis continuation after mesh replacement

Mesh-to-mesh solution mapping 12.4.1

13. Optimization Techniques

Structural optimization: overview

Structural optimization: overview 13.1.1

Optimization models

Design responses 13.2.1
Objectives and constraints 13.2.2
Creating Abaqus optimization models 13.2.3

14. Eulerian Analysis

Eulerian analysis

Eulerian analysis 14.1.1
Defining Eulerian boundaries 14.1.2
Eulerian mesh motion 14.1.3
Defining adaptive mesh refinement in the Eulerian domain 14.1.4

15. Particle Methods

Discrete element method

Discrete element method 15.1.1

Continuum particle analyses

Smoothed particle hydrodynamics 15.2.1
Finite element conversion to SPH particles 15.2.2

Particle generator

Particle generator 15.3.1

16. Sequentially Coupled Multiphysics Analyses

Sequentially coupled multiphysics analyses

Predefined fields for sequential coupling 16.1.1
Sequentially coupled thermal-stress analysis 16.1.2
Predefined loads for sequential coupling 16.1.3

17. Co-simulation

Co-simulation

Co-simulation: overview 17.1.1

Preparing an Abaqus analysis for co-simulation

Preparing an Abaqus analysis for co-simulation 17.2.1

Co-simulation between Abaqus solvers

Structural-to-structural co-simulation 17.3.1
Fluid-to-structural and conjugate heat transfer co-simulation 17.3.2
Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation 17.3.3
Executing a co-simulation 17.3.4

Co-simulation using Abaqus and discrete models

Structural-to-logical co-simulation 17.4.1

18. Extending Abaqus Analysis Functionality

User subroutines and utilities

User subroutines: overview 18.1.1
Available user subroutines 18.1.2
Available utility routines 18.1.3

19. Design Sensitivity Analysis

Design sensitivity analysis

Design sensitivity analysis 19.1.1

20. Parametric Studies

Scripting parametric studies

Scripting parametric studies 20.1.1

Parametric studies: commands

aStudy.combine(): Combine parameter samples for parametric studies. 20.2.1
aStudy.constrain(): Constrain parameter value combinations in parametric studies. 20.2.2

aStudy.define(): Define parameters for parametric studies. 20.2.3
aStudy.execute(): Execute the analysis of parametric study designs. 20.2.4
aStudy.gather(): Gather the results of a parametric study. 20.2.5
aStudy.generate(): Generate the analysis job data for a parametric study. 20.2.6
aStudy.output(): Specify the source of parametric study results. 20.2.7
aStudy=ParStudy(): Create a parametric study. 20.2.8
aStudy.report(): Report parametric study results. 20.2.9
aStudy.sample(): Sample parameters for parametric studies. 20.2.10

Volume III

PART V MATERIALS

21. Materials: Introduction

Introduction

Material library: overview 21.1.1
Material data definition 21.1.2
Combining material behaviors 21.1.3

General properties

Density 21.2.1

22. Elastic Mechanical Properties

Overview

Elastic behavior: overview 22.1.1

Linear elasticity

Linear elastic behavior 22.2.1
No compression or no tension 22.2.2
Plane stress orthotropic failure measures 22.2.3

Porous elasticity

Elastic behavior of porous materials 22.3.1

Hypoelasticity

Hypoelastic behavior 22.4.1

Hyperelasticity

Hyperelastic behavior of rubberlike materials 22.5.1
Hyperelastic behavior in elastomeric foams 22.5.2
Anisotropic hyperelastic behavior 22.5.3

Stress softening in elastomers

Mullins effect 22.6.1
Energy dissipation in elastomeric foams 22.6.2

Linear viscoelasticity

Time domain viscoelasticity 22.7.1
Frequency domain viscoelasticity 22.7.2

Nonlinear viscoelasticity

Hysteresis in elastomers 22.8.1
Parallel rheological framework 22.8.2

Rate sensitive elastomeric foams

Low-density foams 22.9.1

23. Inelastic Mechanical Properties

Overview

Inelastic behavior 23.1.1

Metal plasticity

Classical metal plasticity 23.2.1
Models for metals subjected to cyclic loading 23.2.2
Rate-dependent yield 23.2.3
Rate-dependent plasticity: creep and swelling 23.2.4
Annealing or melting 23.2.5
Anisotropic yield/creep 23.2.6
Johnson-Cook plasticity 23.2.7
Dynamic failure models 23.2.8
Porous metal plasticity 23.2.9
Cast iron plasticity 23.2.10
Two-layer viscoplasticity 23.2.11
ORNL – Oak Ridge National Laboratory constitutive model 23.2.12
Deformation plasticity 23.2.13

Other plasticity models

Extended Drucker-Prager models 23.3.1
Modified Drucker-Prager/Cap model 23.3.2
Mohr-Coulomb plasticity 23.3.3
Critical state (clay) plasticity model 23.3.4
Crushable foam plasticity models 23.3.5

Fabric materials

Fabric material behavior 23.4.1

Jointed materials

Jointed material model 23.5.1

Concrete

Concrete smeared cracking 23.6.1
Cracking model for concrete 23.6.2
Concrete damaged plasticity 23.6.3

Permanent set in rubberlike materials

Permanent set in rubberlike materials 23.7.1

24. Progressive Damage and Failure

Progressive damage and failure: overview

Progressive damage and failure 24.1.1

Damage and failure for ductile metals

Damage and failure for ductile metals: overview 24.2.1
Damage initiation for ductile metals 24.2.2
Damage evolution and element removal for ductile metals 24.2.3

Damage and failure for fiber-reinforced composites

Damage and failure for fiber-reinforced composites: overview 24.3.1
Damage initiation for fiber-reinforced composites 24.3.2
Damage evolution and element removal for fiber-reinforced composites 24.3.3

Damage and failure for ductile materials in low-cycle fatigue analysis

Damage and failure for ductile materials in low-cycle fatigue analysis: overview 24.4.1
Damage initiation for ductile materials in low-cycle fatigue 24.4.2
Damage evolution for ductile materials in low-cycle fatigue 24.4.3

25. Hydrodynamic Properties

Overview

Hydrodynamic behavior: overview 25.1.1

Equations of state

Equation of state 25.2.1

26. Other Material Properties

Mechanical properties

Material damping 26.1.1
Thermal expansion 26.1.2
Field expansion 26.1.3
Viscosity 26.1.4

Heat transfer properties

Thermal properties: overview 26.2.1
Conductivity 26.2.2
Specific heat 26.2.3
Latent heat 26.2.4

Acoustic properties

Acoustic medium 26.3.1

Mass diffusion properties

Diffusivity 26.4.1
Solubility 26.4.2

Electromagnetic properties

Electrical conductivity 26.5.1
Piezoelectric behavior 26.5.2
Magnetic permeability 26.5.3

Pore fluid flow properties

Pore fluid flow properties 26.6.1
Permeability 26.6.2
Porous bulk moduli 26.6.3
Sorption 26.6.4
Swelling gel 26.6.5
Moisture swelling 26.6.6

User materials

User-defined mechanical material behavior 26.7.1
User-defined thermal material behavior 26.7.2

Volume IV

PART VI ELEMENTS

27. Elements: Introduction

Introduction

Element library: overview 27.1.1
Choosing the element's dimensionality 27.1.2
Choosing the appropriate element for an analysis type 27.1.3
Section controls 27.1.4

28. Continuum Elements

General-purpose continuum elements

Solid (continuum) elements 28.1.1
One-dimensional solid (link) element library 28.1.2
Two-dimensional solid element library 28.1.3
Three-dimensional solid element library 28.1.4
Cylindrical solid element library 28.1.5
Axisymmetric solid element library 28.1.6
Axisymmetric solid elements with nonlinear, asymmetric deformation 28.1.7

Fluid continuum elements

Fluid (continuum) elements 28.2.1
Fluid element library 28.2.2

Infinite elements

Infinite elements 28.3.1
Infinite element library 28.3.2

Warping elements

Warping elements 28.4.1
Warping element library 28.4.2

29. Structural Elements

Membrane elements

Membrane elements 29.1.1
General membrane element library 29.1.2
Cylindrical membrane element library 29.1.3
Axisymmetric membrane element library 29.1.4

Truss elements

Truss elements 29.2.1
Truss element library 29.2.2

Beam elements

Beam modeling: overview 29.3.1
Choosing a beam cross-section 29.3.2
Choosing a beam element 29.3.3
Beam element cross-section orientation 29.3.4
Beam section behavior 29.3.5
Using a beam section integrated during the analysis to define the section behavior 29.3.6
Using a general beam section to define the section behavior 29.3.7
Beam element library 29.3.8
Beam cross-section library 29.3.9

Frame elements

Frame elements 29.4.1
Frame section behavior 29.4.2
Frame element library 29.4.3

Elbow elements

Pipes and pipebends with deforming cross-sections: elbow elements 29.5.1
Elbow element library 29.5.2

Shell elements

Shell elements: overview 29.6.1
Choosing a shell element 29.6.2
Defining the initial geometry of conventional shell elements 29.6.3
Shell section behavior 29.6.4
Using a shell section integrated during the analysis to define the section behavior 29.6.5
Using a general shell section to define the section behavior 29.6.6
Three-dimensional conventional shell element library 29.6.7
Continuum shell element library 29.6.8
Axisymmetric shell element library 29.6.9
Axisymmetric shell elements with nonlinear, asymmetric deformation 29.6.10

30. Inertial, Rigid, and Capacitance Elements

Point mass elements

Point masses 30.1.1
Mass element library 30.1.2

Rotary inertia elements

Rotary inertia 30.2.1

Rotary inertia element library 30.2.2

Rigid elements

Rigid elements 30.3.1
Rigid element library 30.3.2

Capacitance elements

Point capacitance 30.4.1
Capacitance element library 30.4.2

31. Connector Elements

Connector elements

Connectors: overview 31.1.1
Connector elements 31.1.2
Connector actuation 31.1.3
Connector element library 31.1.4
Connection-type library 31.1.5

Connector element behavior

Connector behavior 31.2.1
Connector elastic behavior 31.2.2
Connector damping behavior 31.2.3
Connector functions for coupled behavior 31.2.4
Connector friction behavior 31.2.5
Connector plastic behavior 31.2.6
Connector damage behavior 31.2.7
Connector stops and locks 31.2.8
Connector failure behavior 31.2.9
Connector uniaxial behavior 31.2.10

32. Special-Purpose Elements

Spring elements

Springs 32.1.1
Spring element library 32.1.2

Dashpot elements

Dashpots 32.2.1
Dashpot element library 32.2.2

Flexible joint elements

Flexible joint element 32.3.1
Flexible joint element library 32.3.2

Distributing coupling elements

Distributing coupling elements 32.4.1
Distributing coupling element library 32.4.2

Cohesive elements

Cohesive elements: overview 32.5.1
Choosing a cohesive element 32.5.2
Modeling with cohesive elements 32.5.3
Defining the cohesive element's initial geometry 32.5.4
Defining the constitutive response of cohesive elements using a continuum approach 32.5.5
Defining the constitutive response of cohesive elements using a traction-separation description 32.5.6
Defining the constitutive response of fluid within the cohesive element gap 32.5.7
Defining the constitutive response of fluid transitioning from Darcy flow to Poiseuille flow 32.5.8
Two-dimensional cohesive element library 32.5.9
Three-dimensional cohesive element library 32.5.10
Axisymmetric cohesive element library 32.5.11

Gasket elements

Gasket elements: overview 32.6.1
Choosing a gasket element 32.6.2
Including gasket elements in a model 32.6.3
Defining the gasket element's initial geometry 32.6.4
Defining the gasket behavior using a material model 32.6.5
Defining the gasket behavior directly using a gasket behavior model 32.6.6
Two-dimensional gasket element library 32.6.7
Three-dimensional gasket element library 32.6.8
Axisymmetric gasket element library 32.6.9

Surface elements

Surface elements 32.7.1
General surface element library 32.7.2
Cylindrical surface element library 32.7.3
Axisymmetric surface element library 32.7.4

Tube support elements

Tube support elements 32.8.1
Tube support element library 32.8.2

Line spring elements

Line spring elements for modeling part-through cracks in shells 32.9.1
Line spring element library 32.9.2

Elastic-plastic joints

Elastic-plastic joints 32.10.1
Elastic-plastic joint element library 32.10.2

Drag chain elements

Drag chains 32.11.1
Drag chain element library 32.11.2

Pipe-soil elements

Pipe-soil interaction elements 32.12.1
Pipe-soil interaction element library 32.12.2

Acoustic interface elements

Acoustic interface elements 32.13.1
Acoustic interface element library 32.13.2

Eulerian elements

Eulerian elements 32.14.1
Eulerian element library 32.14.2

Fluid pipe elements

Fluid pipe elements 32.15.1
Fluid pipe element library 32.15.2

Fluid pipe connector elements

Fluid pipe connector elements 32.16.1
Fluid pipe connector element library 32.16.2

User-defined elements

User-defined elements 32.17.1
User-defined element library 32.17.2

33. Particle Elements

Discrete particle elements

Discrete particle elements 33.1.1
Discrete particle element library 33.1.2

Continuum particle elements

Continuum particle elements 33.2.1
Continuum particle element library 33.2.2

PART VII ELEMENT INDEXES

EI.1 Abaqus/Standard Element Index

EI.2 Abaqus/Explicit Element Index

EI.3 Abaqus/CFD Element Index

Volume V

PART VIII PRESCRIBED CONDITIONS

35. Prescribed Conditions

Overview

Prescribed conditions: overview 35.1.1
Amplitude curves 35.1.2

Initial conditions

Initial conditions in Abaqus/Standard and Abaqus/Explicit 35.2.1
Initial conditions in Abaqus/CFD 35.2.2

Boundary conditions

Boundary conditions in Abaqus/Standard and Abaqus/Explicit 35.3.1
Boundary conditions in Abaqus/CFD 35.3.2

Loads

Applying loads: overview 35.4.1
Concentrated loads 35.4.2
Distributed loads 35.4.3
Thermal loads 35.4.4
Electromagnetic loads 35.4.5
Acoustic and shock loads 35.4.6
Pore fluid flow 35.4.7

Prescribed assembly loads

Prescribed assembly loads 35.5.1

Predefined fields

Predefined fields 35.6.1

PART IX CONSTRAINTS

36. Constraints

Overview

Kinematic constraints: overview 36.1.1

Multi-point constraints

Linear constraint equations 36.2.1

General multi-point constraints 36.2.2
Kinematic coupling constraints 36.2.3

Surface-based constraints

Mesh tie constraints 36.3.1
Coupling constraints 36.3.2
Shell-to-solid coupling 36.3.3
Mesh-independent fasteners 36.3.4

Embedded elements

Embedded elements 36.4.1

Element end release

Element end release 36.5.1

Overconstraint checks

Overconstraint checks 36.6.1

PART X INTERACTIONS

37. Defining Contact Interactions

Overview

Contact interaction analysis: overview 37.1.1

Defining general contact in Abaqus/Standard

Defining general contact interactions in Abaqus/Standard 37.2.1
Surface properties for general contact in Abaqus/Standard 37.2.2
Contact properties for general contact in Abaqus/Standard 37.2.3
Controlling initial contact status in Abaqus/Standard 37.2.4
Stabilization for general contact in Abaqus/Standard 37.2.5
Numerical controls for general contact in Abaqus/Standard 37.2.6

Defining contact pairs in Abaqus/Standard

Defining contact pairs in Abaqus/Standard 37.3.1
Assigning surface properties for contact pairs in Abaqus/Standard 37.3.2
Assigning contact properties for contact pairs in Abaqus/Standard 37.3.3
Modeling contact interference fits in Abaqus/Standard 37.3.4
Adjusting initial surface positions and specifying initial clearances in Abaqus/Standard contact pairs 37.3.5
Adjusting contact controls in Abaqus/Standard 37.3.6
Defining tied contact in Abaqus/Standard 37.3.7
Extending master surfaces and slide lines 37.3.8

Contact modeling if substructures are present 37.3.9
Contact modeling if asymmetric-axisymmetric elements are present 37.3.10

Defining general contact in Abaqus/Explicit

Defining general contact interactions in Abaqus/Explicit 37.4.1
Assigning surface properties for general contact in Abaqus/Explicit 37.4.2
Assigning contact properties for general contact in Abaqus/Explicit 37.4.3
Controlling initial contact status for general contact in Abaqus/Explicit 37.4.4
Contact controls for general contact in Abaqus/Explicit 37.4.5

Defining contact pairs in Abaqus/Explicit

Defining contact pairs in Abaqus/Explicit 37.5.1
Assigning surface properties for contact pairs in Abaqus/Explicit 37.5.2
Assigning contact properties for contact pairs in Abaqus/Explicit 37.5.3
Adjusting initial surface positions and specifying initial clearances for contact pairs in Abaqus/Explicit 37.5.4
Contact controls for contact pairs in Abaqus/Explicit 37.5.5

38. Contact Property Models

Mechanical contact properties

Mechanical contact properties: overview 38.1.1
Contact pressure-overclosure relationships 38.1.2
Contact damping 38.1.3
Contact blockage 38.1.4
Frictional behavior 38.1.5
User-defined interfacial constitutive behavior 38.1.6
Pressure penetration loading 38.1.7
Interaction of debonded surfaces 38.1.8
Breakable bonds 38.1.9
Surface-based cohesive behavior 38.1.10

Thermal contact properties

Thermal contact properties 38.2.1

Electrical contact properties

Electrical contact properties 38.3.1

Pore fluid contact properties

Pore fluid contact properties 38.4.1

39. Contact Formulations and Numerical Methods

Contact formulations and numerical methods in Abaqus/Standard

Contact formulations in Abaqus/Standard 39.1.1

Contact constraint enforcement methods in Abaqus/Standard 39.1.2
Smoothing contact surfaces in Abaqus/Standard 39.1.3

Contact formulations and numerical methods in Abaqus/Explicit

Contact formulation for general contact in Abaqus/Explicit 39.2.1
Contact formulations for contact pairs in Abaqus/Explicit 39.2.2
Contact constraint enforcement methods in Abaqus/Explicit 39.2.3

40. Contact Difficulties and Diagnostics

Resolving contact difficulties in Abaqus/Standard

Contact diagnostics in an Abaqus/Standard analysis 40.1.1
Common difficulties associated with contact modeling in Abaqus/Standard 40.1.2

Resolving contact difficulties in Abaqus/Explicit

Contact diagnostics in an Abaqus/Explicit analysis 40.2.1
Common difficulties associated with contact modeling using contact pairs in Abaqus/Explicit 40.2.2

41. Contact Elements in Abaqus/Standard

Contact modeling with elements

Contact modeling with elements 41.1.1

Gap contact elements

Gap contact elements 41.2.1
Gap element library 41.2.2

Tube-to-tube contact elements

Tube-to-tube contact elements 41.3.1
Tube-to-tube contact element library 41.3.2

Slide line contact elements

Slide line contact elements 41.4.1
Axisymmetric slide line element library 41.4.2

Rigid surface contact elements

Rigid surface contact elements 41.5.1
Axisymmetric rigid surface contact element library 41.5.2

42. Defining Cavity Radiation in Abaqus/Standard

Defining cavity radiation

Cavity radiation 42.1.1

Part III: Analysis Procedures, Solution, and Control

- Chapter 6, “Analysis Procedures”
- Chapter 7, “Analysis Solution and Control”

6. Analysis Procedures

Introduction	6.1
Static stress/displacement analysis	6.2
Dynamic stress/displacement analysis	6.3
Steady-state transport analysis	6.4
Heat transfer and thermal-stress analysis	6.5
Fluid dynamic analysis	6.6
Electromagnetic analysis	6.7
Coupled pore fluid flow and stress analysis	6.8
Mass diffusion analysis	6.9
Acoustic and shock analysis	6.10
Abaqus/Aqua analysis	6.11
Annealing	6.12

6.1 Introduction

- “Solving analysis problems: overview,” Section 6.1.1
- “Defining an analysis,” Section 6.1.2
- “General and linear perturbation procedures,” Section 6.1.3
- “Multiple load case analysis,” Section 6.1.4
- “Direct linear equation solver,” Section 6.1.5
- “Iterative linear equation solver,” Section 6.1.6

6.1.1 SOLVING ANALYSIS PROBLEMS: OVERVIEW

Overview

A large class of stress analysis problems can be solved with Abaqus/Standard and Abaqus/Explicit. A fundamental division of such problems is into static or dynamic response; dynamic problems are those in which inertia effects are significant. Abaqus/CFD solves a broad range of incompressible flow problems.

An analysis problem history is defined using steps in Abaqus ("Defining an analysis," Section 6.1.2). For each step you choose an analysis procedure, which defines the type of analysis to be performed during the step. The available analysis procedures are listed below and described in more detail in the referenced sections.

Abaqus provides multiphysics capabilities using built-in fully coupled procedures, sequential coupling, and co-simulation as solution techniques for multiphysics simulation. An extensive selection of additional analysis techniques that provide powerful tools for performing your Abaqus analyses more efficiently and effectively is available; see Part IV, "Analysis Techniques."

Abaqus/Standard analysis

Abaqus/Standard offers complete flexibility in making the distinction between static and dynamic response; the same analysis can contain several static and dynamic phases. Thus, a static preload might be applied, and then the linear or nonlinear dynamic response computed (as in the case of vibrations of a component of a rotating machine or the response of a flexible offshore system that is initially moved to an equilibrium position subject to buoyancy and steady current loads and then is excited by wave loading). Similarly, the static solution can be sought after a dynamic event (by following a dynamic analysis step with a step of static loading). See "Static stress/displacement analysis," Section 6.2, and "Dynamic stress/displacement analysis," Section 6.3, for information on these types of procedures. In addition to static and dynamic stress analysis, Abaqus/Standard offers the following analysis types:

- "Steady-state transport analysis," Section 6.4
- "Heat transfer and thermal-stress analysis," Section 6.5
- "Electromagnetic analysis," Section 6.7
- "Coupled pore fluid flow and stress analysis," Section 6.8
- "Mass diffusion analysis," Section 6.9
- "Acoustic and shock analysis," Section 6.10
- "Abaqus/Aqua analysis," Section 6.11

Abaqus/Explicit analysis

Abaqus/Explicit solves dynamic response problems using an explicit direct-integration procedure. See "Dynamic stress/displacement analysis," Section 6.3, for more information on the explicit dynamic procedures available in Abaqus. Abaqus/Explicit also provides heat transfer, acoustic, and annealing analysis capabilities: see "Heat transfer and thermal-stress analysis," Section 6.5; "Acoustic and shock analysis," Section 6.10; and "Annealing," Section 6.12, for details.

Abaqus/CFD analysis

Abaqus/CFD solves a broad range of incompressible flow problems using a second-order projection method. See "Fluid dynamic analysis," Section 6.6, for details on the incompressible flow procedures available in Abaqus.

Multiphysics analyses

Multiphysics is a coupled approach in the numerical solution of multiple interacting physical domains. Abaqus provides built-in fully coupled procedures, sequential coupling, and co-simulation as solution techniques for multiphysics simulation.

Built-in fully coupled procedures

Native Abaqus multiphysics capabilities solve the physics by adding degrees of freedom representing each of the physical fields and using a single solver. Abaqus provides the following built-in fully coupled procedures to solve multidisciplinary simulations, where all physics fields are computed by Abaqus:

- "Fully coupled thermal-stress analysis," Section 6.5.3
- "Coupled thermal-electrical analysis," Section 6.7.3
- "Fully coupled thermal-electrical-structural analysis," Section 6.7.4
- "Piezoelectric analysis," Section 6.7.2 (electrical and mechanical coupling)
- "Eddy current analysis," Section 6.7.5 (electromagnetic)
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1
- "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1
- "Eulerian analysis," Section 14.1.1

Sequential coupling

A sequentially coupled multiphysics analysis can be used when the coupling between one or more of the physical fields in a model is only important in one direction. A common example is a thermal-stress analysis in which the temperature field does not depend strongly on the stress field. A typical sequentially coupled thermal-stress analysis consists of two Abaqus/Standard runs: a heat transfer analysis and a subsequent stress analysis.

You can perform sequentially coupled multiphysics analyses in Abaqus/Standard as described in the following sections:

- "Predefined fields for sequential coupling," Section 16.1.1
- "Sequentially coupled thermal-stress analysis," Section 16.1.2
- "Predefined loads for sequential coupling," Section 16.1.3

Co-simulation

The co-simulation technique is a multiphysics capability for run-time coupling of Abaqus and another analysis program. An Abaqus analysis can be coupled to another Abaqus analysis or to a third-party analysis program to perform multidisciplinary simulations and multidomain (multimodel) coupling.

The co-simulation technique is described in the following sections:

- "Co-simulation: overview," Section 17.1.1
- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1
- "Structural-to-structural co-simulation," Section 17.3.1
- "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2
- "Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation," Section 17.3.3
- "Executing a co-simulation," Section 17.3.4
- "Structural-to-logical co-simulation," Section 17.4.1

6.1.2 DEFINING AN ANALYSIS

Overview

An analysis is defined in Abaqus by:

- dividing the problem history into steps;
- specifying an analysis procedure for each step; and
- prescribing loads, boundary conditions, and output requests for each step.

Abaqus distinguishes between general analysis steps and linear perturbation steps, and you can include multiple steps in your analysis. You can control how prescribed conditions are applied throughout each step. In addition, you can specify

- the incrementation scheme used for controlling the solution,
- the matrix storage and solution scheme in Abaqus/Standard, and
- the precision level of the Abaqus/Explicit executable.

Defining an analysis

An analysis in Abaqus is defined using steps, analysis procedures, and optional history data.

Defining steps

A basic concept in Abaqus is the division of the problem history into *steps*. A step is any convenient phase of the history—a thermal transient, a creep hold, a dynamic transient, etc. In its simplest form a step can be just a static analysis in Abaqus/Standard of a load change from one magnitude to another. You can provide a description of each step that will appear in the data (`.dat`) file; this description is for convenience only.

The step definition includes the type of analysis to be performed and optional history data, such as loads, boundary conditions, and output requests.

Input File Usage: Use the first option to begin a step and the second option to end a step:

```
*STEP
*END STEP
```

The optional data lines on the *STEP option can be used to specify the step description. The first data line given appears in the data (`.dat`) file.

Abaqus/CAE Usage: Step module: **Create Step**: **Description**

Specifying the analysis procedure

For each step you choose an *analysis procedure*. This choice defines the type of analysis to be performed during the step: static stress analysis, dynamic stress analysis, eigenvalue buckling, transient heat transfer analysis, etc. The available analysis procedures are described in "Solving analysis problems: overview," Section 6.1.1. Only one procedure is allowed per step.

Input File Usage: The procedure definition option must immediately follow the *STEP option.

Abaqus/CAE Usage: Step module: **Create Step**: choose the procedure type

Prescribing loads, boundary conditions, and output requests

The step definition includes optional history data, such as loads, boundary conditions, and output requests, as defined in "History data" in "Defining a model in Abaqus," Section 1.3.1. For more information, see "Boundary conditions," Section 34.3; "Loads," Section 34.4; and "Output," Section 4.1.

Details for prescribing these conditions are discussed in the individual procedure sections.

Input File Usage: The optional history data are defined following the procedure definition within a *STEP block.

Abaqus/CAE Usage: You define history data (step-dependent objects) in the Interaction module, Load module, and Step module.

General analysis steps versus linear perturbation steps

There are two kinds of steps in Abaqus: general analysis steps, which can be used to analyze linear or nonlinear response, and linear perturbation steps, which can be used only to analyze linear problems. General analysis steps can be included in an Abaqus/Standard or Abaqus/Explicit analysis; linear perturbation analysis steps are available only in Abaqus/Standard. In Abaqus/Standard linear analysis is always considered to be linear perturbation analysis about the state at the time when the linear analysis procedure is introduced. This linear perturbation approach allows general application of linear analysis techniques in cases where the linear response depends on preloading or on the nonlinear response history of the model. See "General and linear perturbation procedures," Section 6.1.3, for more details.

Multiple load case analysis

In general analysis steps Abaqus/Standard calculates the solution for a single set of applied loads. This is also the default for linear perturbation steps. However, for static, direct steady-state dynamic, and SIM-based steady-state dynamic linear perturbation steps it is possible to find solutions for multiple load cases. See "Multiple load case analysis," Section 6.1.4, for a description of this capability.

Multiple steps

The analysis procedure can be changed from step to step in any meaningful way, so you have great flexibility in performing analyses. Since the state of the model (stresses, strains, temperatures, etc.) is updated throughout all general analysis steps, the effects of previous history are always included in the response in each new analysis step. Thus, for example, if natural frequency extraction is performed after a geometrically nonlinear static analysis step, the preload stiffness will be included. Linear perturbation steps have no effect on subsequent general analysis steps.

The most obvious reason for using several steps in an analysis is to change analysis procedure type. However, several steps can also be used as a matter of convenience—for example, to change output requests, contact pairs in Abaqus/Explicit, boundary conditions, or loading (any information specified as history, or step-dependent, data). Sometimes an analysis may have progressed to a point where the present step definition needs to be modified. Abaqus provides for this contingency with the

restart capability, whereby a step can be terminated prematurely and a new step can be defined for the problem continuation (see "Restarting an analysis," Section 9.1.1).

Optional history data (see "Defining a model in Abaqus," Section 1.3.1) prescribing the loading, boundary conditions, output controls, and auxiliary controls will remain in effect for all subsequent general analysis steps, including those that are defined in a restart analysis, until they are modified or reset. Abaqus will compare all loads and boundary conditions specified in a step with the loads and boundary conditions in effect during the previous step to ensure consistency and continuity. This comparison is expensive if the number of individually specified loads and boundary conditions is very large. Hence, the number of individually specified loads and boundary conditions should be minimized, which can usually be done by using element and node sets instead of individual elements and nodes. For linear perturbation steps only the output controls are continued from one linear perturbation step to the next if there are no intermediate general analysis steps and the output controls are not redefined (see "Output," Section 4.1.1).

Within Abaqus/Standard or Abaqus/Explicit, any combination of available procedures can be used from step to step. However, Abaqus/Standard and Abaqus/Explicit procedures cannot be used in the same analysis. See "Transferring results between Abaqus analyses: overview," Section 9.2.1, for information on importing results from one type of analysis to another.

Defining time varying prescribed conditions

By default, Abaqus assumes that external parameters, such as load magnitudes and boundary conditions, are constant (step function) or vary linearly (ramped) over a step, depending on the analysis procedure, as shown in Table 6.1.2–1. Some exceptions in Abaqus/Standard are discussed below.

Table 6.1.2–1 Default amplitude variations for time domain procedures.

Procedure	Default amplitude variation
Coupled pore fluid diffusion/stress (steady-state)	Ramp
Coupled pore fluid diffusion/stress (transient)	Step
Coupled thermal-electrical (steady-state)	Ramp
Coupled thermal-electrical (transient)	Step
Direct-integration dynamic	Step (exception: Ramp if quasi-static application type is specified)
Fully coupled thermal-electrical-structural in Abaqus/Standard (steady-state)	Ramp
Fully coupled thermal-electrical-structural in Abaqus/Standard (transient)	Step

Procedure	Default amplitude variation
Fully coupled thermal-stress in Abaqus/Standard (steady-state)	Ramp
Fully coupled thermal-stress in Abaqus/Standard (transient)	Step
Fully coupled thermal-stress in Abaqus/Explicit	Step
Incompressible flow	Step
Magnetostatic	Ramp
Mass diffusion (steady-state)	Ramp
Mass diffusion (transient)	Step
Quasi-static	Step
Static	Ramp
Steady-state transport	Ramp
Transient eddy current	Step
Transient modal dynamic	Step
Uncoupled heat transfer	Ramp
Uncoupled heat transfer (transient)	Step

No default amplitude variation is defined for a direct cyclic analysis step; for each applied load or boundary condition, the amplitude must be defined explicitly.

Additional default amplitude variations in Abaqus/Standard

For displacement or rotation degrees of freedom prescribed in Abaqus/Standard using displacement-type boundary conditions or displacement-type connector motions, the default amplitude variation is a ramp function for all procedure types; the default amplitude is a step function for all procedure types when using velocity-type boundary conditions or velocity-type connector motions.

For motions prescribed using a predefined displacement field, the default amplitude variation is a ramp function for all procedure types; the default amplitude is a step function when using a predefined velocity field for all procedures except steady-state transport.

The default amplitude variation is a step function for fluid flux loading in all procedure types.

When a displacement or rotation boundary condition is removed, the corresponding reaction force or moment is reduced to zero according to the amplitude defined for the step. When film or radiation loads are removed, the variation is always a step function.

Prescribing nondefault amplitude variations

You can define complicated time variations of loadings, boundary conditions, and predefined fields by referring to an amplitude curve in the prescribed condition definition (see "Amplitude curves," Section 34.1.2). User subroutines are also provided in Abaqus/Standard and Abaqus/Explicit for coding general loadings (see "User subroutines: overview," Section 18.1.1).

In Abaqus/Standard you can change the default amplitude variation for a step (except the removal of film or radiation loads, as noted above).

Input File Usage: In Abaqus/Standard use the following option to change the default amplitude variation for a step:

*STEP, AMPLITUDE=STEP or RAMP

Abaqus/CAE Usage: In Abaqus/Standard use the following input to change the default amplitude variation for a step:

Step module: step editor: **Other**: **Default load variation with time: Instantaneous** or **Ramp linearly over step**

Boundary conditions in Abaqus/Explicit

Boundary conditions applied during an explicit dynamic response step should use appropriate amplitude references to define the time variation. If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference will be ignored, and a zero velocity boundary condition will be enforced.

Prescribing nondefault amplitude variations in transient procedures in Abaqus/Standard

The default amplitude is a step function for transient analysis procedures (fully coupled thermal-stress, fully coupled thermal-electrical-structural, coupled thermal-electrical, direct-integration dynamic, uncoupled heat transfer, and mass diffusion). Care should be exercised when the nondefault ramp amplitude variation is specified for transient analysis procedures since unexpected results may occur. For example, if a step of a transient heat transfer analysis uses the ramp amplitude variation and temperature boundary conditions are removed in a subsequent step, the reaction fluxes generated in the previous step will be ramped to zero from their initial values over the duration of the step. Therefore, heat flux will continue to flow through the affected boundary nodes over the entire subsequent step even though the temperature boundary conditions were removed.

Incrementation

Each step in an Abaqus analysis is divided into multiple increments. In most cases you have two choices for controlling the solution: automatic time incrementation or user-specified fixed time incrementation. Automatic incrementation is recommended for most cases. The methods for selecting automatic or direct incrementation are discussed in the individual procedure sections.

The issues associated with time incrementation in Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD analyses are quite different. The time increments are generally much smaller in Abaqus/Explicit than in Abaqus/Standard, while the time increments for Abaqus/CFD may be similar to those in Abaqus/Standard in many situations.

Incrementation in Abaqus/Standard

In nonlinear problems Abaqus/Standard will increment and iterate as necessary to analyze a step, depending on the severity of the nonlinearity. In transient cases with a physical time scale, you can provide parameters to indicate a level of accuracy in the time integration, and Abaqus/Standard will choose the time increments to achieve this accuracy. Direct user control is provided because it can sometimes save computational cost in cases where you are familiar with the problem and know a suitable incrementation scheme. Direct control can also occasionally be useful when automatic control has trouble with convergence in nonlinear problems.

Specifying the maximum number of increments

You can define the upper limit to the number of increments in an Abaqus/Standard analysis. In a direct cyclic analysis procedure, this upper limit should be set to the maximum number of increments in a single loading cycle. The default is 100. The analysis will stop if this maximum is exceeded before the complete solution for the step has been obtained. To arrive at a solution, it is often necessary to increase the number of increments allowed by defining a new upper limit.

Input File Usage: *STEP, INC=*n*

Abaqus/CAE Usage: Step module: step editor: **Incrementation**: **Maximum number of increments**

Extrapolation of the solution

In nonlinear analyses Abaqus/Standard uses extrapolation to speed up the solution. Extrapolation refers to the method used to determine the first guess to the incremental solution. The guess is determined by the size of the current time increment and by whether linear, displacement-based parabolic, velocity-based parabolic, or no extrapolation of the previously attained history of each solution variable is chosen. Displacement-based parabolic extrapolation is not relevant for Riks analyses, and velocity-based parabolic extrapolation is available only for direct-integration dynamic procedures. Linear extrapolation (the default for all procedures other than a direct-integration dynamic procedure using the transient fidelity application setting) uses 100% extrapolation (1% for the Riks method) of the previous incremental solution at the start of each increment to begin the nonlinear equation solution for the next increment. No extrapolation is used in the first increment of a step.

In some cases extrapolation can cause Abaqus/Standard to iterate excessively; some common examples are abrupt changes in the load magnitudes or boundary conditions and if unloading occurs as a result of cracking (in concrete models) or buckling. In such cases you should suppress extrapolation.

Displacement-based parabolic extrapolation uses two previous incremental solutions to obtain the first guess to the current incremental solution. This type of extrapolation is useful in situations when the local variation of the solution with respect to the time scale of the problem is expected to be quadratic,

such as the large rotation of structures. If parabolic extrapolation is used in a step, it begins after the second increment of the step: the first increment employs no extrapolation, and the second increment employs linear extrapolation. Consequently, slower convergence rates may occur during the first two increments of the succeeding steps in a multistep analysis.

Velocity-based parabolic extrapolation uses the previous displacement incremental solution to obtain the first guess to the current incremental solution. It is available only for direct-integration dynamic procedures, and it is the default if the transient fidelity application setting is specified as part of this procedure (see "Implicit dynamic analysis using direct integration," Section 6.3.2). This type of extrapolation is useful in situations with smooth solutions—i.e., when velocities do not display so called "saw tooth" patterns—and in such cases it may provide a better first guess than other extrapolations. If velocity-based parabolic extrapolation is used in a step, it begins after the first increment of the step; the first increment employs initial velocities.

Input File Usage: Use the following option to choose linear extrapolation:

*STEP, EXTRAPOLATION=LINEAR (default for all procedures other than a direct-integration dynamic procedure using the transient fidelity application setting)

Use the following option to choose displacement-based parabolic extrapolation:

*STEP, EXTRAPOLATION=PARABOLIC

Use the following option to choose velocity-based parabolic extrapolation:

*STEP, EXTRAPOLATION=VELOCITY PARABOLIC (default for a direct-integration dynamic procedure using the transient fidelity application setting)

Use the following option to choose no extrapolation:

*STEP, EXTRAPOLATION= NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Extrapolation of previous state at start of each increment: Linear**, **Parabolic**, **Velocity parabolic**, **None**, or **Analysis product default**

Incrementation in Abaqus/Explicit

The time increment used in an Abaqus/Explicit analysis must be smaller than the stability limit of the central-difference operator (see "Explicit dynamic analysis," Section 6.3.3); failure to use a small enough time increment will result in an unstable solution. Although the time increments chosen by Abaqus/Explicit generally satisfy the stability criterion, user control over the size of the time increment is provided to reduce the chance of a solution going unstable. The small increments characteristic of an explicit dynamic analysis product make Abaqus/Explicit well suited for nonlinear analysis.

Severe discontinuities in Abaqus/Standard

Abaqus/Standard distinguishes between regular, equilibrium iterations (in which the solution varies smoothly) and severe discontinuity iterations (SDIs) in which abrupt changes in stiffness occur. The most common of such severe discontinuities involve open-close changes in contact and stick-slip

changes in friction. By default, Abaqus/Standard will continue to iterate until the severe discontinuities are sufficiently small (or no severe discontinuities occur) and the equilibrium (flux) tolerances are satisfied. Alternatively, you can choose a different approach in which Abaqus/Standard will continue to iterate until no severe discontinuities occur.

For contact openings with the default approach, a force discontinuity is generated when the contact force is set to zero, and this force discontinuity leads to force residuals that are checked against the time average force in the usual way, as described in "Convergence criteria for nonlinear problems," Section 7.2.3. Similarly, in stick-to-slip transitions the frictional force is set to a lower value, which also leads to force residuals.

For contact closures a severe discontinuity is considered sufficiently small if the penetration error is smaller than the contact compatibility tolerance times the incremental displacement. The penetration error is defined as the difference between the actual penetration and the penetration following from the contact pressure and pressure-overclosure relation. In cases where the displacement increment is essentially zero, a "zero penetration" check is used, similar to the check used for zero displacement increments (see "Convergence criteria for nonlinear problems," Section 7.2.3). The same checks are used for slip-to-stick transitions in Lagrange friction.

To make sure that sufficient accuracy is obtained for contact between hard bodies, it is also required that the estimated contact force error is smaller than the time average force times the contact force error tolerance. The estimated contact force error is obtained by multiplying the penetration by an effective stiffness. For hard contact this effective stiffness is equal to the stiffness of the underlying element, whereas for softened/penalty contact the effective stiffness is obtained by adding the compliance of the contact constraint and the underlying element.

Forcing the iteration process to continue until no severe discontinuities occur is the more traditional, conservative method. However, this method can sometimes lead to convergence problems, particularly in large problems with many contact points or situations where contact conditions are only weakly determined. In such cases excessive iteration may occur and convergence may not be obtained

Input File Usage: *STEP, CONVERT SDI=NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Convert severe discontinuity iterations: Off**

Matrix storage and solution scheme in Abaqus/Standard

Abaqus/Standard generally uses Newton's method to solve nonlinear problems and the stiffness method to solve linear problems. In both cases the stiffness matrix is needed. In some problems—for example, with Coulomb friction—this matrix is not symmetric. Abaqus/Standard will automatically choose whether a symmetric or unsymmetric matrix storage and solution scheme should be used based on the model and step definition used. In some cases you can override this choice; the rules are explained below.

Usually it is not necessary to specify the matrix storage and solution scheme. The choice is available to improve computational efficiency in those cases where you judge that the default value is not the best choice. In certain cases where the exact tangent stiffness matrix is not symmetric, the extra iterations required by a symmetric approximation to the tangent matrix use less computer time than

solving the nonsymmetric tangent matrix at each iteration. Therefore, for example, Abaqus/Standard invokes the symmetric matrix storage and solution scheme automatically in problems with Coulomb friction where every friction coefficient is less than or equal to 0.2, even though the resulting tangent matrix will have some nonsymmetric terms. However, if any friction coefficient is greater than 0.2, Abaqus/Standard will use the unsymmetric matrix storage and solution scheme automatically since it may significantly improve the convergence history. This choice of the unsymmetric matrix storage and solution scheme will consider changes to the friction model. Thus, if you modify the friction definition during the analysis to introduce a friction coefficient greater than 0.2, Abaqus/Standard will activate the unsymmetric matrix storage and solution scheme automatically. In cases in which the unsymmetric matrix storage and solution scheme is selected automatically, you must explicitly turn it off if so desired; it is recommended to do so if friction prevents any sliding motions.

Input File Usage: *STEP, UNSYMM=YES or NO

Abaqus/CAE Usage: Step module: step editor: **Other**: **Storage: Use solver default** or **Unsymmetric** or **Symmetric**

Rules for using the unsymmetric matrix storage and solution scheme

The following rules apply to matrix storage and solution schemes in Abaqus/Standard:

1. Since Abaqus/Standard provides eigenvalue extraction only for symmetric matrices, steps with eigenfrequency extraction or eigenvalue buckling prediction procedures always use the symmetric matrix storage and solution scheme. You cannot change this setting. In such steps Abaqus/Standard will symmetrize all contributions to the stiffness matrix.
2. In all steps except those with eigenfrequency extraction or eigenvalue buckling procedures, Abaqus/Standard uses the unsymmetric matrix storage and solution scheme when any of the following features are included in the model. You cannot change this setting.
   a. Heat transfer convection/diffusion elements (element types DCC*xxx*)
   b. General shell sections with unsymmetric section stiffness matrices (“Three-dimensional conventional shell element library,” Section 29.6.7)
   c. User-defined elements with unsymmetric element matrices (“User-defined elements,” Section 32.17.1)
   d. User-defined material models with unsymmetric material stiffness matrices (“User-defined mechanical material behavior,” Section 26.7.1, or “User-defined thermal material behavior,” Section 26.7.2)
   e. User-defined surface interaction models with unsymmetric interface stiffness matrices (“User-defined interfacial constitutive behavior,” Section 37.1.6)
3. The following features all trigger the unsymmetric matrix storage and solution scheme for the step. You cannot change this setting.
   a. Fully coupled thermal-stress analysis, except when a separated solution scheme is specified for the step (“Fully coupled thermal-stress analysis,” Section 6.5.3)

b. Coupled thermal-electrical analysis, except when a separated solution scheme is specified for the step ("Coupled thermal-electrical analysis," Section 6.7.3)

c. Fully coupled thermal-electrical-structural analysis ("Fully coupled thermal-electrical-structural analysis," Section 6.7.4)

d. Coupled pore fluid diffusion/stress analysis with absorption or exsorption behavior ("Coupled pore fluid diffusion and stress analysis," Section 6.8.1)

e. Coupled pore fluid diffusion/stress analysis (steady-state)

f. Coupled pore fluid diffusion/stress analysis (transient with gravity loading)

g. Mass diffusion analysis ("Mass diffusion analysis," Section 6.9.1)

h. Radiation view factor calculation controls ("Cavity radiation," Section 41.1.1)

4. By default, the unsymmetric matrix storage and solution scheme is used for the complex eigenvalue extraction procedure. You can change this setting.

5. In all other cases you can control whether a symmetric or a full matrix storage and arithmetic solution is chosen. If you do not specify the matrix storage and solution scheme, Abaqus/Standard utilizes the value used in the previous general analysis step.

6. If you do not specify the matrix storage and solution scheme in the first step of an analysis, Abaqus/Standard will choose the unsymmetric scheme when any of the following are used:

   a. Any Abaqus/Aqua load type

   b. The concrete damaged plasticity material model

   c. Friction with a friction coefficient greater than 0.2

   The default value in the first step is the symmetric scheme for all other cases, except those covered by rules 2 and 3 above and for cases in which a friction coefficient is increased above 0.2 after the first step.

7. For radiative heat transfer surface interactions ("Thermal contact properties," Section 37.2.1), certain follower forces (such as concentrated follower forces or moments), three-dimensional finite-sliding analyses, any finite sliding in coupled pore fluid diffusion/stress analyses, and certain material models (particularly nonassociated flow plasticity models and concrete) introduce unsymmetric terms in the model's stiffness matrix. However, Abaqus/Standard does not automatically use the unsymmetric matrix storage and solution scheme when radiative heat transfer surface interactions are used. Specifying that the unsymmetric scheme should be used can sometimes improve convergence in such cases.

8. Coupled structural-acoustic and uncoupled acoustic analysis procedures in Abaqus/Standard generally use symmetric matrix storage and solution. Exceptions are the subspace-based steady-state dynamics or complex frequency procedures used for coupled structural-acoustic problems, where unsymmetric matrices are a consequence of the coupling procedure used in these cases. Using acoustic infinite elements or the acoustic flow velocity option triggers the unsymmetric matrix storage and solution scheme in Abaqus/Standard, except for natural frequency extraction using the Lanczos eigensolver, which uses symmetric matrix operations.

Precision level of the Abaqus/Explicit executable

You can choose a double-precision executable (with 64-bit word lengths) for Abaqus/Explicit on machines with a default, single-precision word length of 32 bits (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2). Most new computers have 32-bit default word lengths even though they may have 64-bit memory addressing. The single-precision executable typically results in a CPU savings of 20% to 30% compared to the double-precision executable, and single precision provides accurate results in most cases. Exceptions in which single precision tends to be inadequate include analyses that require greater than approximately 300,000 increments, have typical nodal displacement increments less than 10^{-6} times the corresponding nodal coordinate values, include hyperelastic materials, or involve multiple revolutions of deformable parts; the double-precision executable is recommended in these cases (for example, see "Simulation of propeller rotation," Section 2.3.15 of the Abaqus Benchmarks Guide).

You can also run only a part of Abaqus/Explicit using double precision, while using single precision for the rest (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2). These options are described below.

- If **double**=**`explicit`** is used or the **double** option is specified without a value, the Abaqus/Explicit analysis will run in double precision, while the packager will run in single precision. While this choice would satisfy higher precision needs in most analyses, the data are written to the state (**`.abq`**) file in single precision. Moreover, analysis-related computations performed in the packager will still be executed in single precision. Thus, new steps, restart, and import analyses will commence from data that are stored/computed in single precision despite the fact that calculations during the step are performed in double precision. Thus, in general, one can expect somewhat noisy solutions at the beginning of the first step, at step transitions, upon restart, and after import.
- If **double**=**`both`** is used, both the Abaqus/Explicit packager and analysis will run in double precision. This is the most expensive option but will ensure the highest overall execution precision. Analysis database floating point data will be written to the state (**`.abq`**) file at the end of packager or of a given step in double precision, thus ensuring in most cases the smoothest transition at step boundaries, upon restart, and after an import.
- There may be cases where the default single precision analysis is inadequate, while the **double**=**`both`** option is too expensive. These are typically models that have complex links of constraints (such as a complex mechanism with connector elements, complex combinations of distributed/kinematic couplings, tie constraints and multi-point constraints, or interactions of such constraints with boundary conditions). For such models it is desirable to solve only the constraints in the model in double precision while the rest of the model is solved in single precision. This combination gives the desired accuracy of the solution while increasing performance compared to a full double precision analysis.
- If **double**=**`constraint`** is used, the constraint packager and constraint solver are executed in double precision, while the remainder of the Abaqus/Explicit packager and analysis are executed in single precision.

- If **double**=**`off`** is used or the **double** option is omitted (default), both the Abaqus/Explicit packager and the analysis will run in single precision. The **double**=**`off`** option is useful when you want to override the setting in the environment file.

The significance of the precision level is indicated by comparing the solutions obtained with single and double precision. If no significant difference is found between single- and double-precision solutions for a particular model, the single-precision executable can be deemed adequate.

6.1.3 GENERAL AND LINEAR PERTURBATION PROCEDURES

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Linear and nonlinear procedures," Section 14.3.2 of the Abaqus/CAE User's Guide

Overview

An analysis step during which the response can be either linear or nonlinear is called a *general* analysis step. An analysis step during which the response can be linear only is called a *linear perturbation* analysis step. General analysis steps can be included in an Abaqus/Standard or Abaqus/Explicit analysis; linear perturbation analysis steps are available only in Abaqus/Standard.

A clear distinction is made in Abaqus/Standard between general analysis and linear perturbation analysis procedures. Loading conditions are defined differently for the two cases, time measures are different, and the results should be interpreted differently. These distinctions are defined in this section.

Abaqus/Standard treats a linear perturbation analysis as a linear perturbation about a preloaded, predeformed state. Abaqus/Foundation, a subset of Abaqus/Standard, is limited entirely to linear perturbation analysis but does not allow preloading or predeformed states.

General analysis steps

A general analysis step is one in which the effects of any nonlinearities present in the model can be included. The starting condition for each general step is the ending condition from the last general step, with the state of the model evolving throughout the history of general analysis steps as it responds to the history of loading. If the first step of the analysis is a general step, the initial conditions for the step can be specified directly ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Abaqus always considers *total time* to increase throughout a general analysis. Each step also has its own *step time*, which begins at zero in each step. If the analysis procedure for the step has a physical time scale, as in a dynamic analysis, step time must correspond to that physical time. Otherwise, step time is any convenient time scale—for example, 0.0 to 1.0—for the step. The step times of all general analysis steps accumulate into total time. Therefore, if an option such as creep (available only in Abaqus/Standard) whose formulation depends on total time is used in a multistep analysis, any steps that do not have a physical time scale should have a negligibly small step time compared to the steps in which a physical time scale does exist.

Sources of nonlinearity

Nonlinear stress analysis problems can contain up to three sources of nonlinearity: material nonlinearity, geometric nonlinearity, and boundary nonlinearity.

Material nonlinearity

Abaqus offers models for a wide range of nonlinear material behaviors (see "Combining material behaviors," Section 21.1.3). Many of the materials are history dependent: the material's response at any time depends on what has happened to it at previous times. Thus, the solution must be obtained by following the actual loading sequence. The general analysis procedures are designed with this in view.

Geometric nonlinearity

It is possible in Abaqus to define a problem as a "small-displacement" analysis, which means that geometric nonlinearity is ignored in the element calculations—the kinematic relationships are linearized. By default, large displacements and rotations are accounted for in contact constraints even if the small-displacement element formulations are used for the analysis; i.e., a large-sliding contact tracking algorithm is used (see "Contact formulations in Abaqus/Standard," Section 38.1.1, and "Contact formulations for contact pairs in Abaqus/Explicit," Section 38.2.2). The elements in a small-displacement analysis are formulated in the reference (original) configuration, using original nodal coordinates. The errors in such an approximation are of the order of the strains and rotations compared to unity. The approximation also eliminates any possibility of capturing bifurcation buckling, which is sometimes a critical aspect of a structure's response (see "Unstable collapse and postbuckling analysis," Section 6.2.4). You must consider these issues when interpreting the results of such an analysis.

The alternative to a "small-displacement" analysis in Abaqus is to include large-displacement effects. In this case most elements are formulated in the current configuration using current nodal positions. Elements therefore distort from their original shapes as the deformation increases. With sufficiently large deformations, the elements may become so distorted that they are no longer suitable for use; for example, the volume of the element at an integration point may become negative. In this situation Abaqus will issue a warning message indicating the problem. In addition, Abaqus/Standard will cut back the time increment before making further attempts to continue the solution. Abaqus/Explicit also offers element failure models to allow elements that reach high strains to be removed from a model; see "Dynamic failure models," Section 23.2.8, for details.

For each step of an analysis you specify whether a small- or large-displacement formulation should be used (i.e., whether geometric nonlinearity should be ignored or included). By default, Abaqus/Standard uses a small-displacement formulation and Abaqus/Explicit uses a large-displacement formulation. The default value for the formulation in an import analysis is the same as the value at the time of import. If a large-displacement formulation is used during any step of an analysis, it will be used in all following steps in the analysis; there is no way to turn it off.

Almost all of the elements in Abaqus use a fully nonlinear formulation. The exceptions are the cubic beam elements in Abaqus/Standard and the small-strain shell elements (those shell elements other than S3/S3R, S4, S4R, and the axisymmetric shells) in which the cross-sectional thickness change is ignored so that these elements are appropriate only for large rotations and small strains. Except for these elements, the strains and rotations can be arbitrarily large.

The calculated stress is the "true" (Cauchy) stress. For beam, pipe, and shell elements the stress components are given in local directions that rotate with the material. For all other elements the stress

components are given in the global directions unless a local orientation ("Orientations," Section 2.2.5) is used at a point. For small-displacement analysis the infinitesimal strain measure is used, which is output with the strain output variable E; strain output specified with output variables LE and NE is the same as with E.

Input File Usage: Use the following option to specify that a large-displacement formulation should be used for the step:

*STEP, NLGEOM=YES (default in Abaqus/Explicit)

Use the following option to specify that a small-displacement formulation should be used for the step:

*STEP, NLGEOM=NO (default in Abaqus/Standard)

Omitting the NLGEOM parameter is equivalent to using the default value.

Abaqus/CAE Usage: Step module: **Create Step**: select any step type: **Basic**: **Nlgeom**: **Off** (for a small-displacement formulation) or **On** (for a large-displacement formulation)

Boundary nonlinearity

Contact problems are a common source of nonlinearity in stress analysis—see "Contact interaction analysis: overview," Section 36.1.1. Other sources of boundary nonlinearity are nonlinear elastic springs, films, radiation, multi-point constraints, etc.

Loading

In a general analysis step the loads must be defined as total values. The rules for applying loads in a general, multistep analysis are defined in "Applying loads: overview," Section 34.4.1.

Incrementation

The general analysis procedures in Abaqus offer two approaches for controlling incrementation. Automatic control is one choice: you define the step and, in some procedures, specify certain tolerances or error measures. Abaqus then automatically selects the increment size as it develops the response in the step. Direct user control of increment size is the alternative approach, whereby you specify the incrementation scheme. The direct approach is sometimes useful in repetitive analyses with Abaqus/Standard, where you have a good "feel" for the convergence behavior of the problem. The methods for selecting automatic or direct incrementation are discussed in the individual procedure sections.

In nonlinear problems in Abaqus/Standard the challenge is always to obtain a convergent solution in the least possible computational time. In these cases automatic control of the time increment is usually more efficient because Abaqus/Standard can react to nonlinear response that you cannot predict ahead of time. Automatic control is particularly valuable in cases where the response or load varies widely through the step, as is often the case in diffusion-type problems such as creep, heat transfer, and consolidation. Ultimately, automatic control allows nonlinear problems to be run with confidence in Abaqus/Standard without extensive experience with the problem.

Strong nonlinearities typically do not present difficulties in Abaqus/Explicit because of the small time increments that are characteristic of an explicit dynamic analysis product.

Stabilization of unstable problems in Abaqus/Standard

Some static problems can be naturally unstable, for a variety of reasons.

Unconstrained rigid body motions

Instability may occur because unconstrained rigid body motions exist. Abaqus/Standard may be able to handle this type of problem with automatic viscous damping (see "Adjusting contact controls in Abaqus/Standard," Section 36.3.6) when rigid body motions exist during the approach of two bodies that will eventually come into contact.

Input File Usage: Use one of the following options:

*CONTACT STABILIZATION
*CONTACT CONTROLS, STABILIZE

Abaqus/CAE Usage: Automatic viscous damping is not supported in Abaqus/CAE.

Localized buckling behavior or material instability

Instability may also be caused by localized buckling behavior or by material instability; such instabilities are especially significant when no time-dependent behavior exists in the material modeling. The static, general analysis procedures in Abaqus/Standard can stabilize this type of problem if you request it (see "Static stress analysis," Section 6.2.2; "Quasi-static analysis," Section 6.2.5; "Steady-state transport analysis," Section 6.4.1; "Fully coupled thermal-stress analysis," Section 6.5.3; "Fully coupled thermal-electrical-structural analysis," Section 6.7.4; or "Coupled pore fluid diffusion and stress analysis," Section 6.8.1).

Input File Usage: Use one of the following options:

*STATIC, STABILIZE
*VISCO, STABILIZE
*STEADY STATE TRANSPORT, STABILIZE
*COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, STABILIZE
*SOILS, CONSOLIDATION, STABILIZE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: *any valid step type*: **Basic**: **Use stabilization with dissipated energy fraction**

Linear perturbation analysis steps

Linear perturbation analysis steps are available only in Abaqus/Standard (Abaqus/Foundation is essentially the linear perturbation functionality in Abaqus/Standard). The response in a linear analysis step is the linear perturbation response about the *base state*. The base state is the current state of the model at the end of the last general analysis step prior to the linear perturbation step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions ("Initial

conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). In Abaqus/Foundation the base state is always determined from the initial state of the model.

Linear perturbation analyses can be performed from time to time during a fully nonlinear analysis by including the linear perturbation steps between the general response steps. The linear perturbation response has no effect as the general analysis is continued. The step time of linear perturbation steps, which is taken arbitrarily to be a very small number, is never accumulated into the total time. A simple example of this method is the determination of the natural frequencies of a violin string under increasing tension (see "Vibration of a cable under tension," Section 1.4.3 of the Abaqus Benchmarks Guide). The tension of the string is increased in several geometrically nonlinear analysis steps. After each of these steps, the frequencies can be extracted in a linear perturbation analysis step.

If geometric nonlinearity is included in the general analysis upon which a linear perturbation study is based, stress stiffening or softening effects and load stiffness effects (from pressure and other follower forces) are included in the linear perturbation analysis.

Load stiffness contributions are also generated for centrifugal and Coriolis loading. In direct steady-state dynamic analysis Coriolis loading generates an imaginary antisymmetric matrix. This contribution is accounted for currently in solid and truss elements only and is activated by using the unsymmetric matrix storage and solution scheme in the step.

Linear perturbation procedures

The following purely linear perturbation procedures are available in Abaqus/Standard:

- "Eigenvalue buckling prediction," Section 6.2.3
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Complex eigenvalue extraction," Section 6.3.6
- "Transient modal dynamic analysis," Section 6.3.7
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Subspace-based steady-state dynamic analysis," Section 6.3.9
- "Response spectrum analysis," Section 6.3.10
- "Random response analysis," Section 6.3.11
- "Time-harmonic analysis" in "Eddy current analysis," Section 6.7.5

In addition, the following analysis techniques are treated as linear perturbation steps in an analysis:

- "Defining substructures," Section 10.1.2
- "Generating matrices," Section 10.3.1

Except for these procedures and the static procedure (explained below), all other procedures can be used only in general analysis steps (in other words, they are not available with Abaqus/Foundation). All linear perturbation procedures except for the complex eigenvalue extraction procedure are available with Abaqus/Foundation.

Linear static perturbation analysis

A linear static stress analysis ("Static stress analysis," Section 6.2.2) can be conducted in Abaqus/Standard.

Input File Usage: Use both of the following options to conduct a linear static perturbation analysis:

```
*STEP, PERTURBATION
*STATIC
```

Omitting the PERTURBATION parameter on the *STEP option implies that a general static analysis is required.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Static, Linear perturbation**

Loading

Load magnitudes (including the magnitudes of prescribed boundary conditions) during a linear perturbation analysis step are defined as the magnitudes of the load perturbations only. Likewise, the value of any solution variable is output as the perturbation value only—the value of the variable in the base state is not included.

Multiple load case analysis

Multiple load cases can be analyzed simultaneously for static, direct-solution steady-state dynamic and SIM-based steady-state dynamic (including subspace projection) linear perturbation steps. See "Multiple load case analysis," Section 6.1.4, for a description of this capability.

Restrictions

A linear perturbation analysis is subject to the following restrictions:

- Since a linear perturbation analysis has no time period, amplitude references ("Amplitude curves," Section 34.1.2) can be used meaningfully only to specify loads or boundary conditions as functions of frequency (in a steady-state dynamics analysis) or to define base motion (in mode-based dynamics procedures). If loads or boundary conditions are specified as functions of time, the amplitude value corresponding to time=0 will be used.
- A general implicit dynamic analysis ("Implicit dynamic analysis using direct integration," Section 6.3.2) cannot be interrupted to perform perturbation analyses: before performing the perturbation analysis, Abaqus/Standard requires that the structure be brought into static equilibrium.
- During a linear perturbation analysis step, the model's response is defined by its linear elastic stiffness at the base state. For viscoelastic materials the elastic moduli at the base state is evaluated as described in "Materials options" in "Static stress analysis," Section 6.2.2, for static procedures, and in "Evaluating frequency-dependent material properties" in "Natural frequency extraction," Section 6.3.5, for frequency-based procedures. Plasticity and other inelastic effects are ignored

during a linear perturbation analysis step. For hyperelasticity (“Hyperelastic behavior of rubberlike materials,” Section 22.5.1) or hypoelasticity (“Hypoelastic behavior,” Section 22.4.1), the tangent elastic moduli in the base state are used. If cracking has occurred—for example, in the concrete model (“Concrete smeared cracking,” Section 23.6.1)—the damaged elastic (secant) moduli are used.

- Contact conditions cannot change during a linear perturbation analysis. The open/closed status of each contact constraint remains as it is in the base state. All points in contact (i.e., with a “closed” status) are assumed to be sticking if friction is present, except the contact nodes for which a velocity differential is imposed by the motion of the reference frame or the transport velocity. At those nodes, slipping conditions are assumed regardless of the friction coefficient.
- The effects of temperature and field variable perturbations are ignored for materials that are dependent on temperature and field variables. However, temperature perturbations will produce perturbations of thermal strain.

6.1.4 MULTIPLE LOAD CASE ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- *LOAD CASE
- *END LOAD CASE
- Chapter 34, "Load cases," of the Abaqus/CAE User's Guide

Overview

A multiple load case analysis:

- is used to study the linear responses of a structure subjected to distinct sets of loads and boundary conditions defined within a step (each set is referred to as a load case);
- can be much more efficient than an equivalent multiple perturbation step analysis;
- allows for the changing of mechanical loads and boundary conditions from load case to load case;
- includes the effects of the base state; and
- can be performed with static perturbation, direct-solution steady-state dynamic and SIM-based steady-state dynamic analyses.

Load cases

A load case refers to a set of loads, boundary conditions, and base motions comprising a particular loading condition. For example, in a simplified model the operational environment of an airplane might be broken into five load cases: (1) take-off, (2) climb, (3) cruise, (4) descent, and (5) landing. Often a load case is defined in terms of unit loads or prescribed boundary conditions, and a multiple load case analysis refers to the simultaneous solution for the responses of each load case in a set of such load cases. These responses can then be scaled and linearly combined during postprocessing to represent the actual loading environment. Other postprocessing manipulations on load cases are also common, such as finding the maximum Mises stress among all load cases. These types of load case manipulations can be requested in the Visualization module of Abaqus/CAE (see the Abaqus/CAE User's Guide).

Using multiple load cases

A multiple load case analysis is conceptually equivalent to a multiple step analysis in which the load case definitions are mapped to consecutive perturbation steps. However, a multiple load case analysis is generally much more efficient than the equivalent multiple step analysis. The exception occurs when a large number of boundary conditions exist that are not common to all load cases (i.e., degrees of freedom are constrained in one load case but not others). It is difficult to define what "large" is since it is model dependent. The relative performance of the two analysis methods can be assessed by performing a data

check analysis for both the multiple load case analysis and the equivalent multiple step analysis. The data check analysis writes resource information for each step to the data file, including the maximum wavefront, number of floating point operations, and minimum memory required. If these numbers are noticeably larger for the multiple load case step compared to those across all steps of the equivalent multiple step analysis (the number of floating point operations should be summed over all steps before comparing), the multiple step analysis will be more efficient.

Although generally more efficient, the multiple load case analysis may consume more memory and disk space than an equivalent multiple step analysis. Thus, for large problems or problems with many load cases it is again advisable, as described above, to compare resource usage between the multiple load case analysis and the equivalent multiple step analysis. If resource requirements for the multiple load case analysis are deemed too large, consider dividing the load cases among a few steps. The resulting analysis (a hybrid of multiple load cases and multiple steps) will require fewer resources while retaining an efficiency advantage over an equivalent pure multiple step analysis.

Defining load cases

You define a load case within a static perturbation, direct-solution steady-state dynamic, and SIM-based steady-state dynamic analyses. Load case definitions do not propagate to subsequent steps. Only the following types of prescribed conditions can be specified within a load case definition:

- Boundary conditions
- Concentrated loads
- Distributed loads
- Distributed surface loads
- Inertia-based loads
- Base motions

Additional rules governing these prescribed conditions are described in the sections that follow. No other types of prescribed conditions can appear in a step that contains load case definitions. All other valid analysis components, such as output requests, must be specified outside load case definitions.

Each load case definition is assigned a name for postprocessing purposes.

Input File Usage: Use the first option to begin a load case and the second option to end a load case:

```
*LOAD CASE, NAME=name
*END LOAD CASE
```

Prescribed conditions specified within a load case definition apply only to that load case. In static perturbation and direct-solution steady-state dynamic analyses, prescribed conditions can be specified outside the load case definitions (in this case they apply to all load cases in the step).

Abaqus/CAE Usage: Load module: **Create Load Case**: **Name**: *name*

In Abaqus/CAE if a step contains load cases, all prescribed conditions in the step must be included in one or more load cases.

Procedures

Load cases can be defined only in perturbation steps with the following procedures:

- Static
- Direct-solution, steady-state dynamic
- SIM-based, steady-state dynamic

As with other perturbation steps, a multiple load case analysis will include the nonlinear effects of the previous general step (base state). The following analysis techniques are not supported in the context of a load case step:

- Restart from a particular load case
- Submodeling using results from other than the first load case in the global analysis
- Importing and transferring results
- Cyclic symmetry analysis
- Contour integrals
- Design sensitivity analysis

Boundary conditions

Boundary conditions can be specified both outside and inside load case definitions in the same step. Specifying a boundary condition outside the load case definitions in a step is equivalent to including it in all load case definitions in the step (i.e., the boundary condition will be applied to all load cases). Unless any boundary conditions are removed in the perturbation step, the boundary conditions that are active in the base state will propagate to all load cases in the perturbation step. If any boundary condition is removed in a step with load cases (either outside or inside load case definitions), the base state boundary conditions will not be propagated to any load case in the step. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1, for more information.

Note: In Abaqus/CAE if a step contains load cases, all boundary conditions in the step must be included in one or more load cases. Boundary conditions can only be used with load cases in static perturbation and direct-solution steady-state dynamic analyses.

Loads

In static perturbation and direct-solution steady-state dynamic analyses concentrated, distributed, and distributed surface loads can be specified both outside and inside load case definitions in the same step. Inertia relief loads can be specified either outside load case definitions or inside load case definitions in the same step but not both simultaneously. Specifying one of these load types outside the load case definitions in a step is equivalent to including it in all load case definitions in the step (i.e., the loading will be applied to all load cases).

In SIM-based steady-state dynamic analyses concentrated, distributed, distributed surface loads, and base motion can be specified only inside load case definitions in the same step. Inertia relief loads are not supported.

Load cases cannot be used in models that include aqua loads (see "Abaqus/Aqua analysis," Section 6.11.1).

As with any perturbation step, perturbation loads must be defined completely within the perturbation step (see "Applying loads: overview," Section 34.4.1).

Note: In Abaqus/CAE if a step contains load cases, all loads in the step must be included in one or more load cases.

Predefined fields

Field variables cannot be specified in a step with load cases.

Elements

Load cases cannot be used in models that include piezoelectric elements (see "Piezoelectric analysis," Section 6.7.2).

Output

In a step containing one or more load cases, only selected field and history output requests to the output database and output requests to the data file are supported. Output requests to the results file are not supported. Output requests can be specified only outside load case definitions, and they apply to all load cases in a step. The step propagation rules for output requests are the same as for other perturbation steps (see "Output," Section 4.1.1).

Element and energy history output variables are not available during a multiple load case analysis (see "Abaqus/Standard output variable identifiers," Section 4.2.1). Additional restrictions apply for a SIM-based steady-state dynamic analysis; see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1, for more information.

The available field output corresponding to each load case is stored in a separate frame on the output database with the load case name included as a frame attribute. To distinguish between load cases for history output variables, the name of the load case is appended to the history variable name. The Visualization module of Abaqus/CAE and the Abaqus Scripting Interface (see Chapter 9, "Using the Abaqus Scripting Interface to access an output database," of the Abaqus Scripting User's Guide) can be used to access and manipulate load case output. Abaqus/Standard does not perform consistency checks on the physical validity of the load case manipulations. For example, the linear superposition of two load cases, each with different boundary conditions, is allowed even though the combined results may not be physically meaningful.

Input file template

```
*HEADING
…
*STEP, PERTURBATION
```

STATIC** ***or ***STEADY STATE DYNAMICS, DIRECT**
…
***OUTPUT, FIELD**
…
***BOUNDARY**
Data lines to specify boundary conditions for all load cases.
***DLOAD**
Data lines to specify distributed loads for all load cases.
***CLOAD**
Data lines to specify point loads for all load cases.
***DSLOAD**
Data lines to specify distributed surface loads for all load cases.
***INERTIA RELIEF**
Data lines to specify inertia relief loading directions.
(This option cannot be used inside load cases if it is used here.)
…
LOAD CASE, NAME=name1*
***BOUNDARY**
Data lines to specify boundary conditions for first load case.
***DLOAD**
Data lines to specify distributed loads for first load case.
***CLOAD**
Data lines to specify point loads for first load case.
***DSLOAD**
Data lines to specify distributed surface loads for first load case.
***INERTIA RELIEF**
Data lines to specify inertia relief loading directions.
(This option cannot be used outside load cases if it is used here.)
***END LOAD CASE**
LOAD CASE, NAME=name2*
Load and boundary condition options for second load case
***END LOAD CASE**
…
Subsequent load case definitions
…
***END STEP**
***STEP, PERTURBATION**
FREQUENCY, SIM** ***or ***FREQUENCY, EIGENSOLVER=AMS**
***END STEP**
…
***STEP, PERTURBATION**
***STEADY STATE DYNAMICS**

LOAD CASE, NAME=name3*
***BASE MOTION**
Data lines to specify base motion for first load case.
***DLOAD**
Data lines to specify distributed loads for first load case.
***CLOAD**
Data lines to specify point loads for first load case.
***DSLOAD**
Data lines to specify distributed surface loads for first load case.
***END LOAD CASE**
LOAD CASE, NAME=name4*
Load and base motion options for second load case.
***END LOAD CASE**
…
Subsequent load case definitions
…
***OUTPUT, HISTORY**
…
***END STEP**

6.1.5 DIRECT LINEAR EQUATION SOLVER

Products: Abaqus/Standard Abaqus/CAE

References

- "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2
- "Using the Abaqus environment settings," Section 3.3.1
- "Iterative linear equation solver," Section 6.1.6
- "Parallel execution in Abaqus/Standard," Section 3.5.2
- "Configuring analysis procedure settings," Section 14.11 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Linear equation solution is used in linear and nonlinear analysis. In nonlinear analysis Abaqus/Standard uses the Newton method or a variant of it, such as the Riks method, within which it is necessary to solve a set of linear equations at each iteration. The direct linear equation solver finds the exact solution to this system of linear equations (up to machine precision). The direct linear equation solver in Abaqus/Standard:

- uses a sparse, direct, Gauss elimination method; and
- often represents the most time consuming part of the analysis (especially for large models)—the storage of the equations occupies the largest part of the disk space during the calculations.

The sparse solver

The direct sparse solver uses a "multifront" technique that can reduce the computational time to solve the equations dramatically if the equation system has a sparse structure. Such a matrix structure typically arises when the physical model is made from several parts or branches that are connected together; a spoked wheel is a good example of a structure that has a sparse stiffness matrix. Space frames and other structures modeled with beams, trusses, and shells often have sparse stiffness matrices. In contrast, a *blocky* structure—such as a single, solid, three-dimensional block (see "Elastic-plastic line spring modeling of a finite length cylinder with a part-through axial flaw," Section 1.4.3 of the Abaqus Example Problems Guide)—provides little opportunity for the sparse solver to reduce the computer time. For large blocky structures, the iterative linear equation solver may be more efficient (see "Iterative linear equation solver," Section 6.1.6).

Input File Usage: Use the following option to use the default direct sparse solver:

*STEP

Abaqus/CAE Usage: Step module: step editor: **Other**: **Method: Direct**

Setting controls for the direct linear solver

The linear equation solver can optimize elimination of constraint equations associated with hard contact and hybrid elements. There are two potential undesirable side-effects associated with this option:

- Possible small degradation of solution accuracy may adversely impact the nonlinear convergence behavior.
- Possible minor performance degradation for models without hard contact constraints and/or hybrid elements.

Input File Usage: Use the following option to turn on constraint optimization:

*SOLVER CONTROLS, CONSTRAINT OPTIMIZATION

Abaqus/CAE Usage: You cannot specify constraint optimization in Abaqus/CAE.

6.1.6 ITERATIVE LINEAR EQUATION SOLVER

Products: Abaqus/Standard Abaqus/CAE

References

- *STEP
- *SOLVER CONTROLS
- "Parallel execution in Abaqus/Standard," Section 3.5.2
- "Customizing solver controls," Section 14.15.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The iterative linear equation solver in Abaqus/Standard:

- can be used for linear and nonlinear static, quasi-static, heat transfer, geostatic, and coupled pore fluid diffusion and stress analysis solution procedures;
- should be used only for large, well-conditioned models for which the direct sparse solver (see "Direct linear equation solver," Section 6.1.5) requires a prohibitively large number of floating point operations;
- is likely to be dramatically faster than the direct equation solver for large, well-conditioned, blocky structures;
- runs totally in-core and uses less storage than the direct sparse solver (memory and disk combined);
- can be used only with three-dimensional models;
- must be the only solver invoked in the analysis (i.e., you cannot use the iterative solver in one step and the direct solver in another);
- cannot be used with automatic stabilization with an adaptive damping factor (see "Adaptive automatic stabilization scheme" in "Solving nonlinear problems," Section 7.1.1);
- can be used with a constant damping factor if stabilization is necessary (see "Automatic stabilization of static problems with a constant damping factor" in "Solving nonlinear problems," Section 7.1.1);
- cannot be used if the system of equations includes Lagrange multiplier degrees of freedom (i.e., associated with distributing couplings, hybrid elements, connector elements, contact with direct enforcement); and
- will degrade performance if used with models containing dense linear constraints (e.g., equations, kinematic couplings, MPCs) that eliminate a large number of slave degrees of freedom per master degree of freedom and/or eliminate some slave degrees of freedom in favor of a large number of master degrees of freedom.

Iterative solver basics

The iterative solver in Abaqus/Standard can be used to find the solution to a linear system of equations and can be invoked in a linear or nonlinear static, quasi-static, geostatic, pore fluid diffusion, or heat transfer analysis step. Since the technique is iterative, a converged solution to a given system of linear equations cannot be guaranteed. In cases where the iterative solver fails to converge to a solution, modifications to the model may be necessary to improve the convergence behavior. In some cases the only choice may be to use the direct solver to obtain a solution. When the iterative solver converges, the accuracy of this solution depends on the relative tolerance that is used; the default tolerance is sufficiently accurate for most purposes. However, tolerance adjustments for particular analyses may improve the overall performance of the simulation. In addition, the performance of the iterative solver relative to the direct sparse solver is highly sensitive to the model geometry, favoring blocky type structures (i.e., models that look more like a cube than a plate) with a high degree of mesh connectivity and a relatively low degree of sparsity. These types of models often demand the most computational and storage resources for the direct sparse solver. Models with a lesser degree of connectivity (often said to have a higher degree of sparsity), such as thin, shell-like structures, are much more suited to the direct sparse solver (see "Direct linear equation solver," Section 6.1.5).

Input File Usage: Use the following option to invoke the iterative solver:

*STEP, SOLVER=ITERATIVE

Abaqus/CAE Usage: Step module: step editor: **Other**: **Method: Iterative**

The iterative solution technique

The iterative solution technique in Abaqus/Standard is based on Krylov methods employing a preconditioner. This solver uses the following general strategy:

1. The Krylov method solver iterates on the system of equations generated by the finite element method while a preconditioner is applied at each iteration.
2. The preconditioner is calculated only once at the beginning of each linear system solve and is used to accelerate the convergence of the Krylov method.
3. In parallel, all components of the iterative solution process (including matrix assembly, preconditioner setup, and the actual solve using the Krylov method) are handled locally on each core with all necessary communication handled through an MPI-based implementation.

The process outlined above is performed entirely internal to Abaqus/Standard, with no user intervention required.

Convergence of the linear system of equations

To generate the solution to the system of linear algebraic equations (denoted by the matrix equation $Ku = f$, where $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{f}$ is the load vector, and $\boldsymbol{u}$ is the desired displacement solution), a sequence of Krylov solver iterations is performed, whereby an approximate solution gets closer to the exact solution at each iteration. The error in the approximate solution is measured by the relative residual of the linear system, defined by $||Ku - f||/||f||$, where $|| \cdot ||$ is the L^2 norm.

The term "convergence" is used to describe this process, and the approximate solution is said to be converged when the relative residual is below a specified tolerance. By default, this tolerance is 10^{-3} for general nonlinear procedures. Linear perturbation procedures have the default tolerance of 10^{-6} . While the default tolerance may seem loose for general nonlinear procedures, it is important to note that the linear solver convergence tolerance is independent from the nonlinear convergence process (i.e., Newton-Raphson method) tolerances that are used to determine if analysis increments converge. The latter are the same regardless of the choice of linear equation solver, iterative or direct.

The rate at which the approximate solution converges is directly related to the conditioning of the original system of equations. A linear system that is well conditioned will converge faster than an ill-conditioned system. If the residual does not converge to tolerance within the maximum number of iterations, the iterative solver is said to have encountered a non-convergence and Abaqus/Standard issues a warning message. However, the analysis will continue running and in some cases the Newton-Raphson iterations within increments may continue to converge.

Setting controls for the iterative linear solver

The default controls provided in Abaqus/Standard are usually sufficient. However, a method for overriding the default relative convergence tolerance and maximum number of solver iterations is provided.

Resetting the solver controls

You can specify that the solver controls be reset to their default values.

Input File Usage: *SOLVER CONTROLS, RESET

Abaqus/CAE Usage: Step module: **Other→Solver Controls→Edit**: **Reset all parameters to their system-defined defaults**

Specifying the relative convergence tolerance

By default, this tolerance is 10^{-3} for procedures other than linear perturbation. Linear perturbation procedures have the default tolerance of 10^{-6}. For nonlinear problems the accuracy of the linear solution can impact the convergence of the Newton method. In some cases it may be necessary to manually specify the iterative solver relative tolerance to improve the convergence of the Newton-Raphson method or to improve performance.

Input File Usage: *SOLVER CONTROLS
relative tolerance for convergence

Abaqus/CAE Usage: Step module: **Other→Solver Controls→Edit**: **Specify**: **Relative tolerance: Specify:** *relative tolerance for convergence*

Specifying the maximum number of solver iterations

In rare instances the linear solver may require more than the default number of iterations to converge to the desired level of accuracy. In this case you can increase the maximum number of iterations allowed by the iterative solver (the default value is 300).

Input File Usage: *SOLVER CONTROLS
, *max number of solver iterations*

Abaqus/CAE Usage: Step module: **Other→Solver Controls→Edit**: **Specify**: **Max. number of iterations: Specify:** *max number of solver iterations*

Specifying the incomplete factorization fill-in levels for soils and geostatic analyses

The preconditioner used for soils and geostatic analyses employs a factorization-based method, also known as ILU(k). In rare instances the linear solver may require more than the default number of incomplete factorization fill-in levels to converge to the desired accuracy level. Incomplete LU factorization of a matrix is a sparse approximation of the LU factorization. LU factorization typically changes the nonzero structure of the stiffness matrices by adding many nonzero entries; ILU factorization approximates the fully factorized matrices by limiting the number of nonzero entries introduced during the factorization. By default, the ILU factorization fill-in level used by the iterative solver is 0 and no nonzero entries are added. You can increase the fill-in level (maximum value is 3) to allow nonzero entries to be added based on the connectivity of the stiffness matrices and obtain a better approximation of the full factorization but with increased computational cost.

Input File Usage: *SOLVER CONTROLS
, , *ILU factorization fill-in level*

Abaqus/CAE Usage: Step module: **Other→Solver Controls→Edit**: **Specify**: **ILU factorization fill-in level: Specify:** *ILU factorization fill-in level*

Deciding to use the iterative solver

Many factors must be carefully weighed before deciding to use the iterative solver in Abaqus/Standard, such as element type, contact and constraint equations, material and geometric nonlinearities, and material properties, all of which can impact robustness and performance. In cases where the model is ill-conditioned the iterative solver may converge very slowly or fail to converge. This may occur, for example, if many elements have poor aspect ratios.

In addition to the robustness issues (relating mainly to the rate of convergence or stagnation), the iterative solver is expected to outperform the direct sparse solver only for blocky models (even when the model is well conditioned) that require a very large number of floating point operations for factorization. Typically, for a well-conditioned solid model, the number of degrees of freedom in the global model must be greater than one million before the iterative solver will be comparable to the direct solver in terms of run time.

Element type and model geometry

The most basic modeling issue that will affect the performance of the iterative solver is the model geometry, which must be carefully considered when deciding if the iterative solver is suited for a particular model. In general, models that are blocky in nature (i.e., look more like a cube than a plate) and are dominated by solid elements will behave well with the iterative solver. Although structural elements such as beams and shells are supported, models with structural elements will not perform

optimally; the direct sparse solver should be used instead for such models. Common modeling techniques such as coating solid elements with a thin layer of membrane elements to recover accurate stresses on the boundary or fixing rigid body motion with weak springs may not work with the iterative solver. Applying loads or boundary conditions to large node sets using locally transformed coordinate systems can also cause convergence difficulties. All of these techniques are likely to lead to extremely slow convergence or stagnation.

Another factor that can influence the convergence of the iterative solver is the quality of the elements. Blocky models, such as an engine block, that contain many poorly shaped elements with high aspect ratios can also lead to poor iterative solver convergence. It is a good idea to look for warning messages about poorly shaped elements when evaluating the performance of the iterative solver.

Currently, hybrid elements and connectors are not supported with the iterative solver.

Using cohesive elements with the iterative solver will likely lead to nonconvergence.

Constraint equations

Although the iterative solver can be used for models that include constraint equations (such as multi-point constraints, surface-based tie constraints, kinematic couplings, etc.), certain limitations may exist in the following situations:

- linear or nonlinear multi-point constraints containing more than a few thousand degrees of freedom;
- more than a few thousand linear or nonlinear multi-point constraints containing shared master degrees of freedom;
- rigid body definition of elements containing more than a few thousand degrees of freedom; or
- kinematic coupling constraints containing more than a few thousand slave degrees of freedom.

If any of these conditions apply to a model, the solution cost of the linear system of equations will grow linearly with the number of such constraints. Furthermore, it is usually recommended to tighten the iterative solver tolerance and increase the number of maximum iterations in the linear iterative solver for nonlinear analysis to achieve convergence. Therefore, it is recommended to keep such constraints to a minimum if possible; otherwise, the increased cost may offset the performance gains that come from using the iterative solver.

Distributing couplings are not supported with the iterative solver.

Contact

Since contact is a form of nonlinear analysis, special care must be taken in selecting the convergence tolerance for the iterative solver (see “Nonlinear analysis” below). Therefore, it is recommended to run the model through a static perturbation analysis before proceeding to the nonlinear problem. This will demonstrate how the iterative solver will perform for the specific model geometry without the added difficulty of nonlinear convergence.

The iterative solver will work only with the penalty-based contact formulation with reasonable penalty stiffness. If contact with direct enforcement (i.e., the Lagrange multiplier method) or penalty contact with an extremely high penalty stiffness is used, Abaqus/Standard may fail to converge. The iterative solver does not support pore fluid contact, regardless of the contact formulation used.

Material properties

When deciding to use the iterative solver, the variation of material properties in the model should be considered. Models that have very large discontinuities in material behavior (many orders of magnitude) will most likely converge slowly and possibly stagnate.

Nonlinear analysis

The iterative solver can be used to solve the linear system of algebraic equations that arises at each iteration of the Newton procedure. However, the convergence of the nonlinear problem will be affected by the convergence of the iterative linear solver. The actual impact depends on the particular model and type of nonlinearities present. In some cases the default iterative solver tolerance of 10^{-3} is sufficient to maintain the convergence of the Newton method; in other cases a smaller linear solver tolerance (for example, 10^{-6}) must be used.

If a nonlinear analysis that uses the iterative solver fails to converge, it is often difficult to determine if this is due to the approximate linear equation solution of the iterative solver or if the Newton process itself is failing to converge. If nonlinear convergence problems occur, the direct solver can be used—given the problem is solvable using the direct solver due to solution cost—to eliminate the approximate linear solution as a possible source of the problem.

6.2 Static stress/displacement analysis

- "Static stress analysis procedures: overview," Section 6.2.1
- "Static stress analysis," Section 6.2.2
- "Eigenvalue buckling prediction," Section 6.2.3
- "Unstable collapse and postbuckling analysis," Section 6.2.4
- "Quasi-static analysis," Section 6.2.5
- "Direct cyclic analysis," Section 6.2.6
- "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7

6.2.1 STATIC STRESS ANALYSIS PROCEDURES: OVERVIEW

A static stress procedure is one in which inertia effects are neglected. Several static stress analysis procedures are available in Abaqus/Standard:

- **Static analysis:** "Static stress analysis," Section 6.2.2, is used for stable problems and can include linear or nonlinear response.
- **Eigenvalue buckling analysis:** "Eigenvalue buckling prediction," Section 6.2.3, is used to estimate the critical (bifurcation) load of "stiff" structures. It is a linear perturbation procedure.
- **Unstable collapse and postbuckling analysis:** "Unstable collapse and postbuckling analysis," Section 6.2.4, is used to estimate the unstable, geometrically nonlinear collapse of a structure. The method can also be helpful in obtaining a solution in other types of unstable problems, and it is often suitable for limit load analyses.
- **Quasi-static analysis:** "Quasi-static analysis," Section 6.2.5, is used to analyze the transient response of structures considering time-dependent material behavior (creep and swelling, viscoelasticity, and viscoplasticity). A quasi-static analysis can be linear or nonlinear.
- **Direct cyclic analysis:** "Direct cyclic analysis," Section 6.2.6, is used to calculate the stabilized cyclic response of the structure directly. It uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized cyclic solution iteratively.
- **Low-cycle fatigue analysis:** "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7, is used to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding growth at the interfaces in laminated composites based on the direct cyclic approach in conjunction with the damage extrapolation technique.

6.2.2 STATIC STRESS ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Static stress analysis procedures: overview," Section 6.2.1
- *STATIC
- "Configuring a static, general procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A static stress analysis:

- is used when inertia effects can be neglected;
- can be linear or nonlinear; and
- ignores time-dependent material effects (creep, swelling, viscoelasticity) but takes rate-dependent plasticity and hysteretic behavior for hyperelastic materials into account.

Time period

During a static step you assign a time period to the analysis. This is necessary for cross-references to the amplitude options, which can be used to determine the variation of loads and other externally prescribed parameters during a step (see "Amplitude curves," Section 34.1.2). In some cases this time scale is quite real—for example, the response may be caused by temperatures varying with time based on a previous transient heat transfer run; or the material response may be rate dependent (rate-dependent plasticity), so that a natural time scale exists. Other cases do not have such a natural time scale; for example, when a vessel is pressurized up to limit load with rate-independent material response. If you do not specify a time period, Abaqus/Standard defaults to a time period in which "time" varies from 0.0 to 1.0 over the step. The "time" increments are then simply fractions of the total period of the step.

Linear static analysis

Linear static analysis involves the specification of load cases and appropriate boundary conditions. If all or part of a problem has linear response, substructuring is a powerful capability for reducing the computational cost of large analyses (see "Using substructures," Section 10.1.1).

Nonlinear static analysis

Nonlinearities can arise from large-displacement effects, material nonlinearity, and/or boundary nonlinearities such as contact and friction (see "General and linear perturbation procedures,"

Section 6.1.3) and must be accounted for. If geometrically nonlinear behavior is expected in a step, the large-displacement formulation should be used. In most nonlinear analyses the loading variations over the step follow a prescribed history such as a temperature transient or a prescribed displacement.

Input File Usage: Use the following option to specify that a large-displacement formulation should be used for a static step:

*STEP, NLGEOM

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Basic**: **Nlgeom**: **On** (to activate the large-displacement formulation)

Unstable problems

Some static problems can be naturally unstable, for a variety of reasons.

Buckling or collapse

In some geometrically nonlinear analyses, buckling or collapse may occur. In these cases a quasi-static solution can be obtained only if the magnitude of the load does not follow a prescribed history; it must be part of the solution. When the loading can be considered proportional (the loading over the complete structure can be scaled with a single parameter), a special approach—called the "modified Riks method"—can be used, as described in "Unstable collapse and postbuckling analysis," Section 6.2.4.

Input File Usage: *STATIC, RIKS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**

Local instabilities

In other unstable analyses the instabilities are local (e.g., surface wrinkling, material instability, or local buckling), in which case global load control methods such as the Riks method are not appropriate. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Incrementation

Abaqus/Standard uses Newton's method to solve the nonlinear equilibrium equations. Many problems involve history-dependent response; therefore, the solution usually is obtained as a series of increments, with iterations to obtain equilibrium within each increment. Increments must sometimes be kept small (in the sense that rotation and strain increments must be small) to ensure correct modeling of history-dependent effects. Most commonly the choice of increment size is a matter of computational efficiency: if the increments are too large, more iterations will be required. Furthermore, Newton's method has a finite radius of convergence; too large an increment can prevent any solution from being obtained because the initial state is too far away from the equilibrium state that is being sought—it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic incrementation

In most cases the default automatic incrementation scheme is preferred because it will select increment sizes based on computational efficiency.

Input File Usage: *STATIC

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Incrementation**: **Type**: **Automatic** (default)

Direct incrementation

Direct user control of the increment size is also provided because if you have considerable experience with a particular problem, you may be able to select a more economical approach.

Input File Usage: *STATIC, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Incrementation**: **Type**: **Fixed**

With direct user control, the solution to an increment can be accepted after the maximum number of iterations allowed has been completed (as defined in "Commonly used control parameters," Section 7.2.2), even if the equilibrium tolerances are not satisfied. This approach is not recommended; it should be used only in special cases when you have a thorough understanding of how to interpret results obtained in this way. Very small increments and a minimum of two iterations are usually necessary if this option is used.

Input File Usage: *STATIC, DIRECT=NO STOP

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General**: **Other**: **Accept solution after reaching maximum number of iterations**

Steady-state frictional sliding

In a static analysis procedure you can model steady-state frictional sliding between two deformable bodies or between a deformable and a rigid body that are moving with different velocities by specifying the motions of the bodies as predefined fields. In this case it is assumed that the slip velocity follows from the difference in the user-specified velocities and is independent of the nodal displacements, as described in "Coulomb friction," Section 5.2.3 of the Abaqus Theory Guide.

Since this frictional behavior is different from the frictional behavior used without steady-state frictional sliding, discontinuities may arise in the solutions between an analysis step in which relative velocity is determined from predefined motions and prior steps. An example is the discontinuity that occurs between the initial preloading of the disc pads in a disc brake system and the subsequent braking analysis where the disc spins with a prescribed rotation. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to the analysis step in which predefined motion is specified use a zero coefficient of friction. You can then modify the friction properties in the steady-state analysis to use the desired friction coefficient (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).

Input File Usage: *MOTION

Abaqus/CAE Usage: Predefined motion fields are not supported in Abaqus/CAE.

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified. "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions.

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6); to warping degree of freedom 7 in open-section beam elements; or, if hydrostatic fluid elements are included in the model, to fluid pressure degree of freedom 8. If boundary conditions are applied to rotation degrees of freedom, you must understand how finite rotations are handled by Abaqus (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). During the analysis prescribed boundary conditions can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2).

Loads

The following loads can be prescribed in a static stress analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

The following predefined fields can be specified in a static stress analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in a static stress analysis, nodal temperatures can be specified as a predefined field. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.

Material options

Most material models that describe mechanical behavior are available for use in a static stress analysis. The following material properties are not active during a static stress analysis: acoustic properties,

thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, and pore fluid flow properties.

Rate-dependent yield ("Rate-dependent yield," Section 23.2.3), hysteresis ("Hysteresis in elastomers," Section 22.8.1), and two-layer viscoplasticity ("Two-layer viscoplasticity," Section 23.2.11) are the only time-dependent material responses that are active during a static analysis. The rate-dependent yield response is often important in rapid processes such as metal-working problems. The hysteresis model is useful in modeling the large-strain, rate-dependent response of elastomers that exhibit a pronounced hysteresis under cyclic loading. The two-layer viscoplasticity model is useful in situations where a significant time-dependent behavior as well as plasticity is observed, which for metals typically occurs at elevated temperatures. An appropriate time scale must be specified so that Abaqus/Standard can treat the rate dependence of the material responses correctly.

Static creep and swelling problems and time-domain viscoelastic models are analyzed by the quasi-static procedure ("Quasi-static analysis," Section 6.2.5). When any of these time-dependent material models are used in a static analysis, a rate-independent elastic solution is obtained and the chosen time scale does not have an effect on the material response. For creep and swelling behavior this implies that the loading is applied instantaneously compared with the natural time scale over which creep effects take place.

The same concept of instantaneous load application applies to time-domain viscoelastic behavior. You can also obtain the fully relaxed long-term viscoelastic solution directly in a static procedure without having to perform a transient analysis; this choice is meaningful only when time-domain viscoelastic material properties are defined. If the long-term viscoelastic solution is requested, the internal stresses associated with each of the Prony series terms are increased gradually from their values at the beginning of the step to their long-term values at the end of the step.

For the two-layer viscoplastic material model, you can obtain the long-term response of the elastic-plastic network alone.

When frequency-domain viscoelastic material properties are defined (see "Frequency domain viscoelasticity," Section 22.7.2), the corresponding elastic moduli must be specified as long-term elastic moduli. This implies that the response corresponds to the long-term elastic solution, regardless of the time period specified for the step.

Input File Usage: Use the following option to obtain the fully relaxed long-term elastic solution with time-domain viscoelasticity or the long-term elastic-plastic solution for two-layer viscoplasticity:

*STATIC, LONG TERM

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, General** or **Static, Riks**: **Other**: **Obtain long-term solution with time-domain material properties**

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a static stress analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). Although velocities are not available in a static stress analysis, dashpots can still be used (they can be useful in stabilizing an unstable problem). The relative velocity will be calculated as described in "Dashpots," Section 32.2.1.

Acoustic elements are not active in a static step. Consequently, if an acoustic-solid analysis includes a static step, only the solid elements will deform. If the deformations are large, the acoustic and solid meshes may not conform, and subsequent acoustic-structural analysis steps may produce misleading results. See "ALE adaptive meshing: overview," Section 12.2.1, for information on using the adaptive meshing technique to deform the acoustic mesh.

Output

The element output available for a static stress analysis includes stress; strain; energies; the values of state, field, and user-defined variables; and composite failure measures. The nodal output available includes displacements, reaction forces, and coordinates. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,NLGEOM)
Once NLGEOM is specified, it will be active in all subsequent steps
*STATIC, DIRECT
Data line to define direct time incrementation
*BOUNDARY
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*END STEP
**
*STEP
*STATIC
Data line to control automatic time incrementation
*BOUNDARY, OP=MOD
Data lines to modify or add zero-valued or nonzero boundary conditions
```

***CLOAD, OP=NEW**
Data lines to specify new concentrated loads; all previous concentrated loads will be removed
***DLOAD, OP=MOD**
Data lines to specify additional or modified distributed loads
***TEMPERATURE and/or *FIELD**
Data lines to specify additional or modified values of predefined fields
***END STEP**

6.2.3 EIGENVALUE BUCKLING PREDICTION

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Static stress analysis procedures: overview," Section 6.2.1
- *BUCKLE
- "Configuring a buckling procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Creating and modifying prescribed conditions," Section 16.4 of the Abaqus/CAE User's Guide

Overview

Eigenvalue buckling analysis:

- is generally used to estimate the critical (bifurcation) load of "stiff" structures;
- is a linear perturbation procedure;
- can be the first step in an analysis of an unloaded structure, or it can be performed after the structure has been preloaded—if the structure has been preloaded, the buckling load from the preloaded state is calculated;
- can be used in the investigation of the imperfection sensitivity of a structure;
- works only with symmetric matrices (hence, unsymmetric stiffness contributions such as the load stiffness associated with follower loads are symmetrized); and
- cannot be used in a model containing substructures.

General eigenvalue buckling

In an eigenvalue buckling problem we look for the loads for which the model stiffness matrix becomes singular, so that the problem

$$K^{MN} v^M = 0$$

has nontrivial solutions. K^{MN} is the tangent stiffness matrix when the loads are applied, and the v^M are nontrivial displacement solutions. The applied loads can consist of pressures, concentrated forces, nonzero prescribed displacements, and/or thermal loading.

Eigenvalue buckling is generally used to estimate the critical buckling loads of stiff structures (classical eigenvalue buckling). Stiff structures carry their design loads primarily by axial or membrane action, rather than by bending action. Their response usually involves very little deformation prior to buckling. A simple example of a stiff structure is the Euler column, which responds very stiffly to a

compressive axial load until a critical load is reached, when it bends suddenly and exhibits a much lower stiffness. However, even when the response of a structure is nonlinear before collapse, a general eigenvalue buckling analysis can provide useful estimates of collapse mode shapes.

The base state

The buckling loads are calculated relative to the base state of the structure. If the eigenvalue buckling procedure is the first step in an analysis, the initial conditions form the base state; otherwise, the base state is the current state of the model at the end of the last general analysis step (see "General and linear perturbation procedures," Section 6.1.3). Thus, the base state can include preloads ("dead" loads), P^N. The preloads are often zero in classical eigenvalue buckling problems.

If geometric nonlinearity was included in the general analysis steps prior to the eigenvalue buckling analysis (see "General and linear perturbation procedures," Section 6.1.3), the base state geometry is the deformed geometry at the end of the last general analysis step. If geometric nonlinearity was omitted, the base state geometry is the original configuration of the body.

The eigenvalue problem

An incremental loading pattern, Q^N, is defined in the eigenvalue buckling prediction step. The magnitude of this loading is not important; it will be scaled by the load multipliers, λ_i, found in the eigenvalue problem:

$$\left(K_0^{NM} + \lambda_i K_\Delta^{NM}\right) v_i^M = 0,$$

where

K_0^{NM}	is the stiffness matrix corresponding to the base state, which includes the effects of the preloads, P^N (if any);
K_Δ^{NM}	is the differential initial stress and load stiffness matrix due to the incremental loading pattern, Q^N;
λ_i	are the eigenvalues;
v_i^M	are the buckling mode shapes (eigenvectors);
M and N	refer to degrees of freedom M and N of the whole model; and
i	refers to the ith buckling mode.

The critical buckling loads are then $P^N + \lambda_i Q^N$. Normally, the lowest value of λ_i is of interest. The preload pattern, P^N, and perturbation load pattern, Q^N, may be different. For example, P^N might be thermal loading caused by temperature changes, while Q^N is caused by application of pressure.

The buckling mode shapes, v_i^M, are normalized vectors and do not represent actual magnitudes of deformation at critical load. They are normalized so that the maximum displacement component is 1.0. If all displacement components are zero, the maximum rotation component is normalized to 1.0. These buckling mode shapes are often the most useful outcome of the eigenvalue analysis, since they predict the likely failure mode of the structure.

Abaqus/Standard can extract eigenvalues and eigenvectors for symmetric matrices only; therefore, K_0^{NM} and K_Δ^{NM} are symmetrized. If the matrices have significant unsymmetric parts, the eigenproblem may not be exactly what you expected to solve.

Selecting the eigenvalue extraction method

Abaqus/Standard offers the Lanczos and the subspace iteration eigenvalue extraction methods. The Lanczos method is generally faster when a large number of eigenmodes is required for a system with many degrees of freedom. The subspace iteration method may be faster when only a few (less than 20) eigenmodes are needed.

By default, the subspace iteration eigensolver is employed. Subspace iteration and the Lanczos solver can be used for different steps in the same analysis; there is no requirement that the same eigensolver be used for all appropriate steps.

For both eigensolvers you specify the desired number of eigenvalues; Abaqus/Standard will choose a suitable number of vectors for the subspace iteration procedure or a suitable block size for the Lanczos method (although you can override this choice, if needed). Significant overestimation of the actual number of eigenvalues can create very large files. If the actual number of eigenvalues is underestimated, Abaqus/Standard will issue a corresponding warning message.

In general, the block size for the Lanczos method should be as large as the largest expected multiplicity of eigenvalues (that is, the largest number of modes with the same eigenvalue). A block size larger than 10 is not recommended. If the number of eigenvalues requested is n, the default block size is the minimum of (7, n). The number of block Lanczos steps is usually determined by Abaqus/Standard, but you can change it when you define the eigenvalue buckling prediction step. In general, if a particular type of eigenproblem converges slowly, providing more block Lanczos steps will reduce the analysis cost. On the other hand, if you know that a particular type of problem converges quickly, providing fewer block Lanczos steps will reduce the amount of in-core memory used. If the number of eigenvalues requested is n, the default is

Block size	$n \leq 10$	$n > 10$
1	40	70
2	40	60
3	30	60
≥ 4	30	30

If the subspace iteration technique is requested, you can also specify the maximum eigenvalue of interest; Abaqus/Standard will extract eigenvalues until either the requested number of eigenvalues has been extracted or the last eigenvalue extracted exceeds the maximum eigenvalue of interest.

If the Lanczos eigensolver is requested, you can also specify the minimum and/or maximum eigenvalues of interest; Abaqus/Standard will extract eigenvalues until either the requested number of eigenvalues has been extracted in the given range or all the eigenvalues in the given range have been extracted.

Input File Usage: Use the following option to perform an eigenvalue buckling analysis using the subspace iteration method:

*BUCKLE, EIGENSOLVER=SUBSPACE (default)

Use the following option to perform an eigenvalue buckling analysis using the Lanczos method:

*BUCKLE, EIGENSOLVER=LANCZOS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Buckle**: **Eigensolver: Lanczos** or **Subspace**

Limitations associated with applying the Lanczos eigensolver to a buckling analysis

The Lanczos eigensolver cannot be used for buckling analyses in which the stiffness matrix is indefinite, as in the following cases:

- A model containing hybrid elements or connector elements.
- A model containing distributing coupling constraints, defined either directly ("Coupling constraints," Section 35.3.2; "Shell-to-solid coupling," Section 35.3.3; or "Mesh-independent fasteners," Section 35.3.4) or by the distributing coupling elements (DCOUP2D and DCOUP3D).
- A model containing contact pairs or contact elements.
- A model that has been preloaded above the bifurcation (buckling) load.
- A model that has rigid body modes.

In such cases Abaqus/Standard will issue an error message and terminate the analysis.

Order of calculation and formation of the stiffness matrices

In an eigenvalue buckling prediction step Abaqus/Standard first does a static perturbation analysis to determine the incremental stresses, $\Delta\boldsymbol{\sigma}$, due to Q^N. If the base state did not include geometric nonlinearity, the stiffness matrix used in this static perturbation analysis is the tangent elastic stiffness. If the base state did include geometric nonlinearity, initial stress and load stiffness terms (due to the preload, P^N) are included. The stiffness matrix K_Δ^{NM} corresponding to $\Delta\boldsymbol{\sigma}$ and Q^N is then formed.

In the eigenvalue extraction portion of the buckling step, the stiffness matrix K_0^{NM} corresponding to the base state geometry is formed. Initial stress and the load stiffness terms due to the preload, P^N, are always included regardless of whether or not geometric nonlinearity is included and are calculated based on the geometry of the base state.

When forming the stiffness matrices K_0^{NM} and K_Δ^{NM}, all contact conditions are fixed in the base state.

Buckling modes with closely spaced eigenvalues

Some structures have many buckling modes with closely spaced eigenvalues, which can cause numerical problems. In these cases it often helps to apply enough preload, P^N, to load the structure to just below the buckling load before performing the eigenvalue extraction.

If $P^N = \mu Q^N$—where μ is a scalar constant and the structure is "stiff" and elastic—and if the problem is linear, the structural stiffness changes to $K_0^{NM} + \mu K_\Delta^{NM}$ and the buckling loads are given by $(\mu + \lambda_i)Q^N$. The process is equivalent to a dynamic eigenfrequency extraction with shift μ. The structure should not be preloaded above the buckling load. In that case the subspace iteration process may fail to converge or produce incorrect results; the Lanczos eigensolver cannot be used (as discussed earlier).

In many cases a series of closely spaced eigenvalues indicates that the structure is imperfection sensitive. An eigenvalue buckling analysis will not give accurate predictions of the buckling load for imperfection-sensitive structures; the static Riks procedure should be used instead (see "Unstable collapse and postbuckling analysis," Section 6.2.4).

Understanding negative eigenvalues

Sometimes, negative eigenvalues are reported in an eigenvalue buckling analysis. In most cases such negative eigenvalues indicate that the structure would buckle if the load were applied in the opposite direction. A classical example is a plate under shear loading; the plate will buckle at the same value for positive and negative applied shear load. Buckling under reverse loading can also occur in situations where it may not be expected. For example, a pressure vessel under external pressure may exhibit a negative eigenvalue (buckling under internal pressure) due to local buckling of a stiffener. Such "physical" negative buckling modes are usually readily understood once they are displayed and can usually be avoided by applying a preload before the buckling analysis.

Negative eigenvalues sometimes correspond to buckling modes that cannot be understood readily in terms of physical behavior, particularly if a preload is applied that causes significant geometric nonlinearity. In this case a geometrically nonlinear load-displacement analysis should be performed ("Unstable collapse and postbuckling analysis," Section 6.2.4).

Including large geometry changes in a buckling analysis

Because buckling analysis is usually done for "stiff" structures, it is not usually necessary to include the effects of geometry change in establishing equilibrium for the base state. However, if significant geometry change is involved in the base state and this effect is considered to be important, it can be included by specifying that geometric nonlinearity should be considered for the base state step (see "General and linear perturbation procedures," Section 6.1.3). In such cases it is probably more realistic to perform a geometrically nonlinear load-displacement analysis (Riks analysis) to determine the collapse loads, especially for imperfection-sensitive structures.

While large deformation can be included in the preload, the eigenvalue buckling theory relies on there being little geometric change due to the "live" buckling load, $\lambda_i Q^N$. If the live load produces significant geometric change, a nonlinear collapse (Riks) analysis must be used. The total buckling load predicted by the eigenvalue analysis, $P^N + \lambda_i Q^N$, may be a good estimate for the limit load in the nonlinear buckling analysis. The Riks method is described in "Unstable collapse and postbuckling analysis," Section 6.2.4.

Initial conditions

The initial values of quantities such as stress, temperature, field variables, and solution-dependent variables can be specified for an eigenvalue buckling analysis. If the buckling step is the first step in the analysis, these initial conditions form the base state of the structure. "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions.

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) or to warping degree of freedom 7 in open-section beam elements ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). A nonzero prescribed boundary condition in a general analysis step preceding the eigenvalue buckling analysis can be used to preload the structure. Nonzero boundary conditions prescribed in an eigenvalue buckling step will contribute to the incremental stress $\Delta\sigma$ and, thus, will contribute to the differential initial stress stiffness. When prescribing nonzero boundary conditions, you must interpret the resulting eigenproblem carefully. Nonzero prescribed boundary conditions will be treated as constraints (i.e., as if they were fixed) during the eigenvalue extraction. Therefore, unless the prescribed boundary conditions are removed for the eigenvalue extraction by specifying buckling mode boundary conditions (see the discussion below), the mode shapes may be altered by these boundary conditions.

Amplitude definitions ("Amplitude curves," Section 34.1.2) cannot be used to vary the magnitudes of prescribed boundary conditions during an eigenvalue buckling analysis.

You can define perturbation load and buckling mode boundary conditions in an eigenvalue buckling prediction step.

Input File Usage: Use either of the following two options to define perturbation load boundary conditions:

```
*BOUNDARY
*BOUNDARY, LOAD CASE=1
```

Use the following option to define buckling mode boundary conditions:

```
*BOUNDARY, LOAD CASE=2, OP=NEW
```

The OP=NEW parameter is required when you define buckling mode boundary conditions in an eigenvalue buckling prediction step; however, the perturbation load boundary conditions in the step can use either OP=NEW or OP=MOD.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Mechanical** for the **Category** and **Symmetry/Antisymmetry/Encastre** for the **Types for Selected Step**: select region: toggle on **Stress perturbation only** to define a perturbation load boundary condition; toggle on **Buckling mode calculation only** to define a buckling mode boundary condition; toggle on **Stress perturbation and buckling mode calculation** to define both types of boundary conditions

Combining boundary conditions

The buckling mode shapes depend on the stresses in the base state as well as the incremental stresses due to the perturbation loading in the buckling step. These stresses are influenced by the boundary conditions used in each step. In a general eigenvalue buckling analysis the following types of boundary conditions can influence the stresses:

1. The boundary conditions in the base state.
2. The boundary conditions used to calculate the linear perturbation stresses, $\Delta\sigma$. These boundary conditions will be:
   a. the perturbation load boundary conditions specified in the eigenvalue buckling step; or
   b. the base-state boundary conditions if no perturbation load boundary conditions are specified in the eigenvalue buckling step; or
   c. the buckling mode boundary conditions if neither perturbation load boundary conditions nor base-state boundary conditions exist.
3. The boundary conditions used for the eigenvalue extraction. These boundary conditions will be:
   a. the buckling mode boundary conditions; or
   b. the perturbation load boundary conditions if buckling mode boundary conditions are not specified in the eigenvalue buckling step; or
   c. the base-state boundary conditions if no boundary condition definition is used in the eigenvalue buckling step.

Table 6.2.3–1 summarizes the use of boundary conditions during an eigenvalue buckling step. When buckling mode boundary conditions are specified, *all* boundary conditions to be imposed during eigenvalue extraction must be specified.

Buckling of symmetric structures

The buckling mode shapes of symmetric structures subjected to symmetric loadings are either symmetric or antisymmetric. In such cases it is often more efficient to model only part of the structure and then perform the buckling analysis twice for each symmetry plane: once with symmetric boundary conditions and once with antisymmetric boundary conditions.

The live load pattern is usually symmetric, so symmetric boundary conditions are required for the calculation of the perturbation stresses used in the formation of the initial stress stiffness matrix. The boundary conditions must be switched to antisymmetric for the eigenvalue extraction to obtain the antisymmetric modes. “Buckling of a cylindrical shell under uniform axial pressure,” Section 1.2.3 of the Abaqus Benchmarks Guide, illustrates such a case.

If the model includes more than one symmetry plane, it may be necessary to study all permutations of symmetric and antisymmetric boundary conditions for each symmetry plane.

Table 6.2.3–1 Boundary conditions in effect during the different portions of an eigenvalue buckling analysis.

User-defined boundary conditions		Boundary conditions used by Abaqus	
Base state	**Eigenvalue buckling prediction step**	**Linear perturbation**	**Eigenvalue extraction**
B	0	B	B
0	1	1	1
0	2	2	2
B	1	1	1
B	2	B	2
0	1, 2	1	2
B	1, 2	1	2
B = base-state boundary conditions; 0 = no boundary conditions specified 1 = perturbation load boundary conditions 2 = buckling mode boundary conditions			

Asymmetric buckling of axisymmetric structures

Axisymmetric structures subjected to compressive loading often collapse in nonaxisymmetric modes. These modes cannot be found with purely axisymmetric modeling such as that provided by shell elements SAX1 and SAX2 (“Axisymmetric shell element library,” Section 29.6.9) or continuum elements CAX4 or CAX8 (“Axisymmetric solid element library,” Section 28.1.6). Such analyses must be done with three-dimensional shell or continuum elements.

Loads

The following types of loading can be prescribed in an eigenvalue buckling analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see “Concentrated loads,” Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see “Distributed loads,” Section 34.4.3. The distributed load types available with particular elements are described in Part VI, “Elements.”

The load stiffness can have a significant effect on the critical buckling load; therefore, Abaqus/Standard will take the load stiffness due to preloads into account when solving the eigenvalue buckling problem. It is important that the structure not be preloaded above the critical buckling load.

Any load applied during the eigenvalue buckling analysis is called a “live” load. This incremental load, Q^N, describes the load pattern for which buckling sensitivity is being investigated; its magnitude

is not important. This incremental loading definition represents linear perturbation loads, as described in "Applying loads: overview," Section 34.4.1.

Follower forces (such as concentrated loads assumed to rotate with the nodal rotation or pressure loads) lead to an unsymmetric load stiffness. Since eigenvalue extraction in Abaqus/Standard can be performed only on symmetric matrices, eigenvalue analysis with follower loads may not yield correct results.

Amplitude definitions cannot be used during an eigenvalue buckling analysis. "Applying loads: overview," Section 34.4.1, describes all of the available loads.

Prescribed boundary conditions can also be used to load the structure in an eigenvalue buckling analysis, as discussed earlier.

Predefined fields

In an eigenvalue buckling prediction step, nodal temperatures can be specified (see "Predefined fields," Section 34.6.1). The specified temperatures will cause thermal strain during the static perturbation analysis if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2), and incremental stresses $\Delta\sigma$ will be generated. Hence, Abaqus/Standard can analyze buckling due to thermal stress. The specified temperature will not affect temperature-dependent material properties during the eigenvalue buckling prediction step; the material properties are based on the temperature in the base state. Amplitude definitions cannot be used to vary the magnitudes of prescribed temperatures during an eigenvalue buckling analysis.

Material options

During an eigenvalue buckling analysis, the model's response is defined by its linear elastic stiffness in the base state. All nonlinear and/or inelastic material properties, as well as effects involving time or strain rate, are ignored during an eigenvalue buckling analysis. In classical eigenvalue buckling the response in the base state is also linear.

If temperature-dependent elastic properties are used, the eigenvalue buckling analysis will not account for changes in the stiffness matrix due to temperature changes. The material properties of the base state will be used.

Acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties are not active during an eigenvalue buckling analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in an eigenvalue buckling analysis, with the exception that hybrid and contact elements cannot be used with the Lanczos eigensolver (as discussed earlier). See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

The values of the eigenvalues, λ_i, will be listed in the printed output file. If output of stresses, strains, reaction forces, etc. is requested, this information will be printed for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Buckling mode shapes can be plotted in the Visualization module of Abaqus/CAE.

Input file template

The following template describes a very general eigenvalue buckling problem, where as many eigenvalue buckling prediction steps as needed can be specified.

Symmetric boundary conditions are specified in the model definition part of the Abaqus/Standard input and, therefore, belong to the base state (see "General and linear perturbation procedures," Section 6.1.3). In the first buckling step Abaqus/Standard uses the base-state boundary conditions to solve for the perturbation stresses as well as for the eigenvalue extraction.

In the second buckling step the boundary conditions for the base state, the initial stress calculation, and the eigenvalue extraction are all different. Abaqus/Standard uses the specified symmetry boundary conditions to solve for the perturbation stresses but uses the specified antisymmetry boundary conditions for the eigenvalue extraction.

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions contributing to the base state
**
*STEP, NLGEOM
The load stiffness terms will be included in the eigenvalue buckling steps
since the NLGEOM parameter is used in this (optional) preload step
*STATIC
Data line to control incrementation
*BOUNDARY
Data lines to specify nonzero boundary conditions (dead loads)
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify dead loads, P^N
*END STEP
**
*STEP
*BUCKLE
Data line to request the desired number of symmetric modes
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify perturbation loading, Q^N
*END STEP
```

**
**STEP
**BUCKLE
Data line to request the desired number of antisymmetric modes
**CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify perturbation loading, Q^N
**BOUNDARY, LOAD CASE=1
Data lines to specify all boundary conditions for perturbation loading
**BOUNDARY, LOAD CASE=2, OP=NEW
Data lines to specify all antisymmetric boundary conditions for eigenvalue extraction
**END STEP

6.2.4 UNSTABLE COLLAPSE AND POSTBUCKLING ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Static stress analysis procedures: overview," Section 6.2.1
- "Introducing a geometric imperfection into a model," Section 11.3.1
- *STATIC
- *IMPERFECTION
- "Configuring a static, Riks procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The Riks method:

- is generally used to predict unstable, geometrically nonlinear collapse of a structure;
- can include nonlinear materials and boundary conditions;
- often follows an eigenvalue buckling analysis to provide complete information about a structure's collapse; and
- can be used to speed convergence of ill-conditioned or snap-through problems that do not exhibit instability.

Unstable response

Geometrically nonlinear static problems sometimes involve buckling or collapse behavior, where the load-displacement response shows a negative stiffness and the structure must release strain energy to remain in equilibrium. Several approaches are possible for modeling such behavior. One is to treat the buckling response dynamically, thus actually modeling the response with inertia effects included as the structure snaps. This approach is easily accomplished by restarting the terminated static procedure ("Restarting an analysis," Section 9.1.1) and switching to a dynamic procedure ("Implicit dynamic analysis using direct integration," Section 6.3.2) when the static solution becomes unstable. In some simple cases displacement control can provide a solution, even when the conjugate load (the reaction force) is decreasing as the displacement increases. Another approach would be to use dashpots to stabilize the structure during a static analysis. Abaqus/Standard offers an automated version of this stabilization approach for the static analysis procedures (see "Static stress analysis," Section 6.2.2; "Quasi-static analysis," Section 6.2.5; "Fully coupled thermal-stress analysis," Section 6.5.3; or "Coupled pore fluid diffusion and stress analysis," Section 6.8.1).

Alternatively, static equilibrium states during the unstable phase of the response can be found by using the "modified Riks method." This method is used for cases where the loading is proportional;

that is, where the load magnitudes are governed by a single scalar parameter. The method can provide solutions even in cases of complex, unstable response such as that shown in Figure 6.2.4–1.

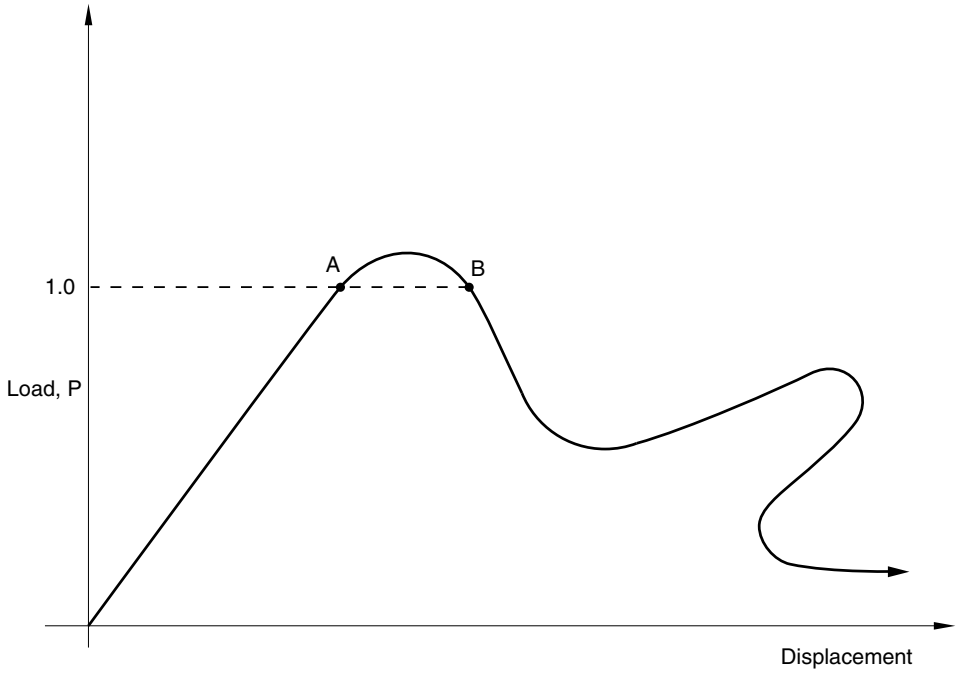


Figure 6.2.4–1 Proportional loading with unstable response.

The Riks method is also useful for solving ill-conditioned problems such as limit load problems or almost unstable problems that exhibit softening.

The Riks method

In simple cases linear eigenvalue analysis ("Eigenvalue buckling prediction," Section 6.2.3) may be sufficient for design evaluation; but if there is concern about material nonlinearity, geometric nonlinearity prior to buckling, or unstable postbuckling response, a load-deflection (Riks) analysis must be performed to investigate the problem further.

The Riks method uses the load magnitude as an additional unknown; it solves simultaneously for loads and displacements. Therefore, another quantity must be used to measure the progress of the solution; Abaqus/Standard uses the "arc length," l, along the static equilibrium path in load-displacement space. This approach provides solutions regardless of whether the response is stable or unstable. See the "Modified Riks algorithm," Section 2.3.2 of the Abaqus Theory Guide, for a detailed description of the method.

Proportional loading

If the Riks step is a continuation of a previous history, any loads that exist at the beginning of the step and are not redefined are treated as "dead" loads with constant magnitude. A load whose magnitude is defined in the Riks step is referred to as a "reference" load. All prescribed loads are ramped from the initial (dead load) value to the reference values specified.

The loading during a Riks step is always proportional. The current load magnitude, P_{total}, is defined by

$$P_{\text{total}} = P_0 + \lambda(P_{\text{ref}} - P_0),$$

where P_0 is the "dead load," P_{ref} is the reference load vector, and λ is the "load proportionality factor." The load proportionality factor is found as part of the solution. Abaqus/Standard prints out the current value of the load proportionality factor at each increment.

Incrementation

Abaqus/Standard uses Newton's method (as described in "Static stress analysis," Section 6.2.2) to solve the nonlinear equilibrium equations. The Riks procedure uses only a 1% extrapolation of the strain increment.

You provide an initial increment in arc length along the static equilibrium path, Δl_{in}, when you define the step. The initial load proportionality factor, $\Delta\lambda_{in}$, is computed as

$$\Delta\lambda_{in} = \frac{\Delta l_{in}}{l_{period}},$$

where l_{period} is a user-specified total arc length scale factor (typically set equal to 1). This value of $\Delta\lambda_{in}$ is used during the first iteration of a Riks step. For subsequent iterations and increments the value of λ is computed automatically, so you have no control over the load magnitude. The value of λ is part of the solution. Minimum and maximum arc length increments, Δl_{min} and Δl_{max}, can be used to control the automatic incrementation.

Input File Usage: *STATIC, RIKS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**

Direct user control of the increment size is also provided; in this case the incremental arc length, Δl, is kept constant. This method is not recommended for a Riks analysis since it prevents Abaqus/Standard from reducing the arc length when a severe nonlinearity is encountered.

Input File Usage: *STATIC, RIKS, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Static, Riks**:
Incrementation: **Type: Fixed**

Ending a Riks analysis step

Since the loading magnitude is part of the solution, you need a method to specify when the step is completed. You can specify a maximum value of the load proportionality factor, λ_{end}, or a maximum displacement value at a specified degree of freedom. The step will terminate when either value is crossed. If neither of these finishing conditions is specified, the analysis will continue for the number of increments specified in the step definition (see "Defining an analysis," Section 6.1.2).

Bifurcation

The Riks method works well in snap-through problems—those in which the equilibrium path in load-displacement space is smooth and does not branch. Generally you do not need take any special precautions in problems that do not exhibit branching (bifurcation). "Snap-through buckling analysis of circular arches," Section 1.2.1 of the Abaqus Example Problems Guide, is an example of a smooth snap-through problem.

The Riks method can also be used to solve postbuckling problems, both with stable and unstable postbuckling behavior. However, the exact postbuckling problem cannot be analyzed directly due to the discontinuous response at the point of buckling. To analyze a postbuckling problem, it must be turned into a problem with continuous response instead of bifurcation. This effect can be accomplished by introducing an initial imperfection into a "perfect" geometry so that there is some response in the buckling mode before the critical load is reached.

Introducing geometric imperfections

Imperfections are usually introduced by perturbations in the geometry. Unless the precise shape of an imperfection is known, an imperfection consisting of multiple superimposed buckling modes must be introduced ("Eigenvalue buckling prediction," Section 6.2.3). Abaqus allows you to define imperfections; see "Introducing a geometric imperfection into a model," Section 11.3.1.

In this way the Riks method can be used to perform postbuckling analyses of structures that show linear behavior prior to (bifurcation) buckling. An example of this method of introducing geometric imperfections is presented in "Buckling of a cylindrical shell under uniform axial pressure," Section 1.2.3 of the Abaqus Benchmarks Guide.

By performing a load-displacement analysis, other important nonlinear effects, such as material inelasticity or contact, can be included. In contrast, all inelastic effects are ignored in a linear eigenvalue buckling analysis and all contact conditions are fixed in the base state. Imperfections based on linear buckling modes can also be useful for the analysis of structures that behave inelastically prior to reaching peak load.

Introducing loading imperfections

Perturbations in loads or boundary conditions can also be used to introduce initial imperfections. In this case fictitious "trigger" loads can be used to initiate the instability. The trigger loads should perturb the structure in the expected buckling modes. Typically, these loads are applied as dead loads prior to the Riks step so that they have fixed magnitudes. The magnitudes of trigger loads must be sufficiently

small so that they do not affect the overall postbuckling solution. It is your responsibility to choose appropriate magnitudes and locations for such fictitious loads; Abaqus/Standard does not check that they are reasonable.

Obtaining a solution at a particular load or displacement value

The Riks algorithm cannot obtain a solution at a given load or displacement value since these are treated as unknowns—termination occurs at the first solution that satisfies the step termination criterion. To obtain solutions at exact values of load or displacement, the solution must be restarted at the desired point in the step ("Restarting an analysis," Section 9.1.1) and a new, non-Riks step must be defined. Since the subsequent step is a continuation of the Riks analysis, the load magnitude in that step must be given appropriately so that the step begins with the loading continuing to increase or decrease according to its behavior at the point of restart. For example, if the load was increasing at the restart point and was positive, a larger load magnitude than the current magnitude should be given in the restart step to continue this behavior. If the load was decreasing but positive, a smaller magnitude than the current magnitude should be specified.

Restrictions

A Riks analysis is subject to the following restrictions:

- A Riks step cannot be followed by another step in the same analysis. Subsequent steps must be analyzed by using the restart capability.
- If a Riks analysis includes irreversible deformation such as plasticity and a restart using another Riks step is attempted while the magnitude of the load on the structure is decreasing, Abaqus/Standard will find the elastic unloading solution. Therefore, restart should occur at a point in the analysis where the load magnitude is increasing if plasticity is present.
- For postbuckling problems involving loss of contact, the Riks method will usually not work; inertia or viscous damping forces (such as those provided by dashpots) must be introduced in a dynamic or static analysis to stabilize the solution.

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified; "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions.

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) or to warping degree of freedom 7 in open-section beam elements ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Amplitude definitions ("Amplitude curves," Section 34.1.2) cannot be used to vary the magnitudes of prescribed boundary conditions during a Riks analysis.

Loads

The following loads can be prescribed in a Riks analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Since Abaqus/Standard scales loading magnitudes proportionally based on the user-specified magnitudes, amplitude references are ignored when the Riks method is chosen.

If follower loads are prescribed, their contribution to the stiffness matrix may be unsymmetric; the unsymmetric matrix storage and solution scheme can be used to improve computational efficiency in such cases (see "Defining an analysis," Section 6.1.2).

Predefined fields

Nodal temperatures can be specified (see "Predefined fields," Section 34.6.1). Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The loads generated by the thermal strain contribute to the "reference" load specified for the Riks analysis and are ramped up with the load proportionality factor. Hence, the Riks procedure can analyze postbuckling and collapse due to thermal straining.

The values of other user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. Since the concept of time is replaced by arc length in a Riks analysis, the use of properties that change due to changes in temperatures and/or field variables is not recommended.

Material options

Most material models that describe mechanical behavior are available for use in a Riks analysis. The following material properties are not active during a Riks analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties. Materials with history dependence can be used; however, it should be realized that the results will depend on the loading history, which is not known in advance.

The concept of time is replaced by arc length in a Riks analysis. Therefore, any effects involving time or strain rate (such as viscous damping or rate-dependent plasticity) are no longer treated correctly and should not be used.

See Part V, "Materials," for details on the material models available in Abaqus/Standard.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in a Riks analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). Dashpots should not be used since velocities will be calculated as displacement increments divided by arc length, which is meaningless.

Output

Output options are provided to allow the magnitudes of individual load components (pressure, point loads, etc.) to be printed or to be written to the results file. The current value of the load proportionality factor, LPF, will be given automatically with any results or output database file output request. These output options are recommended when the Riks method is used so that load magnitudes can be seen directly. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

```
*HEADING
...
*INITIAL CONDITIONS
Data lines to define initial conditions
*BOUNDARY
Data lines to specify zero-valued boundary conditions
**
*STEP, NLGEOM
*STATIC
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify preload (dead load), P_0
*END STEP
**
*STEP, NLGEOM
*STATIC, RIKS
Data line to define incrementation and stopping criteria
*CLOAD and/or *DLOAD and/or *TEMPERATURE
Data lines to specify reference loading, P_ref
*END STEP
```

6.2.5 QUASI-STATIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Static stress analysis procedures: overview," Section 6.2.1
- *VISCO
- "Configuring a transient, static, stress/displacement analysis with time-dependent material response" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A quasi-static stress analysis in Abaqus/Standard:

- is used to analyze problems with time-dependent material response (creep, swelling, viscoelasticity, and two-layer viscoplasticity);
- is used when inertia effects can be neglected; and
- can be linear or nonlinear.

See "Mass scaling," Section 11.6.1, and "Explicit dynamic analysis," Section 6.3.3, for information on conducting quasi-static analysis in Abaqus/Explicit. See "Implicit dynamic analysis using direct integration," Section 6.3.2, for information on conducting quasi-static analysis using a dynamic procedure in Abaqus/Standard.

Incrementation

You can control the time incrementation in a quasi-static analysis directly, or it can be controlled automatically by Abaqus/Standard. Automatic incrementation is preferred in almost all cases.

Fixed incrementation

If you specify the time increments in a quasi-static analysis directly, fixed time increments equal to the specified initial time increment will be used throughout the analysis.

Input File Usage: *VISCO

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**

Automatic incrementation

If you select automatic incrementation, the size of the time increment is limited by the accuracy of the integration. The user-specified accuracy tolerance parameter limits the maximum inelastic strain rate change allowed over an increment:

$$tolerance \geq (\dot{\bar{\varepsilon}}^{cr}|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}|_t)\Delta t,$$

where t is the time at the beginning of the increment, Δt is the time increment (so that $t + \Delta t$ is the time at the end of the increment), and $\dot{\bar{\varepsilon}}^{cr}$ is the equivalent creep strain rate. To achieve accuracy, the value chosen for the accuracy tolerance parameter should be on the order of σ_{err}/E for creep problems, where σ_{err} is an acceptable level of error in the stress and E is a typical elastic modulus, or on the order of the elastic strains for viscoelasticity problems.

Input File Usage: *VISCO, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**: **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance**: *tolerance*

Selecting explicit creep integration

Nonlinear creep problems ("Rate-dependent plasticity: creep and swelling," Section 23.2.4) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For creep at very low stress levels, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases Abaqus/Standard will invoke the implicit integration scheme automatically.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine **CREEP**; you can restrict Abaqus/Standard to using this method for creep problems (with or without geometric nonlinearity included). See "Rate-dependent plasticity: creep and swelling," Section 23.2.4, for further details.

Input File Usage: *VISCO, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Visco**: **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance**: *tolerance* and **Creep/swelling/viscoelastic integration: Explicit**

Integration scheme for viscoelasticity and rate-dependent yield

Problems including "Time domain viscoelasticity," Section 22.7.1, are always integrated with an unconditionally stable operator. The time step in these problems is limited only by the accuracy tolerance parameter defined above.

Problems including "Rate-dependent yield," Section 23.2.3, and "Parallel rheological framework," Section 22.8.2, are always integrated using an implicit, unconditionally stable method. The accuracy tolerance parameter does not limit the inelastic strain rate change and can be set equal to any nonzero value to activate automatic time incrementation.

Unstable problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers the ability to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified, as described in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6); to warping degree of freedom 7 in open-section beam elements; or, if hydrostatic fluid elements are included in the model, to fluid pressure degree of freedom 8. If boundary conditions are applied to rotation degrees of freedom, you must understand how Abaqus handles finite rotations. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Loads

The following types of loading can be prescribed in a quasi-static analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

The following predefined fields can be specified in a quasi-static analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in quasi-static analysis, nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material options

The quasi-static procedure in Abaqus/Standard is generally used to analyze quasi-static creep and swelling problems, which occur over fairly long time periods ("Rate-dependent plasticity: creep and swelling," Section 23.2.4). This procedure can also be used to analyze viscoelastic materials ("Time domain viscoelasticity," Section 22.7.1, and "Parallel rheological framework," Section 22.8.2) and two-layer viscoplastic materials ("Two-layer viscoplasticity," Section 23.2.11). In addition, all material models that are valid in a static analysis procedure can be used.

Elements

Any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in a quasi-static stress analysis—see "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

In addition to the usual output variables available in Abaqus/Standard (see "Abaqus/Standard output variable identifiers," Section 4.2.1), the following variables are provided specifically for creep problems:

Element integration point variables:

CEEQ	Equivalent creep strain, $\int_0^t \dot{\bar{\varepsilon}}^{cr} dt$.
CESW	Magnitude of the swelling strain.
CEMAG	Magnitude of the creep strain, $\sqrt{\frac{2}{3}\boldsymbol{\varepsilon}^{cr} : \boldsymbol{\varepsilon}^{cr}}$.
CEP	Principal creep strains.
CE	Output of all of the creep strain components and CEEQ, CESW, and CEMAG.

Input file template

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,NLGEOM)
*VISCO, CETOL=tolerance
Data line to define time incrementation and a "real" time scale
*BOUNDARY
```

Data lines to describe nonzero boundary conditions
***CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD**
Data lines to specify loading
***END STEP**

6.2.6 DIRECT CYCLIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- *DIRECT CYCLIC
- *TIME POINTS
- *CONTROLS
- "Configuring a direct cyclic procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A direct cyclic analysis:

- is a quasi-static analysis;
- uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized cyclic response of the structure iteratively;
- avoids the considerable numerical expense associated with a transient analysis;
- is ideally suited for very large problems in which many load cycles must be applied to obtain the stabilized response if transient analysis is performed;
- can be performed with linear or nonlinear material with localized plastic deformation;
- can be used to predict the likelihood of plastic ratchetting;
- assumes geometrically linear behavior and fixed contact conditions;
- uses the elastic stiffness, so the equation system is inverted only once; and
- can also be used to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding growth at the interfaces in laminated composites in a low-cycle fatigue analysis.

Introduction

It is well known that after a number of repetitive loading cycles, the response of an elastic-plastic structure, such as an automobile exhaust manifold subjected to large temperature fluctuations and clamping loads, may lead to a stabilized state in which the stress-strain relationship in each successive cycle is the same as in the previous one. The classical approach to obtain the response of such a structure is to apply the periodic loading repetitively to the structure until a stabilized state is obtained. This approach can be quite expensive, since it may require the application of many loading cycles before the stabilized response is obtained. To avoid the considerable numerical expense associated with a transient analysis, a direct cyclic analysis can be used to calculate the cyclic response of the structure directly.

The basis of this method is to construct a displacement function $\overline{u}(t)$ that describes the response of the structure at all times t during a load cycle with period T as shown in Figure 6.2.6–1.

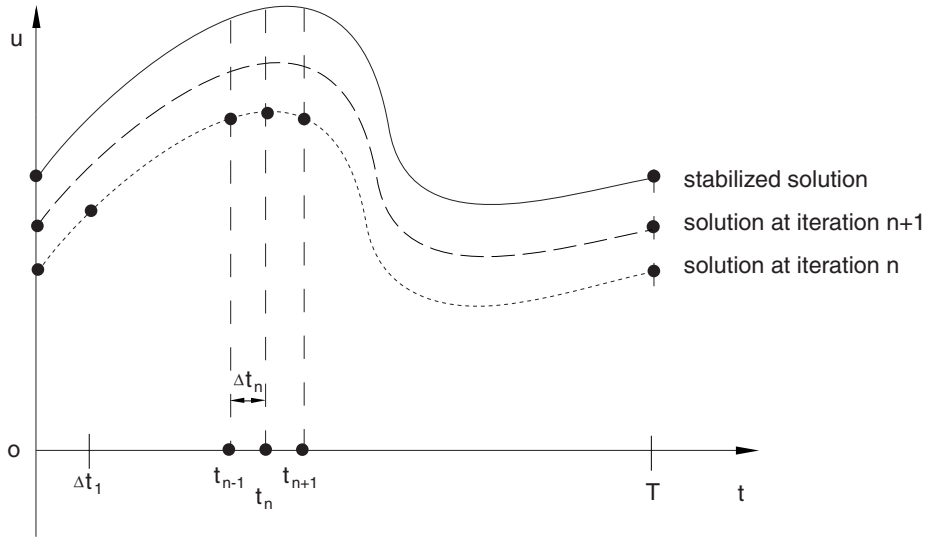


Figure 6.2.6–1 A displacement function at all times t during a load cycle with period T at different iterations.

A truncated Fourier series is used for this purpose,

$$\overline{u}(t) = u_0 + \sum_{k=1}^{n} \left[u_k^s \sin k\omega t + u_k^c \cos k\omega t \right],$$

where n stands for the number of terms in the Fourier series, $\omega = 2\pi/T$ is the angular frequency, and u_0, u_k^s, and u_k^c are unknown displacement coefficients associated with each degree of freedom in the problem. Abaqus/Standard solves for the unknown displacement coefficients by using a modified Newton method, with the elastic stiffness matrix at the beginning of the analysis step serving as the Jacobian in the scheme. We expand the residual vector in the modified Newton method using a Fourier series of the same form as the displacement solution:

$$\overline{R}(t) = R_0 + \sum_{k=1}^{n} \left[R_k^s \sin k\omega t + R_k^c \cos k\omega t \right],$$

where each residual vector coefficient R_0, R_k^s, and R_k^c in the Fourier series corresponds to a displacement coefficient u_0, u_k^s, and u_k^c, respectively. The residual coefficients are obtained by tracking through the entire load cycle. At each instant in time in the cycle Abaqus/Standard obtains the residual vector $R(t)$ by

using standard element-by-element calculations, which—when integrated over the entire cycle—provide the Fourier coefficients

$$R_0 = \frac{1}{T}\int_0^T R(t)\ dt,$$
$$R_k^s = \frac{2}{T}\int_0^T R(t)\sin k\omega t\ dt,$$
$$R_k^c = \frac{2}{T}\int_0^T R(t)\cos k\omega t\ dt.$$

The displacement solution is obtained by solving for corrections to the displacement Fourier coefficients corresponding to each residual coefficient. The updated displacement solution is used in the next iteration to obtain the displacements at each instant in time. This process is repeated until convergence is obtained. Each pass through the complete load cycle can, therefore, be thought of as a single iteration of the solution to the nonlinear problem. Convergence is measured by ensuring that all entries of the residual coefficients are small.

The algorithm to obtain a stabilized cycle is described in detail in "Direct cyclic algorithm," Section 2.2.3 of the Abaqus Theory Guide.

Direct cyclic analysis

A direct cyclic step can be the only step in an analysis, can follow a general or linear perturbation step, or can be followed by a general or linear perturbation step. If a direct cyclic step is followed by a general step, the solution at the end of the direct cyclic step will be the initial state of the general step. If a direct cyclic step follows a general or linear perturbation step, the elastic stiffness matrix at the end of the last general analysis step prior to the direct cyclic step will serve as the Jacobian in the direct cyclic procedure. Any prior (non-cyclic) loads are simply included in the constant part of the Fourier expansion of the residual vectors, and the plastic strains at the end of the preloading step are used as initial conditions for the direct cyclic step.

Multiple direct cyclic analysis steps can be included in a single analysis. In such a case the Fourier series coefficients obtained in the previous step can be used as starting values in the current step. By default, the Fourier coefficients are reset to zero, thus allowing application of cyclic loading conditions that are very different from those defined in the previous direct cyclic step.

You can specify that a direct cyclic step in a restart analysis should use the Fourier coefficients from the previous step, thus allowing continuation of an analysis that has not reached a stabilized cycle. In a direct cyclic analysis a restart file is written at the end of the cycle or time period. Consequently, a restart analysis that is a continuation of a previous direct cyclic analysis will start with a new iteration at $t = 0$ (see "Restarting an analysis," Section 9.1.1).

Input File Usage: Use the following option to reset the Fourier series coefficients to zero:

*DIRECT CYCLIC, CONTINUE=NO (default)

Use the following option to specify that the current step is a continuation of the previous direct cyclic step:

*DIRECT CYCLIC, CONTINUE=YES

Abaqus/CAE Usage: Use the following option to reset the Fourier series coefficients to zero (default):

Step module: **Create Step**: **General**: **Direct cyclic**

Use the following option to specify that the current step is a continuation of the previous direct cyclic step:

Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Use displacement Fourier coefficients from previous direct cyclic step**

Using the direct cyclic approach to perform low-cycle fatigue analysis

The direct cyclic procedure can also be used in conjunction with the damage extrapolation technique to predict progressive damage and failure for ductile bulk materials and/or to predict delamination/debonding at the interfaces in laminated composites in a low-cycle fatigue analysis. In this case multiple cycles can be included in a single direct cyclic analysis, as described in "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7.

Input File Usage: *DIRECT CYCLIC, FATIGUE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**

Controlling the solution accuracy

Direct cyclic analysis combines a Fourier series approximation with time integration of the nonlinear material behavior to obtain the stabilized cyclic solution iteratively using a modified Newton method. The accuracy of the algorithm depends on the number of Fourier terms used, the number of iterations taken to obtain the stabilized solution, and the number of time points within the load period at which the material response and residual vector are evaluated. Abaqus/Standard allows you to control the solution in several ways, as described below.

Controlling the iterations in the modified Newton method

In the direct cyclic method global Newton iterations are performed to determine corrections to the displacement Fourier coefficients. During each global iteration Abaqus/Standard tracks through the entire time cycle to compute the residual vector at a suitable number of time points. This involves standard element-by-element finite element calculations in which history-dependent material variables are integrated. The residual vector is integrated over the period to obtain the Fourier residual coefficients, which in turn yield corrections in displacement coefficients when the system of equations is solved. Abaqus/Standard will continue with the iterative process until convergence is obtained or until the maximum number of iterations allowed has been reached. You can specify the maximum number of iterations when you define the direct cyclic step; the default is 200 iterations.

Input File Usage: *DIRECT CYCLIC
, , , , , , , *max number of iterations*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Maximum number of iterations:** *max number of iterations*

Specifying convergence criteria

Convergence is best measured by ensuring that all the residual coefficients are sufficiently small compared to the time averaged force and that all the corrections to displacement Fourier coefficients are sufficiently small compared to the displacement Fourier coefficients. The time averaged force is defined in "Convergence criteria for nonlinear problems," Section 7.2.3. Abaqus/Standard requires that the ratio of the maximum residual coefficient to the time averaged force, CR_n^α, and the ratio of the maximum correction to the displacement coefficients to the largest displacement coefficient, CU_n^α, are less than the tolerances. The default values are CR_n^α = 0.005 and CU_n^α = 0.005. To change these values, you must define direct cyclic controls.

When a stabilized cyclic response does not exist, the method will not converge. In the case where plastic ratchetting occurs, the displacement and residual coefficients of all the periodic terms (u_k^s, u_k^c, and R_k^c, R_k^c) in the Fourier series converge. However, the displacement and the residual coefficients of the constant term (u_0 and R_0) in the Fourier series continue to grow from one iteration to another iteration. The user-specified tolerances CR_0^α and CU_0^α are used to detect the plastic ratchetting. The default values are CR_0^α = 0.005 and CU_0^α = 0.005. For more information, see "Controlling the solution accuracy in direct cyclic analysis" in "Commonly used control parameters," Section 7.2.2.

Input File Usage: *CONTROLS, TYPE=DIRECT CYCLIC

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**; **Specify: Direct Cyclic**

Controlling the Fourier representations

The number of Fourier terms required to obtain an accurate solution depends on the variation of the load as well as the variation of the structural response over the period. In determining the number of terms, keep in mind that the objective of this kind of analysis is to make low-cycle fatigue predictions. Hence, the goal is to obtain good approximation of the plastic strain cycle at each point; local inaccuracies in the stresses are less important. More Fourier terms usually provide a more accurate solution but at the expense of additional data storage and computational time. In addition, an accurate integration of the Fourier residual coefficients requires that the residual vector be evaluated at an adequate number of time points during the cycle. Abaqus/Standard uses a trapezoidal rule, which assumes a linear variation of the residual over a time increment, to integrate the residual coefficients. For accurate integration the number of time points must be larger than the number of Fourier coefficients (which is equal to $2n + 1$, where n represents the number of Fourier terms). Abaqus/Standard will automatically reduce the number of Fourier coefficients used for the next iteration if it is found to be greater than the number of increments taken to complete an iteration.

Abaqus/Standard uses an adaptive algorithm to determine the number of Fourier terms. By default, Abaqus/Standard starts with 11 terms and determines the response of the structure by using the iterative method described before. Once convergence is obtained (which is measured by ensuring that all the residual vector coefficients and all the corrections to displacement coefficients in the Fourier series are sufficiently small), Abaqus/Standard evaluates if a sufficient number of Fourier terms are used by determining if equilibrium was satisfied at all the time points during the cycle. If equilibrium is satisfied at all time points, the solution is accepted. Otherwise, Abaqus/Standard increases the number of Fourier

terms (by default, 5 terms are added) and continues with the iterative scheme until convergence with the new number of Fourier terms is obtained. This process is repeated until equilibrium is reached or until the maximum number of Fourier terms has been used. This scheme is best illustrated in Figure 6.2.6–2, where both local equilibrium and overall convergence are obtained when the number of Fourier terms is equal to 21. A maximum number of 25 Fourier terms is used by default. You can specify the initial and maximum number of Fourier terms and the increment in the number of terms when you define the direct cyclic step.

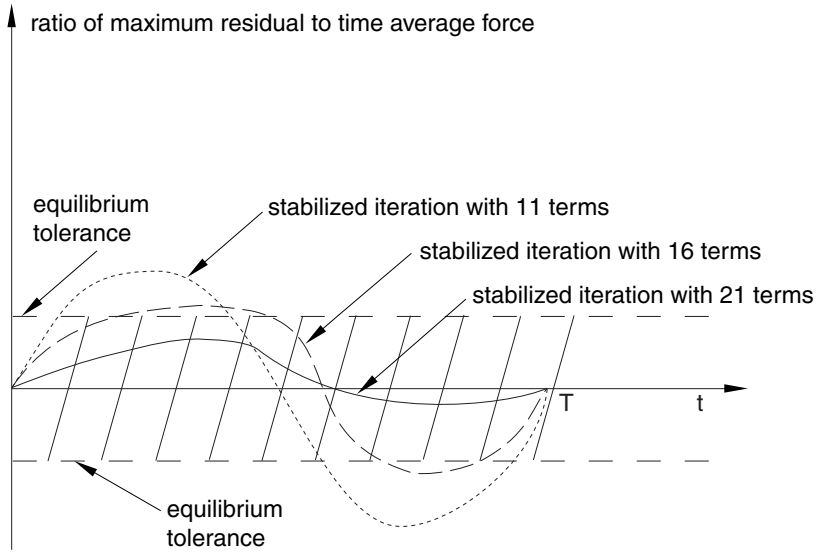


Figure 6.2.6–2 Stabilized iterations with different Fourier terms.

You can also define the convergence criteria for determining convergence and for determining whether equilibrium is achieved at all time points through the period (see "Commonly used control parameters," Section 7.2.2), with suitable defaults set by Abaqus/Standard.

In a direct cyclic analysis that has not reached a stabilized cycle, you can increase the number of iterations or Fourier terms upon restart, thus allowing continuation of an analysis.

Abaqus/Standard provides detailed output of the maximum residual at each time point, the maximum residual coefficient, the maximum displacement coefficient, the maximum correction to displacement coefficients, and the number of Fourier terms at the end of each iteration in the message (**`.msg`**) file. This output is described in more detail below.

Input File Usage: *DIRECT CYCLIC
,,,, *initial number of terms*, *max number of terms*, *increment in number of terms*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Number of Fourier terms: Initial:** *initial number of terms*, **Maximum:** *max number of terms*, **Increment:** *increment in number of terms*

Controlling the incrementation during the cyclic time period

To ensure an accurate solution, the material history as well as the residual vector must be evaluated at a sufficient number of time points during the cycle. The number of time points, n_T, at which the response is computed must be larger than the number of Fourier coefficients; i.e., $n_T > 2n + 1$. Abaqus/Standard will automatically adjust the number of Fourier coefficients if such a condition is not satisfied. You can specify the time incrementation over the cycle directly, or it can be determined automatically by Abaqus/Standard.

You should specify the maximum number of increments allowed in the time period as part of the step definition. The default is 100.

Automatic incrementation

There are several ways to choose the automatic incrementation scheme. If you specify only the maximum allowable nodal temperature change in an increment, the time increments are selected automatically based on this value. Abaqus/Standard will restrict the time increments to ensure that the maximum temperature change is not exceeded at any node during any increment of the analysis.

For rate-dependent constitutive equations you can limit the size of the time increment by the accuracy of the integration. The user-specified accuracy tolerance parameter limits the maximum inelastic strain rate change allowed over an increment:

$$tolerance \geq (\dot{\bar{\varepsilon}}^{cr}|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}|_t)\Delta t,$$

where t is the time at the beginning of the increment, Δt is the time increment (so that $t + \Delta t$ is the time at the end of the increment), and $\dot{\bar{\varepsilon}}^{cr}$ is the equivalent creep strain rate. To achieve sufficient accuracy, the value chosen for the accuracy tolerance parameter should be on the order of σ_{err}/E for creep problems, where σ_{err} is an acceptable level of error in the stress and E is a typical elastic modulus, or on the order of the elastic strains for viscoelasticity problems.

If rate-dependent constitutive equations are used in combination with a varying temperature, both controls can be used simultaneously. Abaqus/Standard will then choose the increments that satisfy both criteria.

If the time integration accuracy measure specified by either or both of the above controls is satisfied after I_T consecutive increments without cutbacks, the next time increment will be increased by a factor of D_M. Both I_T and D_M are user-defined parameters (see "Increasing the time increment size" in "Time integration accuracy in transient problems," Section 7.2.4). The defaults are I_T = 3 and D_M = 1.5.

Input File Usage: Use the following option to specify the maximum allowable nodal temperature change:

∗DIRECT CYCLIC, DELTMX=$\Delta\theta_{max}$

Use the following option to specify the accuracy tolerance parameter:

∗DIRECT CYCLIC, CETOL=*tolerance*

Abaqus/CAE Usage: Use the following option to specify the maximum allowable nodal temperature change:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Use the following option to specify the accuracy tolerance parameter:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Fixed time incrementation

If neither the accuracy tolerance parameter nor the maximum allowable nodal temperature change is specified, the size of the time increment is fixed. You must specify the time increment Δt and the time period T.

Input File Usage: *DIRECT CYCLIC
Δt, T

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Cycle time period:** T; **Incrementation: Type: Fixed**, **Increment size:** Δt

Defining the time points at which the response must be evaluated

The user-defined time incrementation for a direct cyclic step can be augmented or superseded by specifying particular time points in the loading history at which the response of the structure should be evaluated. This feature is particularly useful if you know prior to the analysis at which time points in the analysis the load reaches a maximum and/or minimum value or when the response will change rapidly. An example is the analysis of the heating/cooling thermal cycle of an engine component where you typically know when the temperature reaches a maximum value.

When time points are used with fixed time incrementation, the time incrementation specified for the direct cyclic step is ignored and instead the time incrementation precisely follows the specified time points. If time points are used with automatic incrementation, the time incrementation is variable; but the response of the structure will be evaluated at the specified time points.

The time points can be listed individually, or they can be generated automatically by specifying the starting time point, ending time point, and increment in time between the two specified time points.

Input File Usage: Use the following options to list time points individually:

*TIME POINTS, NAME=*time points name*
*DIRECT CYCLIC, TIME POINTS=*time points name*

Use the following options to generate time points automatically:

*TIME POINTS, NAME=*time points name*, GENERATE
*DIRECT CYCLIC, TIME POINTS=*time points name*

Abaqus/CAE Usage: Use the following options to list time points individually:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Evaluate structure response at time points:** *time points name*

Use the following options to generate time points automatically:

Step module: **Create Step**: **General**: **Direct cyclic**; **Incrementation: Evaluate structure response at time points: Create**; **Edit Time Points: Specify using delimiters: Start**, **End**, **Increment**

Controlling the application of periodicity conditions

By default, Abaqus/Standard imposes periodic conditions during the iterative solution process by using the state obtained at the end of the previous iteration as the starting state for the current iteration; i.e., $s^{i+1}_{t=0} = s^{i}_{t=T}$, where $\boldsymbol{s}$ is a solution variable such as plastic strain.

In cases where the periodic solution is not easily found (for example, when the loading is close to causing ratchetting), the state around which the periodic solution is obtained may show considerably more "drift" than would be obtained in a transient analysis. In such cases you may wish to delay the application of periodic conditions as an artificial method to reduce this drift. Figure 6.2.6–3 compares the response of two identical structures subjected to the same set of cyclic loads and boundary conditions, where each structure experienced a different loading history prior to the application of the cyclic loads. Figure 6.2.6–3 shows that the prior loading history only affects the mean value of stress and strain; it does not affect the shape of the stress-strain curves or the amount of energy dissipated during the cycle.

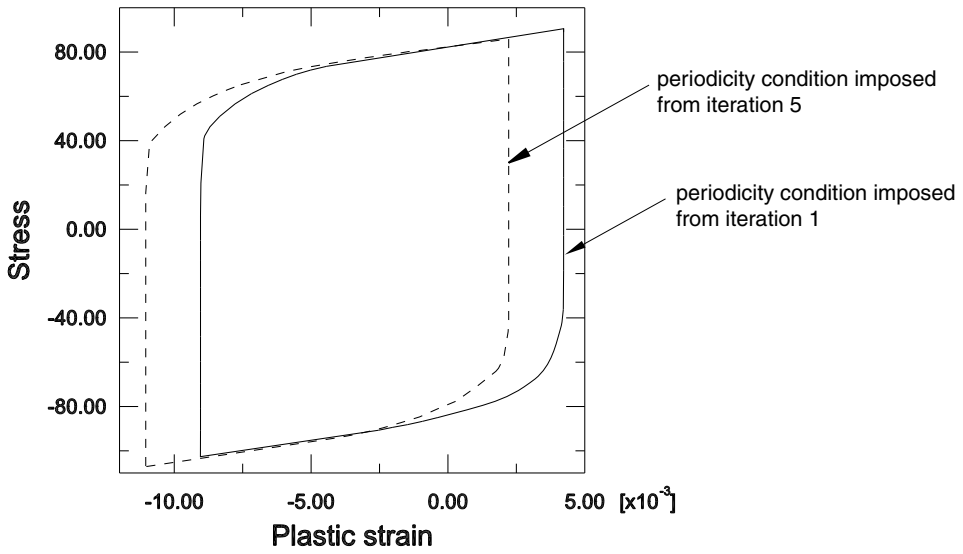


Figure 6.2.6–3 Influence of periodicity condition on mean value of the strains over a stabilized cycle.

By delaying the application of periodicity conditions, you can influence the mean stress and strain level. However, this is rarely necessary since the average stress and strain levels are usually not needed for low-cycle fatigue life predictions.

You can control when the periodicity conditions are applied by defining direct cyclic controls to specify the variable I_{PI}. This variable defines from which iteration onward the application of periodic conditions will be activated. For example, setting $I_{PI} = 6$ means that the periodicity conditions are applied from iteration 6 onwards. The default is $I_{PI} = 1$, which is appropriate for most analyses.

Input File Usage: *CONTROLS, TYPE=DIRECT CYCLIC
I_{PI}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**;
Direct Cyclic: I_{PI}

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom. During the analysis, prescribed boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the analysis consists of several steps, the usual rules apply (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). At each new step the boundary condition can either be modified or completely defined. All boundary conditions defined in previous steps remain unchanged unless they are redefined.

Loads

The following loads can be prescribed in a direct cyclic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

During the analysis each load must have an amplitude definition that is cyclic over the step where the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the analysis consists of several steps, the usual rules apply (see "Applying loads: overview," Section 34.4.1). At each new step the loading can either be modified or completely defined. All loads defined in previous steps remain unchanged unless they are redefined.

Predefined fields

The following predefined fields can be specified in a direct cyclic analysis, as described in "Predefined fields," Section 34.6.1:

- Temperature is not a degree of freedom in a direct cyclic analysis, but nodal temperatures can be specified as a predefined field. The temperature values specified must be cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Alternatively, you can ramp the temperatures back to their initial

condition values, as described in "Predefined fields," Section 34.6.1. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. The field variable values specified must be cyclic over the step.

Material options

Most material models, including user-defined materials (defined using user subroutine **UMAT**), that describe mechanical behavior are available for use in a direct cyclic analysis.

The following material properties are not active during a direct cyclic analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, piezoelectric properties, and pore fluid flow properties.

Rate-dependent yield ("Rate-dependent yield," Section 23.2.3), rate-dependent creep ("Rate-dependent plasticity: creep and swelling," Section 23.2.4), and two-layer viscoplasticity ("Two-layer viscoplasticity," Section 23.2.11) can also be used during a direct cyclic analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a direct cyclic analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3).

Output

Different types of output are available for postprocessing and for monitoring a direct cyclic analysis.

Message file information

Abaqus/Standard prints the residual force, time average force, and a flag to indicate if equilibrium was satisfied in the message (**.msg**) file at different time increments for each iteration. You can control the frequency in increments at which information is printed to the message file, and you can suppress the output; the default is to print output every 10 increments (see "The Abaqus/Standard message file" in "Output," Section 4.1.1, for more information).

Abaqus/Standard also prints the number of Fourier terms used, the maximum residual coefficient, the maximum correction to displacement coefficients, and the maximum displacement coefficient in the Fourier series in the message file at the end of each iteration. An example of the output is shown below:

```
                    ITERATION    26 STARTS
INC      TIME        STEP       LARG. RESI.    TIME AVG.    FORCE
         INC         TIME       FORCE          FORCE        EQUV.
10       0.250       2.50       1.008E+01      50.9          N
```

```
20      0.250       5.00       1.622E+01     76.8          N
30      0.250       7.50       4.622E-02     99.8          Y

                    ITERATION    26 SUMMARY
NUMBER OF FOURIER TERMS USED 40, TOTAL NUMBER OF INCREMENTS  120
CYCLE/STEP TIME   30.0,    TOTAL TIME COMPLETED       31.0
AVERAGE FORCE     21.2     TIME AVG. FORCE     25.7

MAX. COEFFICIENT OF DISP.                    0.142  AT NODE 24 DOF 2
MAX. COEFF. OF RESI. FORCE ON CONST. TERM     31.7  AT NODE 44 DOF 1
MAX. COEFF. OF RESI. FORCE ON PERI. TERMS     0.82  AT NODE  6 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON CONST. TERM 0.002 AT NODE 50 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON PERI. TERMS 0.015 AT NODE 50 DOF 3
```

Results output

Element and nodal output are written only when the stabilized cycle is reached. If a stabilized cycle has not been reached at the end of an analysis, output is written for the last iteration of the step. The element output available for a direct cyclic analysis includes stress; strain; energies; and the values of state, field, and user-defined variables. All the energies are set equal to zero at the beginning of each iteration since energies dissipated over an entire stabilized cycle are of interest in making fatigue life predictions in direct cyclic analysis. The nodal output available includes displacements, reaction forces, and coordinates. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Recovering additional results for an iteration

You may want to recover additional results for an iteration rather than for the stabilized cycle. You can extract these results from the restart data (see "Recovering additional results output from restart data in Abaqus/Standard" in "Output," Section 4.1.1). This feature is particularly useful if you want to evaluate the shift of the strain from one iteration to another iteration when plastic ratchetting occurs.

Input File Usage: *POST OUTPUT, ITERATION=n

Abaqus/CAE Usage: Recovering additional results for an iteration is not supported in Abaqus/CAE.

Specifying output at exact times

Output at exact times is not supported for direct cyclic analysis. If output at exact times is requested, Abaqus will issue a warning message and change the output to an output at approximate times.

Limitations

A direct cyclic analysis is subject to the following limitations:

- Contact conditions cannot change during a direct cyclic analysis; they remain as they were defined at the beginning of the analysis or at the end of any general step prior to the direct cyclic step. Frictional slipping is not allowed during direct cyclic analyses; all points in contact are assumed to be sticking if friction is present.
- A direct cyclic step is always performed using the original coordinates of a model, even when the direct cyclic step follows a geometrically nonlinear step. To perform a direct cyclic analysis on the updated coordinates, you can use the import capability to import both the current state as well as the current configuration from the end of the desired geometrically nonlinear step.

Input file template

```
*HEADING
…
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
*AMPLITUDE
Data lines to define amplitude variations
**
*STEP (,INC=)
Set INC equal to the maximum number of increments in a single loading cycle
*DIRECT CYCLIC
Data line to define time increment, cycle time, initial number of Fourier terms,
maximum number of Fourier terms, increment in number of Fourier terms,
and maximum number of iterations
*TIME POINTS
Data lines to list time points
*BOUNDARY, AMPLITUDE=
Data lines to prescribe zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD, AMPLITUDE=
Data lines to specify loads
*TEMPERATURE and/or *FIELD, AMPLITUDE=
Data lines to specify values of predefined fields
*END STEP
**
*STEP(,INC=)
*DIRECT CYCLIC, DELTMX
Data line to control automatic time incrementation and Fourier representations
*BOUNDARY, OP=MOD,AMPLITUDE=
Data lines to modify or add zero-valued or nonzero boundary conditions
*CLOAD, OP=NEW, AMPLITUDE=
```

Data lines to specify new concentrated loads; all previous concentrated loads will be removed
***DLOAD, OP=MOD, AMPLITUDE=**
Data lines to specify additional or modified distributed loads
***TEMPERATURE and/or *FIELD, AMPLITUDE=**
Data lines to specify additional or modified values of predefined fields
***END STEP**

6.2.7 LOW-CYCLE FATIGUE ANALYSIS USING THE DIRECT CYCLIC APPROACH

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Static stress analysis procedures: overview," Section 6.2.1
- "Direct cyclic analysis," Section 6.2.6
- "Crack propagation analysis," Section 11.4.3
- "Damage and failure for ductile materials in low-cycle fatigue analysis," Section 24.4
- "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1
- *DAMAGE EVOLUTION
- *DAMAGE INITIATION
- *DEBOND
- *DIRECT CYCLIC
- *FRACTURE CRITERION
- *CONTROLS
- "Configuring a direct cyclic procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A low-cycle fatigue analysis:

- is characterized by states of stress high enough for inelastic deformation to occur in most cases;
- is a quasi-static analysis on a structure subjected to sub-critical cyclic loading;
- can be associated with thermal as well as mechanical loading;
- uses the direct cyclic approach to obtain the stabilized cyclic response of the structure directly;
- models progressive damage and failure in bulk ductile material based on a continuum damage mechanics approach, in which case damage initiation and evolution are characterized by the accumulated inelastic hysteresis strain energy per stabilized cycle;
- models propagation of a discrete crack along an arbitrary, solution-dependent path without remeshing in the bulk material based on the principles of linear elastic fracture mechanics (LEFM) with the extended finite element method, in which case the onset and growth of fatigue crack are characterized by the relative fracture energy release rate;

- models progressive delamination growth along a predefined path at the interfaces in laminated composites, in which case the onset and growth of fatigue delamination at the interfaces are characterized by the relative fracture energy release rate;
- uses the damage extrapolation technique to accelerate the low-cycle fatigue analysis; and
- assumes geometrically linear behavior and fixed contact conditions within each loading cycle.

Approaches to low-cycle fatigue analysis

The traditional approach for determining the fatigue limit for a structure is to establish the $S - N$ curves (load versus number of cycles to failure) for the materials in the structure. Such an approach is still used as a design tool in many cases to predict fatigue resistance of engineering structures. However, this technique is generally conservative, and it does not define a relationship between the cycle number and the degree of damage or crack length.

One alternative approach is to predict the fatigue life by using a crack/damage evolution law based on the inelastic strain/energy when the structure's response is stabilized after many cycles. Because the computational cost to simulate the slow progressive damage in a material over many load cycles is prohibitively expensive for all but the simplest models, numerical fatigue life studies usually involve modeling the response of the structure subjected to a small fraction of the actual loading history. This response is then extrapolated over many load cycles using empirical formulae such as the Coffin-Manson relationship (see Coffin, 1954, and Manson, 1953) to predict the likelihood of crack initiation and propagation. Since this approach is based on a constant crack/damage growth rate, it may not realistically predict the evolution of the crack or damage.

Low-cycle fatigue analysis in Abaqus/Standard

The direct cyclic analysis capability in Abaqus/Standard provides a computationally effective modeling technique to obtain the stabilized response of a structure subjected to periodic loading and is ideally suited to perform low-cycle fatigue calculations on a large structure. The capability uses a combination of Fourier series and time integration of the nonlinear material behavior to obtain the stabilized response of the structure directly. The theory and algorithm to obtain a stabilized response using the direct cyclic approach are described in detail in "Direct cyclic algorithm," Section 2.2.3 of the Abaqus Theory Guide.

The direct cyclic low-cycle fatigue procedure models the progressive damage and failure both in bulk materials (such as in solder joints in an electronic chip packaging or intra-laminar crack growth in laminated composites) and at material interfaces (such as delamination in laminated composites). The former can be based on either a continuum damage mechanics approach or the principles of linear elastic fracture mechanics with the extended finite element method. The response is obtained by evaluating the behavior of the structure at discrete points along the loading history (see Figure 6.2.7–1). The solution at each of these points is used to predict the degradation and evolution of material properties that will take place during the next increment, which spans a number of load cycles, ΔN. The degraded material properties are then used to compute the solution at the next increment in the load history. Therefore, the crack/damage growth rate is updated continually throughout the analysis.

The elastic material stiffness at a material point remains constant and contact conditions remain unchanged when the stabilized solution is computed at a given point in the loading history. Each of the

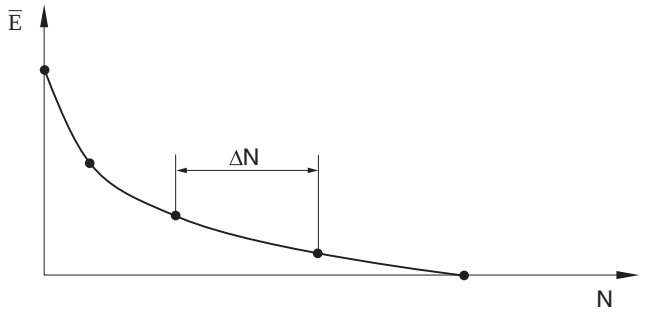


Figure 6.2.7–1 Elastic stiffness degradation as a function of the cycle number.

solutions along the loading history represents the stabilized response of the structure subjected to the applied period loads, with a level of material damage at each point in the structure computed from the previous solution. This process is repeated up to a point in the loading history at which a fatigue life assessment can be made.

In bulk material, there are two approaches to modeling the progressive damage and failure. One approach is based on continuum damage mechanics. This approach is more appropriate for ductile material, in which the cyclic loading leads to stress reversals and the accumulation of plastic strains, which in turn cause the initiation and propagation of cracks. The damage initiation and evolution are characterized by the stabilized accumulated inelastic hysteresis strain energy per cycle as illustrated in Figure 6.2.7–2. The other approach is based on the principles of linear elastic fracture mechanics with the extended finite element method. This approach is more appropriate for brittle material or material with small scale yielding, in which the cyclic loading leads to material strength degradation causing fatigue crack growth along an arbitrary path. The onset and growth of the crack are characterized by the relative fracture energy release rate at the crack tip based on the Paris law (Paris, 1961).

At interfaces of laminated composites the cyclic loading leads to interface strength degradation causing fatigue delamination growth. The onset and growth of delamination are also characterized by the relative fracture energy release rate at the crack tip based on the Paris law (Paris, 1961).

Both the progressive damage mechanism in the bulk material and the progressive delamination growth mechanism at interfaces can be considered simultaneously, with the failure occurring first at the weakest link in a model.

Defining a low-cycle fatigue analysis using the direct cyclic approach is similar to defining a direct cyclic analysis. See "Direct cyclic analysis," Section 6.2.6, for details on how to specify the number of Fourier terms, number of iterations, and the increment sizes. You specify the maximum numbers of cycles, N_{max}, when you define the low-cycle fatigue analysis step.

Input File Usage: ∗DIRECT CYCLIC, FATIGUE
first data line
, , N_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Maximum number of cycles: Value:** N_{max}

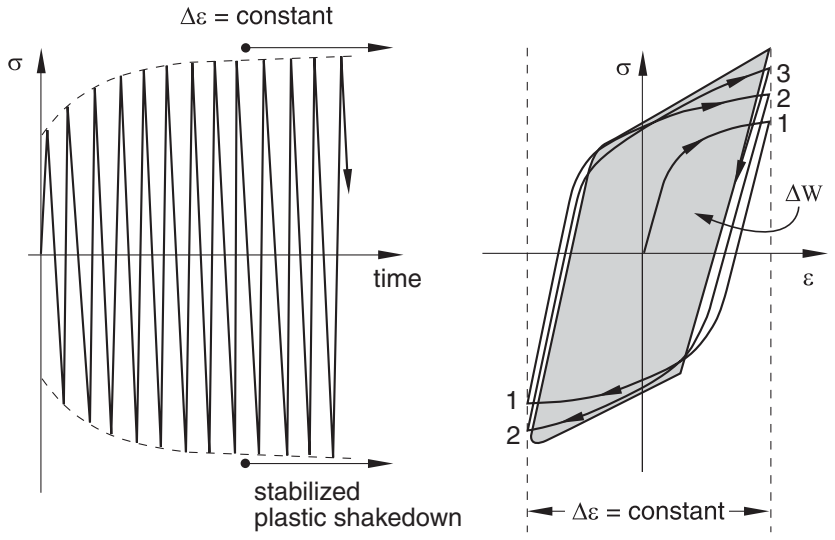


Figure 6.2.7–2 Plastic shakedown in a direct cyclic analysis.

Determining whether to use the Fourier coefficients from the previous step

A low-cycle fatigue step using the direct cyclic approach can be the only step in an analysis, can follow a general or linear perturbation step, or can be followed by a general or linear perturbation step. Multiple low-cycle fatigue analysis steps can be included in a single analysis. In such a case the Fourier series coefficients obtained in the previous step can be used as starting values in the current step. By default, the Fourier coefficients are reset to zero, thus allowing application of cyclic loading conditions that are very different from those defined in the previous low-cycle fatigue step.

As in a direct cyclic analysis, you can specify that a low-cycle fatigue step in a restart analysis should use the Fourier coefficients from the previous step, thus allowing continuation of an analysis to simulate more loading cycles. In a low-cycle fatigue analysis a restart file is written at the end of the stabilized cycle. Consequently, a restart analysis that is a continuation of a previous low-cycle fatigue analysis will start with a new loading cycle at $t = 0$ (see "Restarting an analysis," Section 9.1.1).

Input File Usage: Use the following option to specify that the current step is a continuation of the previous low-cycle fatigue step using the direct cyclic approach:

*DIRECT CYCLIC, FATIGUE, CONTINUE=YES

Use the following option to reset the Fourier series coefficients to zero:

*DIRECT CYCLIC, FATIGUE, CONTINUE=NO (default)

Abaqus/CAE Usage: Use the following option to specify that the current step is a continuation of the previous low-cycle fatigue step using the direct cyclic approach:

Step module: **Create Step**: **General**: **Direct cyclic**; **Basic: Use displacement Fourier coefficients from previous direct cyclic step**; **Fatigue: Include low-cycle fatigue analysis**

Use the following option to reset the Fourier series coefficients to zero:

Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**

Progressive damage and damage extrapolation in bulk ductile material based on continuum damage mechanics approach

Low-cycle fatigue analysis in Abaqus/Standard allows modeling of progressive damage and failure for ductile materials in any elements whose response is defined in terms of a continuum-based constitutive model ("Material library: overview," Section 21.1.1). This includes cohesive elements modeled using a continuum approach ("Modeling of an adhesive layer of finite thickness" in "Defining the constitutive response of cohesive elements using a continuum approach," Section 32.5.5). The inelastic definition in a material point must be used in conjunction with the linear elastic material model ("Linear elastic behavior," Section 22.2.1), the porous elastic material model ("Elastic behavior of porous materials," Section 22.3.1), or the hypoelastic material model ("Hypoelastic behavior," Section 22.4.1).

After damage initiation the elastic material stiffness is degraded progressively in each cycle (as shown in Figure 6.2.7–1) based on the accumulated stabilized inelastic hysteresis energy. It is impractical and computationally expensive to perform a cycle-by-cycle simulation for a low-cycle fatigue analysis; Instead, to accelerate the low-cycle fatigue analysis, each increment extrapolates the current damaged state in the bulk material forward over many cycles to a new damaged state after the current loading cycle is stabilized.

Damage initiation and evolution

Damage initiation refers to the beginning of degradation of the response of a material point. In a low-cycle fatigue analysis the damage initiation criterion is characterized by the accumulated inelastic hysteresis energy per cycle, Δw. Δw and material constants are used to determine the number of the cycle in which damage is initiated, N_0. At the end of a stabilized loading cycle, N, Abaqus/Standard checks to see if the damage initiation criterion $N > N_0$ is satisfied in any material point; material stiffness at a material point will not be degraded unless this criterion is satisfied. The calculations and output associated with damage initiation are discussed in detail in "Damage initiation for ductile materials in low-cycle fatigue," Section 24.4.2.

Once the damage initiation criterion is satisfied at a material point, the damage state is calculated and updated based on the inelastic hysteresis energy for the stabilized cycle. Abaqus/Standard assumes that the degradation of the elastic stiffness can be modeled using the scalar damage variable, D. The rate of the damage in a material point per cycle, $\frac{dD}{dN}$, is calculated based on the accumulated inelastic hysteresis energy, the characteristic length associated with an integration point, and material constants. For details, see "Damage evolution for ductile materials in low-cycle fatigue," Section 24.4.3.

Typically, a material has completely lost its load carrying capacity when $D = 1$. You can remove an element from the mesh if all of the section points at all integration locations of the element have lost their load carrying capability.

Damage extrapolation technique in the bulk material

If the damage initiation criterion is satisfied in any material point at the end of a stabilized cycle, N, Abaqus/Standard extrapolates the damage variable D_N from the current cycle forward to the next increment over a number of cycles, ΔN. The new damage state, $D_{N+\Delta N}$, is given by

$$D_{N+\Delta N} = D_N + \frac{\Delta N}{L} c_3 \Delta w^{c_4},$$

where L is the characteristic length associated with an integration point, and c_3 and c_4 are material constants (see "Damage evolution for ductile materials in low-cycle fatigue," Section 24.4.3, for more information).

You specify the minimum (ΔN_{min}) and maximum (ΔN_{max}) number of cycles over which the damage is extrapolated forward in any given increment. The default values are 100 and 1000, respectively.

Input File Usage: *DIRECT CYCLIC, FATIGUE
first data line
ΔN_{min}, ΔN_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Cycle increment size: Minimum:** ΔN_{min}, **Maximum:** ΔN_{max}

Discrete crack propagation along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method

Low-cycle fatigue analysis in Abaqus/Standard allows the modeling of discrete crack growth along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method. You complete the definition of the crack propagation capability by defining a fracture-based surface behavior and specifying the fracture criterion in enriched elements. The fracture energy release rates at the crack tips in enriched elements are calculated based on the modified virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue crack growth occurs, although nonlinear material deformations can occur somewhere else in the bulk materials. For more information about defining fracture criteria and VCCT in enriched elements, see "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1.

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used, which advances the crack by at least one element length after each stabilized cycle.

Onset and growth of fatigue crack

The onset and growth of fatigue crack at an enriched element are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue crack growth: one criterion is based on material constants, ΔG, and the current cycle number, N; the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value. Once the onset of fatigue crack growth criterion is satisfied at the enriched elements, the crack growth rate, $\frac{da}{dN}$, is a piecewise function based on material constants and ΔG (the Paris law). The criteria for fatigue crack onset and growth are discussed in detail in "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1.

Damage extrapolation technique

If the onset of crack growth criterion is satisfied at any crack tip in the enriched element at the end of a stabilized cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by fracturing at least one enriched element ahead of the crack tips. Given the material constants c_3 and c_4 (as defined in "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1), combined with the known element length and likely propagation direction $\Delta a_{N\,j} = a_{N+\Delta N} - a_N$ at the enriched elements ahead of the crack tips, the number of cycles necessary to fail each enriched element ahead of the crack tip can be calculated as ΔN_j, where j represents the enriched element ahead of the jth crack tip. The analysis is set up to advance the crack by at least one enriched element per increment after the loading cycle is stabilized. The element with the fewest cycles is identified to be fractured, and its $\Delta N_{min} = min(\Delta N_j)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{N\,min} = min(\Delta a_{N\,j})$. The most critical element is completely fractured with a zero constraint and a zero stiffness at the cracked surfaces at the end of the stabilized cycle. As the enriched element is fractured, the load is redistributed, and a new relative fracture energy release rate must be calculated for the enriched elements ahead of the crack tips for the next cycle. This capability allows at least one enriched element ahead of the crack tips to be fractured after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Progressive delamination growth along a pre-defined path at interfaces

Low-cycle fatigue analysis in Abaqus/Standard also allows the modeling of progressive delamination growth at the interfaces in laminated composites. The interface along which the delamination (or crack) propagates must be indicated in the model using a fracture criterion definition. The fracture energy release rates at the crack tips in the interface elements are calculated based on the virtual crack closure technique (VCCT). VCCT uses the principles of linear elastic fracture mechanics. Therefore, VCCT is appropriate for problems in which brittle fatigue delamination growth occurs along predefined surfaces, although nonlinear material deformations can occur in the bulk materials. For more information about defining fracture criteria and VCCT, see "Crack propagation analysis," Section 11.4.3.

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used, which releases at least one element length at the crack tip along the interface after each stabilized cycle. When

both brittle fatigue delamination at interfaces and ductile damage or discrete crack growth in bulk materials are considered in an analysis, failure occurs first at the weakest link.

Onset and growth of fatigue delamination

The onset and growth of fatigue delamination at a defined crack interface are characterized by using the Paris law, which relates the relative fracture energy release rate, ΔG, to crack growth rates. Two criteria must be met to initiate fatigue delamination growth: one criterion is based on material constants, ΔG, and the current cycle number, N; the other criterion is based on the maximum fracture energy release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value. Once the onset of delamination growth criterion is satisfied at the interface, the delamination growth rate, $\frac{da}{dN}$, is a piecewise function based on material constants and ΔG (the Paris law). The criteria for fatigue delamination onset and growth are discussed in detail in "Low-cycle fatigue criterion" in "Crack propagation analysis," Section 11.4.3.

Damage extrapolation technique at the interface elements

If the onset of delamination growth criterion is satisfied at any crack tip in the interface at the end of a stabilized cycle, N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over a number of cycles, ΔN, to $a_{N+\Delta N}$ by releasing at least one element at the interface. Given the material constants c_3 and c_4 (as defined in "Low-cycle fatigue criterion" in "Crack propagation analysis," Section 11.4.3), combined with the known node spacing $\Delta a_{N\,j} = a_{N+\Delta N} - a_N$ at the interface elements at the crack tips, the number of cycles necessary to fail each interface element at the crack tip can be calculated as ΔN_j, where j represents the node at the jth crack tip. The analysis is set up to release at least one interface element per increment after the loading cycle is stabilized. The element with the fewest cycles is identified to be released, and its $\Delta N_{min} = min(\Delta N_j)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{N\,min} = min(\Delta a_{N\,j})$. The most critical element is completely released with a zero constraint and a zero stiffness at the end of the stabilized cycle. As the interface element is released, the load is redistributed, and a new relative fracture energy release rate must be calculated for the interface elements at the crack tips for the next cycle. This capability allows at least one interface element at the crack tips to be released after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

Controlling the solution accuracy

Low-cycle fatigue analysis utilizes the direct cyclic approach to obtain the stabilized cyclic solution iteratively by combining a Fourier series approximation with time integration of the nonlinear material behavior using a modified Newton method. The accuracy of the algorithm depends on the number of Fourier terms used, the number of iterations taken to obtain the stabilized solution, and the number of time points within the load period at which the material response and residual vector are evaluated. Some methods for controlling the solution accuracy in a direct cyclic analysis are described in detail in "Direct cyclic analysis," Section 6.2.6. They all remain valid in a low-cycle fatigue analysis using the direct cyclic approach. In addition, the accuracy of a low-cycle fatigue analysis depends on the number of cycles over which the damage is extrapolated forward, as described below.

Controlling the accuracy of damage extrapolation in the bulk material when using continuum damage mechanics approach

To accelerate the low-cycle fatigue analysis, the damage extrapolation technique is used at the end of a stabilized cycle. In addition to specifying the minimum and maximum number of cycles over which the damage is extrapolated (see "Damage extrapolation technique in the bulk material" above), you can specify the damage extrapolation tolerance, ΔD_{tol}, to control the accuracy of damage extrapolation in the bulk material. The default is $\Delta D_{tol} = 1.0$.

Input File Usage: Use the following option to specify the damage extrapolation tolerance:

*DIRECT CYCLIC, FATIGUE
first data line
, , , ΔD_{tol}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Direct cyclic**; **Fatigue: Include low-cycle fatigue analysis**, **Damage extrapolation tolerance:** ΔD_{tol}

Determining the increment over which damage is extrapolated forward

Abaqus/Standard uses an adaptive algorithm to determine the number of cycles over which the damage is extrapolated forward in each increment. By default, Abaqus/Standard starts with 500 cycles (half of the default value of maximum increment in number of cycles) and determines the maximum damage increment at any material points based on

$$\Delta D = \frac{\Delta N c_3 \Delta w^{c_4}}{L} < \Delta D_{tol}.$$

If the maximum damage increment, ΔD, is greater than the damage extrapolation tolerance that you specify, the number of cycles over which the damage is extrapolated forward is reduced accordingly to ensure the maximum damage increment is less than the damage extrapolation tolerance. On the other hand, if the maximum damage increment at all material points is less than half of the damage extrapolation tolerance that you specify, the number of cycles is increased accordingly to ensure the maximum damage increment is equal to the damage extrapolation tolerance.

Controlling the accuracy of damage extrapolation at the interface elements and at the enriched elements

To accelerate the low-cycle fatigue analysis and to provide a smooth solution for the crack front, you can specify a nonzero tolerance, ΔD_{Ntol}, for the least number of cycles to fracture an interface element or an enriched element:

$$\frac{Log\Delta N_j - Log\Delta N_{min}}{Log\Delta N_{min}} \leq \Delta D_{Ntol}.$$

In addition to the interface element or the enriched element that takes the fewest cycles, ΔN_{min}, to be fractured, all the other interface or enriched elements that are satisfied by the above condition will be fractured completely.

Input File Usage: Use the following option to specify the tolerance for the least number of cycles to fracture an element:

*DIRECT CYCLIC, FATIGUE
first data line
, , , , ΔD_{Ntol}

Abaqus/CAE Usage: Specifying a tolerance for the least number of cycles to fracture an element is not supported in Abaqus/CAE.

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom. During the analysis, prescribed boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the analysis consists of several steps, the usual rules apply (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). At each new step the boundary condition can either be modified or completely defined. All boundary conditions defined in previous steps remain unchanged unless they are redefined.

Loads

The following loads can be prescribed in a low-cycle fatigue analysis using the direct cyclic approach:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

During the analysis each load must have an amplitude definition that is cyclic over the step where the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the analysis consists of several steps, the usual rules apply (see "Applying loads: overview," Section 34.4.1). At each new step the loading can either be modified or completely defined. All loads defined in previous steps remain unchanged unless they are redefined.

Predefined fields

The following predefined fields can be specified in a low-cycle fatigue analysis using the direct cyclic approach, as described in "Predefined fields," Section 34.6.1:

- Temperature is not a degree of freedom in a low-cycle fatigue analysis using the direct cyclic approach, but nodal temperatures can be specified as a predefined field. The temperature values specified must be cyclic over the step: the start value must be equal to the end value (see

"Amplitude curves," Section 34.1.2). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Alternatively, you can ramp the temperatures back to their initial condition values, as described in "Predefined fields," Section 34.6.1. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any. The field variable values specified must be cyclic over the step.

Material options

Most ductile material models that describe mechanical behavior are available for use in a low-cycle fatigue analysis. The inelastic definition in a material point must be used in conjunction with the linear elastic material model ("Linear elastic behavior," Section 22.2.1), the porous elastic material model ("Elastic behavior of porous materials," Section 22.3.1), or the hypoelastic material model ("Hypoelastic behavior," Section 22.4.1).

The following material properties are not active during a low-cycle fatigue analysis: acoustic properties, thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, piezoeletric properties, and pore fluid flow properties.

Rate-dependent yield ("Rate-dependent yield," Section 23.2.3), rate-dependent creep ("Rate-dependent plasticity: creep and swelling," Section 23.2.4), and two-layer viscoplasticity ("Two-layer viscoplasticity," Section 23.2.11) can also be used during a low-cycle fatigue analysis.

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a low-cycle fatigue analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). This includes cohesive elements with finite thickness ("Modeling of an adhesive layer of finite thickness" in "Defining the constitutive response of cohesive elements using a continuum approach," Section 32.5.5). However, when modeling fatigue crack growth based on the principles of linear elastic fracture mechanics with the extended finite element method, only first-order continuum stress/displacement elements and second-order stress/displacement tetrahedron elements can be associated with an enriched feature (see "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1).

Output

Different types of output are available for postprocessing and for monitoring a low-cycle fatigue analysis using the direct cyclic approach.

Message file information

As in a direct cyclic analysis, low-cycle fatigue analysis using the direct cyclic approach in Abaqus/Standard prints the residual force, time average force, and a flag to indicate if equilibrium was satisfied in the message (**.msg**) file at different time increments for each iteration in each loading cycle. You can control the frequency in increments at which information is printed to the message file, and you can suppress the output; the default is to print output every 10 increments (see "The Abaqus/Standard message file" in "Output," Section 4.1.1, for more information).

Abaqus/Standard also prints the number of Fourier terms used, the maximum residual coefficient, the maximum correction to displacement coefficients, and the maximum displacement coefficient in the Fourier series in the message file at the end of each iteration in each cycle. An example of the output is shown below:

```
                        CYCLE  5 STARTS

                     ITERATION    26 STARTS
INC      TIME        STEP        LARG. RESI.    TIME AVG.    FORCE
         INC         TIME        FORCE          FORCE        EQUV.
10       0.250       2.50        1.008E+01      50.9          N
20       0.250       5.00        1.622E+01      76.8          N
30       0.250       7.50        4.622E-02      99.8          Y

                     ITERATION    26 SUMMARY
NUMBER OF FOURIER TERMS USED 40, TOTAL NUMBER OF INCREMENTS  120
CYCLE/STEP TIME   30.0,    TOTAL TIME COMPLETED       31.0
AVERAGE FORCE     21.2     TIME AVG. FORCE     25.7


MAX. COEFFICIENT OF DISP.                    0.142  AT NODE 24 DOF 2
MAX. COEFF. OF RESI. FORCE ON CONST. TERM    31.7  AT NODE 44 DOF 1
MAX. COEFF. OF RESI. FORCE ON PERI. TERMS    0.82  AT NODE  6 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON CONST. TERM 0.002 AT NODE 50 DOF 3
MAX. CORR. TO COEFF. OF DISP. ON PERI. TERMS 0.015 AT NODE 50 DOF 3
```

Results output

Element and nodal output are written only when the stabilized cycle is reached. If a stabilized cycle has not been reached at the end of a cycle, output is written for the last iteration of the cycle. All standard output variables in Abaqus/Standard ("Abaqus/Standard output variable identifiers," Section 4.2.1) are available. In addition, the following variables are available for progressive damage in bulk ductile material based on the continuum damage mechanics approach:

STATUS	Status of element (the status of an element is 1.0 if the element is active, 0.0 if the element is not).

SDEG	Scalar stiffness degradation, D.
CYCLEINI	Number of cycles to initialize the damage at the material point.

The following variables are available for discrete crack propagation along an arbitrary path based on the principles of linear elastic fracture mechanics with the extended finite element method:

STATUSXFEM	Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked, 0.0 if the element is not. If the element is partially cracked, the value lies between 1.0 and 0.0.)
CYCLEINIXFEM	Number of cycles to initialize the crack at the enriched element.
ENRRTXFEM	All components of strain energy release rate range; i.e., the difference between the energy release rate at the maximum loading and at the minimum loading.

Recovering additional results for a stabilized cycle

You may want to recover additional results for a stabilized cycle. You can extract these results from the restart data (see "Recovering additional results output from restart data in Abaqus/Standard" in "Output," Section 4.1.1).

Input File Usage: *POST OUTPUT, CYCLE=n

Abaqus/CAE Usage: Recovering additional results for a stabilized cycle is not supported in Abaqus/CAE.

Specifying output at exact times

Output at exact times is not supported for low-cycle fatigue analysis. If output at exact times is requested, Abaqus will issue a warning message and change the output to an output at approximate times.

Limitations

A low-cycle fatigue analysis using the direct cyclic approach is subject to the following limitations:

- Contact conditions cannot change during a given cycle when direct cyclic analysis is used iteratively to obtain a stabilized solution.
- The analysis may not perform well when there is compressive load on the crack surface during a loading cycle because the global stiffness is formed only one time at the beginning of each given loading cycle.
- Geometric nonlinearity can be included only in any general step prior to a direct cyclic step; however, only small displacements and strains will be considered during the cyclic step.

Input file template

The following is an example of modeling progressive damage and failure in the bulk material based on the continuum damage mechanics approach and progressive delamination growth at the interface:

***HEADING**
…
***BOUNDARY**
Data lines to specify zero-valued boundary conditions
***INITIAL CONDITIONS**
Data lines to specify initial conditions
***AMPLITUDE**
Data lines to define amplitude variations
******
***MATERIAL**
Options to define material properties
***DAMAGE INITIATION, CRITERION=HYSTERESIS ENERGY**
Data lines to define material constants for bulk ductile material damage initiation
***DAMAGE EVOLUTION, TYPE=HYSTERESIS ENERGY**
Data lines to define material constants for bulk ductile material damage evolution
******
SURFACE, NAME=slave*
Data lines to define slave surface at delamination interface
SURFACE, NAME=master*
Data lines to define master surface at delamination interface
***CONTACT PAIR**
slave, master
******
***STEP (,INC=)**
Set INC equal to the maximum number of increments in a single loading cycle
***DIRECT CYCLIC, FATIGUE**
Data line to define time increment, cycle time, initial number of Fourier terms, maximum number of Fourier terms, increment in number of Fourier terms, and maximum number of iterations
Data line to define minimum increment in number of cycles, maximum increment in number of cycles, total number of cycles, and damage extrapolation tolerance
DEBOND, SLAVE=slave***, MASTER=***master*
***FRACTURE CRITERION, TYPE=FATIGUE**
Data lines to define material constants used in Paris law and fracture criterion
******
***BOUNDARY, AMPLITUDE=**

Data lines to prescribe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD, AMPLITUDE=**
Data lines to specify loads
***TEMPERATURE and/or *FIELD, AMPLITUDE=**
Data lines to specify values of predefined fields
******
***END STEP**

The following is an example of modeling discrete crack growth in the bulk material based on the principles of linear elastic fracture mechanics with the extended finite element method and progressive delamination growth at the interface:

***HEADING**
…
***ENRICHMENT, TYPE=PROPAGATION CRACK, INTERACTION=INTERACTION, ELSET=ENRICHED**
***BOUNDARY**
Data lines to specify zero-valued boundary conditions
***INITIAL CONDITIONS**
Data lines to specify initial conditions
***AMPLITUDE**
Data lines to define amplitude variations
******
***MATERIAL**
Options to define material properties
***SURFACE, INTERACTION=INTERACTION**
***SURFACE BEHAVIOR**
***FRACTURE CRITERION, TYPE=FATIGUE**
Data lines to define material constants used in the Paris law and fracture criterion in the bulk material for enriched elements
******
SURFACE, NAME=slave*
Data lines to define slave surface at delamination interface
SURFACE, NAME=master*
Data lines to define master surface at delamination interface
***CONTACT PAIR**
slave, master
******
***STEP (,INC=)**
Set INC equal to the maximum number of increments in a single loading cycle
***DIRECT CYCLIC, FATIGUE**
Data line to define time increment, cycle time, initial number of Fourier terms, maximum number of Fourier terms, increment in number of Fourier terms,

and maximum number of iterations
Data line to define minimum increment in number of cycles,
maximum increment in number of cycles, total number of cycles,
and damage extrapolation tolerance
DEBOND, SLAVE=slave***, MASTER=***master*
***FRACTURE CRITERION, TYPE=FATIGUE**
Data lines to define material constants used in the Paris law and fracture criterion at the interface
******
***BOUNDARY, AMPLITUDE=**
Data lines to prescribe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD, AMPLITUDE=**
Data lines to specify loads
***TEMPERATURE and/or *FIELD, AMPLITUDE=**
Data lines to specify values of predefined fields
******
***END STEP**

Additional references

- Coffin, L., "A Study of the Effects of Cyclic Thermal Stresses on a Ductile Metal," Transactions of the American Society of Mechanical Engineering, vol. 76, pp. 931–951, 1954.
- Manson, S., "Behavior of Materials under Condition of Thermal Stress," Heat Transfer Symposium, University of Michigan Engineering Research Institute, Ann Arbor, MI, pp. 9–75, 1953.
- Paris, P., M. Gomaz, and W. Anderson, "A Rational Analytic Theory of Fatigue," The Trend in Engineering, vol. 15, 1961.

6.3 Dynamic stress/displacement analysis

- "Dynamic analysis procedures: overview," Section 6.3.1
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Explicit dynamic analysis," Section 6.3.3
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Complex eigenvalue extraction," Section 6.3.6
- "Transient modal dynamic analysis," Section 6.3.7
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Subspace-based steady-state dynamic analysis," Section 6.3.9
- "Response spectrum analysis," Section 6.3.10
- "Random response analysis," Section 6.3.11

6.3.1 DYNAMIC ANALYSIS PROCEDURES: OVERVIEW

Overview

Abaqus offers several methods for performing dynamic analysis of problems in which inertia effects are considered. Direct integration of the system must be used when nonlinear dynamic response is being studied. Implicit direct integration is provided in Abaqus/Standard; explicit direct integration is provided in Abaqus/Explicit. Modal methods are usually chosen for linear analyses because in direct-integration dynamics the global equations of motion of the system must be integrated through time, which makes direct-integration methods significantly more expensive than modal methods. Subspace-based methods are provided in Abaqus/Standard and offer cost-effective approaches to the analysis of systems that are mildly nonlinear.

In Abaqus/Standard dynamic studies of linear problems are generally performed by using the eigenmodes of the system as a basis for calculating the response. In such cases the necessary modes and frequencies are calculated first in a frequency extraction step. The mode-based procedures are generally simple to use; and the dynamic response analysis itself is usually not expensive computationally, although the eigenmode extraction can become computationally intensive if many modes are required for a large model. The eigenvalues can be extracted in a prestressed system with the "stress stiffening" effect included (the initial stress matrix is included if the base state step definition included nonlinear geometric effects), which may be necessary in the dynamic study of preloaded systems. It is not possible to prescribe nonzero displacements and rotations directly in mode-based procedures. The method for prescribing motion in mode-based procedures is explained in "Base motions in modal-based procedures," Section 2.5.9 of the Abaqus Theory Guide.

Density must be defined for all materials used in any dynamic analysis, and damping (both viscous and structural) can be specified either at the material or step level, as described below in "Damping in dynamic analysis."

Implicit versus explicit dynamics

The direct-integration dynamic procedure provided in Abaqus/Standard offers a choice of implicit operators for integration of the equations of motion, while Abaqus/Explicit uses the central-difference operator. In an implicit dynamic analysis the integration operator matrix must be inverted and a set of nonlinear equilibrium equations must be solved at each time increment. In an explicit dynamic analysis displacements and velocities are calculated in terms of quantities that are known at the beginning of an increment; therefore, the global mass and stiffness matrices need not be formed and inverted, which means that each increment is relatively inexpensive compared to the increments in an implicit integration scheme. The size of the time increment in an explicit dynamic analysis is limited, however, because the central-difference operator is only conditionally stable; whereas the implicit operator options available in Abaqus/Standard are unconditionally stable and, thus, there is no such limit on the size of the time increment that can be used for most analyses in Abaqus/Standard (accuracy governs the time increment in Abaqus/Standard).

The stability limit for the central-difference method (the largest time increment that can be taken without the method generating large, rapidly growing errors) is closely related to the time required for a

stress wave to cross the smallest element dimension in the model; thus, the time increment in an explicit dynamic analysis can be very short if the mesh contains small elements or if the stress wave speed in the material is very high. The method is, therefore, computationally attractive for problems in which the total dynamic response time that must be modeled is only a few orders of magnitude longer than this stability limit; for example, wave propagation studies or some "event and response" applications. Many of the advantages of the explicit procedure also apply to slower (quasi-static) processes for cases in which it is appropriate to use mass scaling to reduce the wave speed (see "Mass scaling," Section 11.6.1).

Abaqus/Explicit offers fewer element types than Abaqus/Standard. For example, only first-order, displacement method elements (4-node quadrilaterals, 8-node bricks, etc.) and modified second-order elements are used, and each degree of freedom in the model must have mass or rotary inertia associated with it. However, the method provided in Abaqus/Explicit has some important advantages:

1. The analysis cost rises only linearly with problem size, whereas the cost of solving the nonlinear equations associated with implicit integration rises more rapidly than linearly with problem size. Therefore, Abaqus/Explicit is attractive for very large problems.
2. The explicit integration method is often more efficient than the implicit integration method for solving extremely discontinuous short-term events or processes.
3. Problems involving stress wave propagation can be far more efficient computationally in Abaqus/Explicit than in Abaqus/Standard.

In choosing an approach to a nonlinear dynamic problem you must consider the length of time for which the response is sought compared to the stability limit of the explicit method; the size of the problem; and the restriction of the explicit method to first-order, pure displacement method or modified second-order elements. In some cases the choice is obvious, but in many problems of practical interest the choice depends on details of the specific case. Experience is then the only useful guide.

Direct-solution versus modal superposition procedures

Direct solution procedures must be used for dynamic analyses that involve a nonlinear response. Modal superposition procedures are a cost-effective option for performing linear or mildly nonlinear dynamic analyses.

Direct-solution dynamic analysis procedures

The following direct-solution dynamic analyses procedures are available in Abaqus:

- **Implicit dynamic analysis:** Implicit direct-integration dynamic analysis ("Implicit dynamic analysis using direct integration," Section 6.3.2) is used to study (strongly) nonlinear transient dynamic response in Abaqus/Standard.
- **Subspace-based explicit dynamic analysis:** The subspace projection method in Abaqus/Standard uses direct, explicit integration of the dynamic equations of equilibrium written in terms of a vector space spanned by a number of eigenvectors ("Implicit dynamic analysis using direct integration," Section 6.3.2). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. This method can be very effective for systems

with mild nonlinearities that do not substantially change the mode shapes. It cannot be used in contact analyses.

- **Explicit dynamic analysis:** Explicit direct-integration dynamic analysis ("Explicit dynamic analysis," Section 6.3.3) is available in Abaqus/Explicit.
- **Direct-solution steady-state harmonic response analysis:** The steady-state harmonic response of a system can be calculated in Abaqus/Standard directly in terms of the physical degrees of freedom of the model ("Direct-solution steady-state dynamic analysis," Section 6.3.4). The solution is given as in-phase (real) and out-of-phase (imaginary) components of the solution variables (displacement, stress, etc.) as functions of frequency. The main advantage of this method is that frequency-dependent effects (such as frequency-dependent damping) can be modeled. The direct method is the most accurate but also the most expensive steady-state harmonic response procedure. The direct method can also be used if nonsymmetric terms in the stiffness are important or if model parameters depend on frequency.

Modal superposition procedures

Abaqus includes a full range of modal superposition procedures. Modal superposition procedures can be run using a high-performance linear dynamics software architecture called SIM. The SIM architecture offers advantages over the traditional linear dynamics architecture for some large-scale analyses, as discussed below in "Using the SIM architecture for modal superposition dynamic analyses."

Prior to any modal superposition procedure, the natural frequencies of a system must be extracted using the eigenvalue analysis procedure ("Natural frequency extraction," Section 6.3.5). Frequency extraction can be performed using the SIM architecture.

The following modal superposition procedures are available in Abaqus:

- **Mode-based steady-state harmonic response analysis:** A steady-state dynamic analysis based on the natural modes of the system can be used to calculate a system's linearized response to harmonic excitation ("Mode-based steady-state dynamic analysis," Section 6.3.8). This mode-based method is typically less expensive than the direct method. The solution is given as in-phase (real) and out-of-phase (imaginary) components of the solution variables (displacement, stress, etc.) as functions of frequency. Mode-based steady-state harmonic analysis can be performed using the SIM architecture.
- **Subspace-based steady-state harmonic response analysis:** In this type of Abaqus/Standard analysis the steady-state dynamic equations are written in terms of a vector space spanned by a number of eigenvectors ("Subspace-based steady-state dynamic analysis," Section 6.3.9). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. The method is attractive because it allows frequency-dependent effects to be modeled and is much cheaper than the direct analysis method ("Direct-solution steady-state dynamic analysis," Section 6.3.4). Subspace-based steady-state harmonic response analysis can be used if the stiffness is nonsymmetric and can be performed using the SIM architecture.
- **Mode-based transient response analysis:** The modal dynamic procedure ("Transient modal dynamic analysis," Section 6.3.7) provides transient response for linear problems using modal superposition. Mode-based transient analysis can be performed using the SIM architecture.

- **Response spectrum analysis:** A linear response spectrum analysis ("Response spectrum analysis," Section 6.3.10) is often used to obtain an approximate upper bound of the peak significant response of a system to a user-supplied input spectrum (such as earthquake data) as a function of frequency. The method has a very low computational cost and provides useful information about the spectral behavior of a system. Response spectrum analysis can be performed using the SIM architecture.
- **Random response analysis:** The linearized response of a model to random excitation can be calculated based on the natural modes of the system ("Random response analysis," Section 6.3.11). This procedure is used when the structure is excited continuously and the loading can be expressed statistically in terms of a "Power Spectral Density" (PSD) function. The response is calculated in terms of statistical quantities such as the mean value and the standard deviation of nodal and element variables. Random response analysis can be performed using the SIM architecture.
- **Complex eigenvalue extraction:** The complex eigenvalue extraction procedure performs eigenvalue extraction to calculate the complex eigenvalues and the corresponding complex mode shapes of a system ("Complex eigenvalue extraction," Section 6.3.6). The eigenmodes of the system extracted in a frequency extraction step are used as the global basis vectors. The complex eigenvalue extraction can be performed using the SIM architecture.

Using the SIM architecture for modal superposition dynamic analyses

SIM is a high-performance software architecture available in Abaqus that can be used to perform modal superposition dynamic analyses. The SIM architecture is much more efficient than the traditional architecture for large-scale linear dynamic analyses (both model size and number of modes) with minimal output requests.

SIM-based analyses can be used to efficiently handle nondiagonal damping generated from element or material contributions, as discussed below in "Damping in a mode-based steady-state and transient linear dynamic analysis using the SIM architecture." Therefore, SIM-based procedures are an efficient alternative to subspace-based linear dynamic procedures for models with element damping or frequency-independent materials.

Activating the SIM architecture

To use the SIM architecture for a modal superposition dynamic analysis, activate SIM for the initial frequency extraction procedure. SIM-based frequency extraction procedures write the mode shapes and other modal system information to a special linear dynamics data (`.sim`) file. By default, this data file is written to the scratch directory and deleted upon job completion; however, if restart is requested, the file is saved in the user directory. All subsequent mode-based steady-state or transient dynamic steps in an analysis automatically use this linear dynamics data file (and by extension the SIM architecture). If you restart an analysis that uses the SIM architecture, you must include the linear dynamics data file.

For more information about frequency extraction procedures, see "Natural frequency extraction," Section 6.3.5.

Input File Usage: *FREQUENCY, SIM

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Use SIM-based linear dynamics procedures**

Example

The SIM architecture will be used for the entire linear dynamic analysis in the following input file template:

```
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS, SIM
Data line to control eigenvalue extraction
*COMPOSITE MODAL DAMPING
Data lines to define fraction of critical damping
*END STEP
**
*STEP
*MODAL DYNAMIC
Data line to control time incrementation
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING, VISCOUS=COMPOSITE
Data lines to define composite modal damping
*END STEP
**
*STEP
*STEADY STATE DYNAMICS
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*END STEP
```

Output in a SIM-based analysis

Output is a fundamental factor in the performance of a linear dynamic analysis. Since it is difficult to predict the desired output quantities for a linear dynamic analysis, preselected output requests are ignored in SIM-based modal superposition procedures (except complex eigenvalue extraction). You must always specify output requests to the output database (`.odb`) file; otherwise, the analysis will not be performed.

There are several restrictions on available output requests that apply specifically to SIM-based analyses:

- You cannot request output to the results (`.fil`) file.
- Element variables cannot be output to the printed data (`.dat`) file except for random response analysis.

Limitations of the SIM architecture

The cyclic symmetry modeling feature cannot be used in SIM-based analyses.

Nonphysical material properties in dynamic analyses

Abaqus relies on user-supplied model data and assumes that the material's physical properties reflect experimental results. Examples of meaningful material properties are a positive mass density per volume, a positive Young's modulus, and a positive value for any available damping coefficients. However, in special cases you may want to "adjust" a value of density, mass, stiffness, or damping in a region or a part of the model to bring the overall mass, stiffness, or damping to the expected required levels. Certain material options in Abaqus allow you to introduce nonphysical material properties to achieve this adjustment.

For example, to adjust the mass of the model, you can define a nonstructural mass with a negative mass value, use mass elements with a negative mass over a region of nodes, or introduce additional elements with negative density. Similarly, to adjust damping levels, you can use negative damping coefficients or introduce dashpot elements with a negative dashpot constant to reduce the overall damping levels. Springs with negative stiffness can be defined to adjust the model stiffness.

If you specify nonphysical but allowed material properties, Abaqus issues a warning message. However, if you specify nonphysical material properties that are not allowed, Abaqus issues an error message. When introducing nonphysical material properties, you must be aware that the overall behavior should be "physical"; for example, the mass values at all nodes must be positive in an eigenvalue extraction procedure.

There are consequences of using nonphysical material properties that are easy to check and interpret, and there are others beyond the control of Abaqus. Therefore, you should fully understand the stated problem and the consequences of using nonphysical material properties before you specify the properties. This is particularly important in Abaqus/Explicit analyses, where the size of the time increment depends on material properties. For example, distributed mass-dependent loads are calculated based on the overall mass density (positive and negative) provided.

Damping in dynamic analysis

Every nonconservative system exhibits some energy loss that is attributed to material nonlinearity, internal material friction, or to external (mostly joint) frictional behavior. Conventional engineering materials like steel and high strength aluminum alloys provide small amounts of internal material damping, not enough to prevent large amplification at or near resonant frequencies. Damping properties increase in modern composite fiber-reinforced materials, where the energy loss occurs through plastic or viscoelastic phenomena as well as from friction at the interfaces between the matrix and reinforcement.

Still larger material damping is exhibited by thermoplastics. Mechanical dampers may be added to models to introduce damping forces to the system. In general, it is difficult to quantify the source of a system's damping. It usually comes from several sources simultaneously; e.g., from energy loss during hysteretic loading, viscoelastic material properties, and external joint friction.

Users that work with a specific system know the source of the energy loss from experience. A variety of methods are available in Abaqus to specify damping that accurately models the energy loss in a dynamic system.

Sources of damping

Abaqus has four categories of damping sources: material and element damping, global damping, modal damping, and damping associated with time integration. If necessary, you can include multiple damping sources and combine different damping sources in a model.

Material and element damping

Damping may be specified as part of a material definition that is assigned to a model (see "Material damping," Section 26.1.1). In addition, Abaqus has elements such as dashpots, springs with their complex stiffness matrix, and connectors that serve as dampers, all with viscous and structural damping factors. Viscous damping can be included in mass, beam, pipe, and shell elements with general section properties; and it can also be used in substructure elements (see "Defining substructures," Section 10.1.2). In direct steady-state dynamic analysis you can define the viscous and structural damping due to the interaction between the contacting surfaces by using user subroutine **UINTER** (see "UINTER," Section 1.1.42 of the Abaqus User Subroutines Reference Guide). Contact damping is not applicable for linear perturbation procedures.

In acoustic elements, velocity proportional viscous damping is implemented using the volumetric drag parameter (see "Acoustic medium," Section 26.3.1). Acoustic infinite elements and impedance conditions also add damping to a model.

Global damping

In situations where material or element damping is not appropriate or sufficient, you can apply abstract damping factors to an entire model. Abaqus allows you to specify global damping factors for both viscous (Rayleigh damping) and structural damping (imaginary stiffness matrix).

Modal damping

Modal damping applies only to mode-based linear dynamic analyses. This technique allows you to apply damping directly to the modes of the system. By definition, modal damping contributes only diagonal entries to the modal system of equations and can be defined several different ways.

Damping associated with time integration

Marching through a simulation with a finite time increment size causes some damping. This type of damping applies only to analyses using direct time integration. See "Implicit dynamic analysis using direct integration," Section 6.3.2, for further discussion of this source of damping.

Damping in a linear dynamic analysis

Damping can be applied to a linear dynamic system in two forms:

- velocity proportional viscous damping; and
- displacement proportional structural damping, which is for use in frequency domain dynamics. The exception is SIM-based transient modal dynamic analysis, where the structural damping is converted to the equivalent diagonal viscous damping (see "Modal dynamic analysis," Section 2.5.5 of the Abaqus Theory Guide).

An additional type of damping known as composite damping serves as a means to calculate a model average critical damping with the material density as the weight factor and is intended for use in mode-based dynamics (excluding subspace projection steady-state analysis and SIM-based dynamic analyses). For additional information, see "Damping options for modal dynamics," Section 2.5.4 of the Abaqus Theory Guide.

The types of damping available for linear dynamic analyses depend on the procedure type and the architecture (traditional or SIM) used to perform the analysis, as outlined in Table 6.3.1–1 and Table 6.3.1–2. For completeness, Table 6.3.1–1 also includes the damping options for a direct steady-state dynamic analysis. In addition to directly specified modal damping, global damping can be used in all linear dynamic procedures. Material and element damping can be used in subspace-based and SIM-based linear dynamic procedures.

Table 6.3.1–1 Damping sources for traditional architecture.

Traditional Architecture	Damping Source		
	Modal	Global	Material and Element
Mode-based steady-state dynamics	√	√	
Subspace-based steady-state dynamics		√	√
Transient modal dynamics	√	√	
Random response analysis	√	√	
Complex frequency		√	√
Response spectrum	√	√	
Direct steady-state dynamics		√	√

Table 6.3.1–2 Damping sources for SIM architecture.

SIM Architecture	Damping Source		
	Modal	Global	Material and Element
Mode-based steady-state dynamics	√	√	√
Subspace-based steady-state dynamics	√	√	√
Transient modal dynamics	√	√	√
Random response analysis	√	√	
Complex frequency	√	√	√
Response spectrum	√	√	

In a subspace-based or SIM-based linear dynamic analysis, material and element damping operators must first be projected onto the basis of mode shapes. This projection results in a full modal damping matrix for both viscous and structural damping; therefore, a modal steady-state response analysis requires the solution of a system of linear equations at each frequency point. The size of this system is equal to the number of modes used in the response calculation. In a mode-based transient analysis, the projected damping operator is treated explicitly in time by including it on the right-hand side of the system of equations.

Frequency-dependent damping is supported only for the subspace-based and direct-integration steady-state dynamic procedures.

Material and element damping is not supported for the response spectrum or the random response procedures. In these procedures, only modal and global damping are allowed, and material or element damping is ignored.

Damping in a mode-based steady-state and transient linear dynamic analysis using the SIM architecture

SIM-based linear dynamic analyses may include material and element damping contributions that introduce both diagonal and nondiagonal terms in the modal system of equations. The projection of material and element damping operators onto the basis of mode shapes is performed during the natural frequency extraction procedure, which enables a high-performance projection operation to be performed when used with the AMS eigensolver. If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

When the structural and viscous damping operators are projected onto the mode shapes, the full modal damping matrix is stored in the linear dynamics data (**`.sim`**) file. The full modal damping matrix is combined with any diagonal contributions from global damping or traditional modal damping. The combined damping operator matrix is included in subsequent mode-based transient or steady-state dynamics steps. If there are nondiagonal (i.e., projected) damping contributions and a large number of modes are included, performance of the linear dynamics calculations will be impacted since a direct solve must be performed at each frequency point.

Acoustic damping due to impedance conditions is projected onto the subspace of acoustic eigenvectors. These contributions are taken into account in a subspace-based steady-state dynamics analysis that uses the SIM architecture.

The default behavior for a SIM-based frequency extraction step is to project any element and material damping onto the mode shapes. You can turn off this damping projection if it is not desired; however, in this case only diagonal damping is available for subsequent modal superposition steps. If the projected damping matrices are not desired in a particular mode-based linear dynamic step for performance reasons, they can be deactivated in that step using the damping control techniques discussed above in "Damping in dynamic analysis."

Input File Usage: Use the following option to project material and element damping operators in a SIM-based analysis:

*FREQUENCY, SIM, DAMPING PROJECTION=ON (default)

Use the following option to turn off damping projection in a SIM-based analysis:

*FREQUENCY, SIM, DAMPING PROJECTION=OFF

Abaqus/CAE Usage: To control the projection of element and material damping in a SIM-based frequency extraction step that uses the Lanczos eigensolver:

Step module: **Step→Create**: **Frequency**: **Eigensolver: Lanczos**, **Use SIM-based linear dynamics procedures**, toggle **Project damping operators**

To control the projection of element and material damping in a frequency extraction step that uses the AMS eigensolver:

Step module: **Step→Create**: **Frequency**: **Eigensolver: AMS**, toggle **Project damping operators**

Defining viscous damping

Abaqus allows you to choose a particular source of viscous damping, to add several sources, or to exclude viscous damping effects.

Defining material/element viscous damping

You can choose to model the viscous damping matrix, $D^{el}_{viscous}$, by using material damping properties and/or damping elements (such as dashpot or mass elements). The viscous, mass, and/or stiffness proportional damping matrix will include the material Rayleigh damping factors, α^{mat}_R and β^{mat}_R, as well as the element-oriented damping factor, α^{el}_R (e.g., for mass elements). The material/element-based viscous damping matrix can be written as

$$D^{el}_{viscous} = \sum_{el=1}^{Num\,elems} \int_V \boldsymbol{\alpha}^{mat}_R N^T N \rho dv + \sum_{el=1}^{Num\,elems} \boldsymbol{\alpha}^{el}_R m^{el} + \sum_{el=1}^{Num\,elems} \int_V \boldsymbol{\beta}^{mat}_R B^T D B dv + \sum_{el=1}^{Num\,elems} d^{el}_{viscous},$$

where $d^{el}_{viscous}$ represents the viscous damping matrix for elements such as dashpots. In mode-based procedures projection of $D^{el}_{viscous}$ into the eigenmodes results in a non-diagonal matrix.

Input File Usage: Use the following option to specify material viscous damping for elements with mechanical degrees of freedom:

∗DAMPING, ALPHA=α^{mat}_R, BETA=β^{mat}_R

Use the following option to specify material viscous damping for acoustic elements:

∗ACOUSTIC MEDIUM, VOLUMETRIC DRAG

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damping**:
Alpha: α^{mat}_R or **Beta**: β^{mat}_R

Property module: material editor: **Other→Acoustic Medium**:
Volumetric Drag

Defining global viscous damping

You can supply global mass and stiffness proportional viscous damping factors, α_{global} and β_{global}, respectively, to create the global damping matrix using the global model mass and stiffness matrices, M and K, respectively:

$$D^{g}_{viscous} = \boldsymbol{\alpha}_{global} M + \boldsymbol{\beta}_{global} K.$$

These parameters can be specified for the entire model (default), for the mechanical degree of freedom field (displacements and rotations) only, or for the acoustic field only.

Input File Usage: Use the following option to specify global viscous damping:

∗GLOBAL DAMPING, ALPHA=α_{global}, BETA=β_{global}

Abaqus/CAE Usage: Global viscous damping is not supported in Abaqus/CAE.

Defining viscous modal damping

Rayleigh damping introduces a damping matrix, $[C]$, defined as

$$[C] = \alpha\,[M] + \beta\,[K]\,,$$

where $[M]$ is the mass matrix of the model, $[K]$ is the stiffness matrix of the model, and α and β are factors that you define.

In Abaqus/Standard you can define α and β independently for each mode, so that the above equation becomes

$$c_M = \alpha_M m_M + \beta_M k_M \quad \text{(no sum on M)},$$

where the subscript M refers to the mode number and c_M, m_M, and k_M are the damping, mass, and stiffness terms associated with the Mth mode.

Input File Usage: Use the following option to define Rayleigh damping by specifying mode numbers:

*MODAL DAMPING, VISCOUS=RAYLEIGH,
DEFINITION=MODE NUMBERS

Use the following option to define Rayleigh damping by specifying a frequency range:

*MODAL DAMPING, VISCOUS=RAYLEIGH,
DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define Rayleigh damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Rayleigh**: **Use Rayleigh damping data**

Use the following input to define Rayleigh damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Rayleigh**: **Use Rayleigh damping data**

Defining viscous modal damping as a fraction of the critical damping

You can also specify the damping in each eigenmode in the model or for the specified frequency as a fraction of the critical damping. Critical damping is defined as

$$c_{cr} = 2\sqrt{mk},$$

where m is the mass of the system and k is the stiffness of the system. Typical values of the fraction of critical damping, ξ_i, are from 1% to 10% of critical damping, c_{cr}; but Abaqus/Standard accepts any positive value. The critical damping factors can be changed from step to step.

Input File Usage: Use the following option to define the fraction of critical damping by specifying mode numbers:

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING,
DEFINITION=MODE NUMBERS

Use the following option to define the fraction of critical damping by specifying a frequency range:

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING,
DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define the fraction of critical damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Direct modal**: **Use direct damping data**

Use the following input to define the fraction of critical damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Direct modal**: **Use direct damping data**

Viscous modal damping for uncoupled structural-acoustic frequency extractions

For uncoupled structural-acoustic frequency extractions performed using the AMS eigensolver, you can apply different damping to the structural and acoustic modes. This technique can be used only when damping is specified for a range of frequencies.

Input File Usage: Use the following option to apply the specified damping to only the structural modes:

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=MECHANICAL

Use the following option to apply the specified damping to only the acoustic modes:

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=ACOUSTIC

Use the following option to apply the specified damping to both structural and acoustic modes (default):

*MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE, FIELD=ALL

Abaqus/CAE Usage: The ability to specify different damping for structural and acoustic modes is not supported in Abaqus/CAE.

Controlling the sources of viscous damping

The material/element and global viscous damping sources can be controlled at the step level; controls are not available for modal damping. If both the material/element and global viscous damping matrices are supplied, both will be used as a combined damping matrix unless you request that only the element or global damping factor be used. The combined material/element and global viscous damping is

$$D_{viscous} = D^{el}_{viscous} + D^{g}_{viscous}.$$

Input File Usage: Use the following option to activate only the material/element viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=ELEMENT

Use the following option to activate only the global viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=FACTOR

Use the following option to activate the combined material/element and global viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=COMBINED

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Excluding viscous damping effects

You can choose to exclude the effects of viscous damping altogether at the step level.

Input File Usage: Use the following option to exclude the viscous damping matrix:

*DAMPING CONTROLS, VISCOUS=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Defining structural damping

Abaqus allows you to choose a particular source of structural damping, to add several sources, or to exclude structural damping effects.

Defining material/element structural damping

The material/element structural damping matrix (that represents the imaginary stiffness and is proportional to forces or displacements) is defined as

$$K_s^m = \sum_{el=1}^{Num\,elems} \int_V \mathbf{s} B^T D B dv + \sum_{el=1}^{Num\,elems} \mathbf{s}^{el} k^{el},$$

where $\mathbf{s}$ represents the material structural damping, $\mathbf{s}^{el}$ represents the structural damping coefficient for elements such as springs with complex stiffnesses and connectors, and k^{el} is the real element stiffness matrix. In mode-based procedures the projection of K_s^m onto the mode shapes results in a full modal damping matrix. When using SIM-based modal procedures, the projected material and element damping matrix may be combined with global and modal damping (see "Defining and using both global and modal diagonal damping," below). Material/element structural damping is not available for acoustic elements.

Input File Usage: Use the following option to specify material structural damping:

*DAMPING, STRUCTURAL=$\mathbf{s}$

Abaqus/CAE Usage: Property module: material editor: **Mechanical**→**Damping**: **Structural**: $\mathbf{s}$

Defining global structural damping

You can define the global structural damping factor, $\mathbf{s}_{global}$, to get

$$K_s^g = \mathbf{s}_{global} K.$$

Global structural damping can be specified for the entire model (default), for the mechanical degree of freedom field (displacements and rotations) only, or for the acoustic field only.

Input File Usage: Use the following option to specify global structural damping:

*GLOBAL DAMPING, STRUCTURAL=$\mathbf{s}_{global}$

Abaqus/CAE Usage: Global structural damping is not supported in Abaqus/CAE.

Defining structural modal damping

Structural damping assumes that the damping forces are proportional to the forces caused by stressing of the structure and are opposed to the velocity (see "Structural damping" in "Material damping," Section 26.1.1, for more information). This form of damping can be used only when the displacement and velocity are exactly 90° out of phase, as in steady-state and random response analyses where the excitation is purely sinusoidal.

Structural damping can be defined as diagonal modal damping for mode-based steady-state dynamic and random response analyses.

Input File Usage: Use the following option to define structural damping by specifying mode numbers:

*MODAL DAMPING, STRUCTURAL, DEFINITION=MODE NUMBERS

Use the following option to define structural damping by specifying a frequency range:

*MODAL DAMPING, STRUCTURAL,
DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define structural damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Modes**, **Structural**: **Use structural damping data**

Use the following input to define structural damping by specifying frequency ranges:

Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Specify damping over ranges of: Frequencies**, **Structural**: **Use structural damping data**

Controlling the sources of structural damping

The material/element and global structural damping sources can be controlled at the step level; controls are not available for modal damping. If both the material/element and global structural damping matrices are supplied, both will be combined unless you request that only the element or global damping factor be used. The combined structural damping matrix is

$$K_s = K_s^m + K_s^g.$$

Input File Usage: Use the following option to activate only the material/element structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=ELEMENT

Use the following option to activate only the global structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=FACTOR

Use the following option to activate the combined material/element and global structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=COMBINED

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Excluding structural damping effects

You can choose to exclude the effects of structural damping altogether at the step level.

Input File Usage: Use the following option to exclude structural damping matrix:

*DAMPING CONTROLS, STRUCTURAL=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Defining both viscous and structural damping

The imaginary contribution to the frequency domain dynamics equation, which represents the effect of damping, may include both viscous and structural damping and can be written as

$$D = \Omega D_{viscous} + K_s,$$

where Ω is the forcing frequency.

Defining composite modal damping

Composite modal damping allows you to define a damping factor for each material or element in the model as a fraction of critical damping. These factors are then combined into a damping factor for each mode as weighted averages of the mass matrix associated with each material or element; when using the SIM architecture, you can also include the weighted averages of the stiffness matrix. Composite modal damping can be defined only by specifying mode numbers; it cannot be defined by specifying a frequency range.

Defining composite modal damping for analyses using the traditional architecture

You specify composite modal damping in the material definition for analyses that use the traditional architecture. The damping per eigenmode is calculated as:

$$\xi_\alpha = \frac{1}{m_\alpha}\phi_\alpha^T \left(\sum_m \xi_m M_m \right) \phi_\alpha,$$

where ξ_α is the critical damping fraction used in mode α, ξ_m is the critical damping fraction defined for material m, M_m is the mass matrix associated with material m, ϕ_α is the eigenvector of mode α, and m_α is the generalized mass associated with mode α:

$$m_\alpha = \phi_\alpha^T M \phi_\alpha.$$

If you specify composite modal damping, Abaqus calculates the damping coefficients ξ_α in the eigenfrequency extraction step from the damping factors ξ_m that you defined for each material.

Input File Usage: Use both of the following options:
*DAMPING, COMPOSITE=ξ_m
*MODAL DAMPING, VISCOUS=COMPOSITE

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damping**: **Composite:** ξ_m
Step module: **Create Step**: **Linear perturbation**: *any valid step type*: **Damping**: **Composite modal**: **Use composite damping data**

Defining composite modal damping for analyses using the SIM architecture

You can specify composite modal damping for SIM-based analyses that use the Lanczos eigensolver. Composite modal damping is specified for each element. You can also assign critical damping fractions to both mass and stiffness matrix input. The mass weighted damping per eigenmode is calculated as:

$$\xi_\alpha^M = \phi_\alpha^T M_\xi \phi_\alpha = \phi_\alpha^T \left(\sum_{elements} \xi_{element}^M m_{element} + \xi_{\substack{matrix\\input}}^M M_{\substack{matrix\\input}} \right) \phi_\alpha,$$

where ξ_α^M is the mass weighted critical damping fraction used in mode α, M_ξ is a damped portion of the mass matrix, ξ^M are fractions of critical damping for the element mass matrix and mass matrix input, and ϕ_α is the eigenvector of mode α.

The stiffness weighted damping per eigenmode is calculated as:

$$\xi_\alpha^K = \frac{1}{\omega_\alpha^2}\phi_\alpha^T K_\xi \phi_\alpha = \frac{1}{\omega_\alpha^2}\phi_\alpha^T \left(\sum_{elements} \xi_{element}^K k_{element} + \xi_{\substack{matrix\\input}}^K K_{\substack{matrix\\input}} \right) \phi_\alpha,$$

where ξ_α^K is the stiffness weighted critical damping fraction used in mode α, K_ξ is a damped portion of the stiffness matrix, ξ^K are fractions of critical damping for the element stiffness and matrix input stiffness, and ϕ_α is the eigenvector of mode α.

Input File Usage: Use both of the following options to specify composite modal damping:

*FREQUENCY, EIGENSOLVER=LANCZOS, SIM
*COMPOSITE MODAL DAMPING

Use the following option to specify the fraction of critical damping for all mass matrices included in matrix input:

*COMPOSITE MODAL DAMPING, MASS MATRIX INPUT

Use the following option to specify the fraction of critical damping for all stiffness matrices included in matrix input:

*COMPOSITE MODAL DAMPING, STIFFNESS MATRIX INPUT

Abaqus/CAE Usage: Defining composite modal damping for analyses using the SIM architecture is not supported in Abaqus/CAE.

Defining global damping for acoustic fields

If your model contains acoustic elements, Abaqus applies any specified global damping to both the acoustic fields and the structural fields in the model by default. If desired, you can specify that a global damping definition applies only to the acoustic fields or only to the displacement and rotation fields (not supported in a mode-based steady-state dynamic analysis that uses coupled acoustic-structural modes).

Input File Usage: Use the following option to apply global damping to all of the displacement, rotation, and acoustic fields in a model:

*GLOBAL DAMPING, FIELD=ALL (default)

Use the following option to apply global damping only to the acoustic fields in a model:

*GLOBAL DAMPING, FIELD=ACOUSTIC

Use the following option to apply global damping only to the displacement and rotation fields in a model:

*GLOBAL DAMPING, FIELD=MECHANICAL

Abaqus/CAE Usage: Global damping is not supported in Abaqus/CAE.

Defining and using both global and modal diagonal damping

Mode-based procedures—such as steady-state dynamics, transient modal dynamic, response spectrum, and random response analyses—can also use a step-dependent, modal damping definition that is specified per eigenmode. When multiple modal damping definitions are used with different damping types, the damping is additive. If the same damping type is specified more than once, the last specification is used. If modal damping is used with global damping, both types of damping will contribute to the damping matrix.

Damping controls have no effect on modal damping. If damping controls are used to exclude certain global damping effects in a step, all modal damping effects are still included in the step. To exclude modal damping, the damping definition must be specifically removed from the step definition.

Controlling damping of low frequency modes

You can include or exclude damping of the low frequency eigenmodes in transient modal analyses. This control is useful for free structures and models with secondary base motions. For details, see "Transient modal dynamic analysis," Section 6.3.7.

Acoustic contribution factors in mode-based and subspace-based steady-state dynamic analyses

You can compute acoustic contribution factors for the linear, eigenmode-based, steady-state dynamic procedures. Computation of the acoustic contribution factors makes it possible to answer the following questions:

- Which noise source has the greatest contribution?
- Which point does the loudest noise come from?
- Which structural component does the loudest noise come from?
- Which natural frequency contributes the most to the noise?

By answering these questions you can determine the major noise sources as well as make the necessary changes to your design to reduce the noise levels. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. It is available in steady-state mode-based and subspace-based dynamic analyses performed using the high-performance SIM architecture. To enable computation of the acoustic contribution factors, the preceding frequency extraction step has to use the uncoupled modes formulation as well as to activate the SIM architecture. Abaqus/Standard supports the computation of the following contribution factors:

- Acoustic modal contribution factors,
- Acoustic structural modal contribution factors,
- Acoustic load modal contribution factors,
- Acoustic load contribution factors,
- Panel contribution factors, and
- Grid contribution factors.

The acoustic node set defines a set of the response nodes in the acoustic domain. You can specify up to 20 response nodes in this set. You can also select the acoustic or structural eigenmodes that will be used to compute the modal contribution factors. You specify the lower and upper bounds of the frequency range to apply to the active eigenmodes (see "Selecting the modes and specifying damping" in "Mode-based steady-state dynamic analysis," Section 6.3.8, and "Selecting the modes on which to project" in "Subspace-based steady-state dynamic analysis," Section 6.3.9).

The computed contribution factors are saved in the SIM database file. You can retrieve the data as described in "Plug-in utility for visualizing Acoustic Contribution Factors" in the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base.

Input File Usage: Use the following option to request computation of the acoustic modal contribution factors:

∗ACOUSTIC CONTRIBUTION, NAME=*name*,
ACOUSTIC NODES=*acoustic_nset*
f_{min}, f_{max}

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying acoustic modal contribution factors

Acoustic modal contribution factors show the contribution of each acoustic mode into the total acoustic pressure at the response points. You can also select the acoustic eigenmodes that will be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic modal contribution factors:

∗ACOUSTIC CONTRIBUTION, TYPE=MODAL ACOUSTIC

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying acoustic structural modal contribution factors

Acoustic structural modal contribution factors measure the contribution of each structural mode into the acoustic pressure caused by the structural components. You can also select the structural eigenmodes that will be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic structural modal contribution factors:

∗ACOUSTIC CONTRIBUTION, TYPE=MODAL STRUCTURAL

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying acoustic load modal contribution factors

Acoustic load modal contribution factors define the contribution of each acoustic mode due to the acoustic sources into the acoustic pressure. You can specify the acoustic eigenmodes that are going to be used to compute the contribution factors.

Input File Usage: Use the following option to compute the acoustic load modal contribution factors:

∗ACOUSTIC CONTRIBUTION, TYPE=MODAL LOAD

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying acoustic load contribution factors

Acoustic load contribution factors define the contribution of the acoustic sources into the acoustic pressure.

Input File Usage: Use the following option to compute the acoustic load contribution factors:

∗ACOUSTIC CONTRIBUTION, TYPE=LOAD

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying panel contribution factors

Panel contribution factors measure the contribution of the user-defined structural surfaces into the acoustic pressure caused by structural sources.

Optionally, you can specify a set of nodes that defines a structural panel—a set of nodes on the acoustic-structural interface that is considered to be a single noise source. If this node set contains other structural or acoustic nodes that do not belong to the acoustic-structural interface, such nodes are filtered out and are not considered for panel contribution factor computations. If you do not specify a set of nodes on the acoustic-structural interface, all nodes on the interface are used to determine the panel contribution factors.

Input File Usage: Use the following option to compute the panel contribution factor:

```
*ACOUSTIC CONTRIBUTION, TYPE=PANEL,
STRUCTURAL NODES=structural_nset
```

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

Specifying grid contribution factors

Grid contribution factors measure the contribution of each node on the acoustic-structural interface into the acoustic pressure caused by structural sources.

Optionally, you can specify a set of nodes on the acoustic-structural interface. Each node in this set is considered to be an individual noise source. If this node set contains other structural or acoustic nodes that do not belong to the acoustic-structural interface, such nodes will be filtered out and will not be considered for the grid contribution factor computations. If you do not specify a set of nodes on the acoustic-structural interface, all nodes on the interface are used to determine the grid contribution factors. Due to the large amount of data generated for grid contribution factors, the number of nodes in this node set is limited to 10,000 nodes. If this number is exceeded, the first 10,000 nodes are used.

Input File Usage: Use the following option to compute the grid contribution factor:

```
*ACOUSTIC CONTRIBUTION, TYPE=GRID,
STRUCTURAL NODES=structural_nset
```

Abaqus/CAE Usage: Specifying acoustic contribution factors is not supported in Abaqus/CAE.

6.3.2 IMPLICIT DYNAMIC ANALYSIS USING DIRECT INTEGRATION

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Dynamic analysis procedures: overview," Section 6.3.1
- *DYNAMIC
- "Configuring a dynamic, implicit procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A direct-integration dynamic analysis in Abaqus/Standard:

- must be used when nonlinear dynamic response is being studied;
- can be fully nonlinear (general dynamic analysis) or can be based on the modes of the linear system (subspace projection method); and
- can be used to study a variety of applications, including:
  - dynamic responses requiring transient fidelity and involving minimal energy dissipation;
  - dynamic responses involving nonlinearity, contact, and moderate energy dissipation; and
  - quasi-static responses in which considerable energy dissipation provides stability and improved convergence behavior for determining an essentially static solution.

General dynamic analysis

General nonlinear dynamic analysis in Abaqus/Standard uses implicit time integration to calculate the transient dynamic or quasi-static response of a system. The procedure can be applied to a broad range of applications calling for varying numerical solution strategies, such as the amount of numerical damping required to obtain convergence and the way in which the automatic time incrementation algorithm proceeds through the solution. Typical dynamic applications fall into three categories:

- Transient fidelity applications, such as an analysis of satellite systems, require minimal energy dissipation. In these applications small time increments are taken to accurately resolve the vibrational response of the structure, and numerical energy dissipation is kept at a minimum. These stringent requirements tend to degrade convergence behavior for simulations involving contact or nonlinearities.
- Moderate dissipation applications encompass a more general range of dynamic events in which a moderate amount of energy is dissipated by plasticity, viscous damping, or other effects. Typical applications include various insertion, impact, and forming analyses. The response of these structures can be either monotonic or nonmonotonic. Accurate resolution of high-frequency

vibrations is usually not of interest in these applications. Some numerical energy dissipation tends to reduce solution noise and improve convergence behavior in these applications without significantly degrading solution accuracy.

- Quasi-static applications are primarily interested in determining a final static response. These problems typically show monotonic behavior, and inertia effects are introduced primarily to regularize unstable behavior. For example, the statically unstable behavior may be due to temporarily unconstrained rigid body modes or "snap-through" phenomena. Large time increments are taken when possible to obtain the final solution at minimal computational cost. Considerable numerical dissipation may be required to obtain convergence during certain stages of the loading history.

An example of a transient fidelity application is available in "Modeling of an automobile suspension," Section 2.1.7 of the Abaqus Example Problems Guide. An analysis that includes both a moderate dissipation step and a quasi-static step is described in "Impact analysis of a pawl-ratchet device," Section 2.1.17 of the Abaqus Example Problems Guide.

Specifying the application type

Based on the classifications listed above, you should indicate the type of application you are studying when performing a general dynamic analysis. Abaqus/Standard assigns numerical settings based on your classification of the application type, and this classification can significantly affect a simulation. In some cases accurate results can be obtained with more than one application-type setting, in which case analysis efficiency should be considered. A general trend is that—among the three classifications—the high-dissipation quasi-static classification tends to result in the best convergence behavior and the low-dissipation transient fidelity classification tends to have the highest likelihood of convergence difficulty.

Input File Usage: Use the following option for transient fidelity applications:

*DYNAMIC, APPLICATION=TRANSIENT FIDELITY (default for models without contact)

Use the following option for moderate dissipation applications:

*DYNAMIC, APPLICATION=MODERATE DISSIPATION (default for models with contact)

Use the following option for quasi-static applications:

*DYNAMIC, APPLICATION=QUASI-STATIC

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Implicit**

The application type is specified in the **Edit Step** dialog box:

Basic: **Application: Transient fidelity**, **Moderate dissipation**, **Quasi-static**, or **Analysis product default**

Diagnostics for modeling errors associated with mass properties

Accurate representation of inertia properties is necessary for accurate dynamic analyses. In some cases Abaqus/Standard provides diagnostic messages when it detects likely modeling errors associated with the

specification of inertia properties. The most common way of specifying inertia properties is with material densities. Abaqus/Standard issues a warning message to the data (`.dat`) file if a material density is omitted in a dynamic analysis (this warning is not issued if the density is zero only for certain values of temperature or field variables). Other methods of specifying inertia properties include:

- point mass and rotary inertia definitions, and
- constraining nodes without inertia themselves to nodes having inertia properties defined.

In some circumstances Abaqus/Standard attempts to solve systems of equations involving effective inversion of the global mass matrix to directly adjust velocities and accelerations during a general dynamic analysis as described in "Initial conditions" and "Intermittent contact/impact" below. These additional velocity and acceleration adjustments occur by default only for transient fidelity application types as defined above. If the global mass matrix is found to be singular, Abaqus/Standard issues an error message by default, because singular mass is an indication that the mass properties are not realistic due to a modeling error.

Diagnostic feedback specific to the global mass matrix being singular is typically not provided for quasi-static and moderate dissipation application types, although warnings typically are issued regarding the lack of material density. Singular mass is not necessarily detrimental to a quasi-static analysis. For example, it would be reasonable to only define inertia properties (such as density) in components or regions with temporary static instabilities (such as initially unconstrained rigid body modes that become constrained once contact occurs) in a quasi-static analysis.

You can control the course of action Abaqus/Standard takes upon detecting a singular global mass matrix.

Input File Usage: Use the following default option to issue an error message and stop execution if a singular global mass matrix is detected when calculating velocity and acceleration adjustments:

```
*DYNAMIC, SINGULAR MASS=ERROR
```

Use the following option to issue a warning message and avoid velocity and acceleration adjustments (i.e., continue time integration using current velocities and accelerations) if a singular global mass matrix is detected:

```
*DYNAMIC, SINGULAR MASS=WARNING
```

Use the following option to adjust velocities and accelerations even if a singular mass matrix is detected. This setting can result in large, non-physical velocity and/or acceleration adjustments, which can, in turn, cause poor time integration solutions and artificial convergence difficulties. *This approach is not generally recommended; it should be used only in special cases when the analyst has a thorough understanding of how to interpret results obtained in this way.*

```
*DYNAMIC, SINGULAR MASS=MAKE ADJUSTMENTS
```

Abaqus/CAE Usage: The default singular mass setting cannot be modified in Abaqus/CAE.

Numerical details

The effect of the application-type classification on numerical aspects of general dynamic analyses is described below. In most cases the settings determined by the application type are sufficient to successfully perform an analysis. However, detailed user controls are provided to override settings on an individual basis.

Time integration methods

Abaqus/Standard uses the Hilber-Hughes-Taylor time integration by default unless you specify that the application type is quasi-static. The Hilber-Hughes-Taylor operator is an extension of the Newmark β-method. Numerical parameters associated with the Hilber-Hughes-Taylor operator are tuned differently for moderate dissipation and transient fidelity applications (as discussed later in this section). The backward Euler operator is used by default if the application classification is quasi-static.

These time integration operators are implicit, which means that the operator matrix must be inverted and a set of simultaneous nonlinear dynamic equilibrium equations must be solved at each time increment. This solution is done iteratively using Newton's method. The principal advantage of these operators is that they are unconditionally stable for linear systems; there is no mathematical limit on the size of the time increment that can be used to integrate a linear system. An unconditionally stable integration operator is of great value when studying structural systems because a conditionally stable integration operator (such as that used in the explicit method) can lead to impractically small time steps and, therefore, a computationally expensive analysis.

Marching through a simulation with a finite time increment size generally introduces some degree of numerical damping. This damping differs from the material damping discussed in "Material damping," Section 26.1.1 (and in many cases these two forms of damping will work well together). The amount of damping associated with the time integration varies among the operator types (for example, the backward Euler operator tends to be more dissipative than the Hilber-Hughes-Taylor operator) and in many cases (such as with the Hilber-Hughes-Taylor operator) depends on settings of numerical parameters associated with the operator. The ability of the operator to effectively treat contact conditions is often of considerable importance with respect to their usefulness. For example, some changes in contact conditions can result in "negative damping" (nonphysical energy source) for many time integrators, which can be very undesirable.

It is possible to override the time integrator implied by the application-type classification; for example, you can perform a moderate dissipation dynamic analysis using the backward Euler integrator. Changing the default integrator is not generally recommended but may be useful in special cases.

Input File Usage: Use the following option to use the Hilber-Hughes-Taylor integrator with default integrator parameter settings corresponding to those for transient fidelity applications:

```
*DYNAMIC, TIME INTEGRATOR=HHT-TF
```

Use the following option to use the Hilber-Hughes-Taylor integrator with default integrator parameter settings corresponding to those for moderate dissipation applications:

```
*DYNAMIC, TIME INTEGRATOR=HHT-MD
```

Use the following option to use the backward Euler integrator:

*DYNAMIC, TIME INTEGRATOR=BWE

Abaqus/CAE Usage: The default time integrator cannot be modified in Abaqus/CAE.

Additional control over integrator parameters

Additional user controls enable modifications to settings of numerical parameters associated with the Hilber-Hughes-Taylor operator (see Hilber, Hughes, and Taylor (1977) for descriptions of the numerical parameters). The default parameter settings depend on the specified application type, as indicated in Table 6.3.2–1 (see Czekanski, El-Abbasi, and Meguid (2001) for the basis of these settings).

Table 6.3.2–1 Default parameters for the Hilber-Hughes-Taylor integrator.

	Application	
Parameter	**Transient Fidelity**	**Moderate Dissipation**
α	–0.05	–0.41421
β	0.275625	0.5
γ	0.55	0.91421

These parameters can be adjusted or modified individually if the Hilber-Hughes-Taylor operator is being used. If the default settings of these parameters correspond to the transient fidelity settings shown in Table 6.3.2–1 and you explicitly modify the α parameter alone, the other parameters will be adjusted automatically to $\beta = 1/4 \cdot (1 - \alpha)^2$ and $\gamma = 1/2 - \alpha$. This relation provides control of the numerical damping associated with the time integrator while preserving desirable characteristics of the integrator. The numerical damping grows with the ratio of the time increment to the period of vibration of a mode. Negative values of α provide damping; whereas $\alpha = 0$ results in no damping (energy preserving) and is exactly the trapezoidal rule (sometimes called the Newmark β-method, with $\beta = 1/4$ and $\gamma = 1/2$). The setting $\alpha = -1/3$ provides the maximum numerical damping. It gives a damping ratio of about 6% when the time increment is 40% of the period of oscillation of the mode being studied. Allowable values of α, β, and γ are: $-\frac{1}{2} \le \alpha \le 0$, $\beta > 0$, $\gamma \ge \frac{1}{2}$.

Input File Usage: *DYNAMIC, ALPHA=α, BETA=β, GAMMA=γ

Abaqus/CAE Usage: Only the α parameter can be modified in Abaqus/CAE:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Alpha: Specify:** α

Default incrementation schemes

Automatic time incrementation is used by default for nonlinear dynamic procedures. The main factors used to control adjustments to the time increment size for an implicit dynamic procedure are the convergence behavior of the Newton iterations and the accuracy of the time integration. The

time increment size may vary considerably during an analysis. Details of the time increment control algorithm depend on the type of dynamic application you are studying.

The following factors are considered by default in the time increment control algorithm if you specify a quasi-static–type application (the same factors control the time increment size for purely static analyses):

- The time increment size is reduced if an increment appears to be diverging or if the convergence rate is slow.
- The time increment size is fairly aggressively increased if rapid convergence occurs in previous increments.

Analyses for moderate dissipation-type applications also use these same factors, as well as a default upper bound on the time increment size equal to one-tenth of the step duration.

The following factors are considered by default in the time increment control algorithm if you specify a transient fidelity–type application:

- The time increment size is reduced if an increment appears to be diverging or if the convergence rate is slow.
- The time increment size is reduced if changes in contact status are detected during the first attempt of processing an increment. The new increment size is set such that the end of the increment corresponds to the average time of the contact status changes that were detected with the previous increment size. (In such cases an additional very small time increment is used to enforce compatibility of velocities and accelerations across active contact interfaces.)
- The time increment size is reduced if the half-increment residual (out-of-balance force) halfway through a time increment exceeds the half-increment residual tolerance, which is 10,000 times the time average force for a contact analysis or 1000 times the time average force for an analysis without contact.
- The time increment is gradually increased if rapid convergence occurs in previous increments.
- The upper bound for the time increment size is equal to 1/100 of the step duration.

Intermittent contact/impact

The second and third factors described in the preceding list often result in very small time increment sizes for contact simulations that are performed as a transient fidelity application (and the time increment size tends to remain small due to the fourth factor). This problem can be avoided by specifying a different application type or by using more detailed user controls, as discussed below.

General settings for the time increment controls

A high level user control over which factors are considered by the time increment control algorithm can be used to override the defaults implied by the specified application type for the analysis. Regardless of the application type you have specified, you can enforce time increment controls associated with either quasi-static applications or transient fidelity applications.

Input File Usage: Use the following option to obtain the aggressive time increment control settings associated with quasi-static applications:

*DYNAMIC, INCREMENTATION=AGGRESSIVE

Use the following option to obtain the more conservative time increment control settings associated with transient fidelity applications:

*DYNAMIC, INCREMENTATION=CONSERVATIVE

Abaqus/CAE Usage: The default time incrementation control settings cannot be modified in Abaqus/CAE.

Controlling the half-increment residual

Controls associated with the half-increment residual tolerance are provided for tuning the time incrementation. These controls are intended for advanced users and typically do not need to be modified.

Input File Usage: Use the following option to specify that no check of the half-increment residual should be performed:

*DYNAMIC, NOHAF

Use the following option to specify the half-increment residual tolerance as a scale factor of the time average force (moment):

*DYNAMIC, HALFINC SCALE FACTOR=*scale factor*

Use the following option to directly specify the half-increment residual force tolerance (the half-increment residual moment tolerance is the half-increment residual force tolerance times the characteristic element length automatically calculated):

*DYNAMIC, HAFTOL=*tolerance*

Abaqus/CAE Usage: Use the following option to specify that no check of the half-increment residual should be performed:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: toggle on **Suppress half-increment residual calculation**

Use the following option to specify the half-increment residual tolerance as a scale factor of the time average force (moment):

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Half-increment Residual**: **Specify scale factor:** *scale factor*

Use the following option to specify the half-increment residual force tolerance directly:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Half-increment Residual: Specify value:** *tolerance*

Controlling incrementation involving contact

By default, specifying a transient fidelity application typically results in reduced time increment sizes upon changes in contact status. An extra time increment with a very small size is subsequently performed to enforce compatibility of velocities and accelerations across active contact interfaces. Direct user control over these incrementation aspects is available.

Input File Usage: Use the following option to avoid automatically cutting back the increment size and enforcing velocity and acceleration compatibility in the contact region upon changes in contact status:

*DYNAMIC, IMPACT=NO

Use the following option to automatically cut back the increment size and enforce velocity and acceleration compatibility in the contact region upon changes in contact status:

*DYNAMIC, IMPACT=AVERAGE TIME

Use the following option to enforce velocity and acceleration compatibility in the contact region without automatically cutting back the increment size upon changes in contact status:

*DYNAMIC, IMPACT=CURRENT TIME

Abaqus/CAE Usage: The default contact incrementation scheme cannot be modified in Abaqus/CAE.

Direct time incrementation

You may directly specify the time increment size to be used. This approach is not generally recommended but may be useful in special cases. The analysis will terminate if convergence tolerances are not satisfied within the maximum number of iterations allowed.

It is possible to ignore convergence tolerances: the solution to an increment is accepted after the specified maximum number of iterations allowed even if convergence tolerances are not satisfied. Ignoring convergence tolerances can result in highly nonphysical results and is not recommended except by analysts with a thorough understanding of how to interpret results obtained this way.

Input File Usage: Use the following option to directly specify the time increment:

*DYNAMIC, DIRECT

Use the following option to ignore convergence tolerances after the maximum number of iterations is reached:

*DYNAMIC, DIRECT=NO STOP

Abaqus/CAE Usage: Use the following option to specify the time increment directly:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Incrementation**: **Fixed**

Use the following option to ignore convergence tolerances after the maximum number of iterations is reached:

Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Accept solution after reaching maximum number of iterations**

Default amplitude for loads

Loads such as applied forces or pressures are ramped on by default if you have selected the quasi-static application classification; such ramping tends to enhance robustness because the load increment size is proportional to the time increment size. For example, if the Newton iterations are not able to converge for a particular time increment size, the automatic time incrementation algorithm will reduce the time increment size and restart the Newton iterations with a smaller load incremental considered.

For the other application classifications the dynamic procedure applies loads with a step function by default such that the full load is applied in the first increment of the step (regardless of the time increment size) and the load magnitude remains constant over each step. Thus, if the first increment is unable to converge with the original time increment size, reducing the time increment will not reduce the load increment by default. In some cases the convergence behavior will still improve upon reducing the time increment because the regularizing effect of inertia on the integration operators is inversely proportional to the square of the time increment size. See "Defining an analysis," Section 6.1.2, for more information on default amplitude types for the various procedures and how to override the default.

The "subspace projection" method

The alternative approach provided in Abaqus/Standard for nonlinear dynamic problems is the "subspace projection" method. See "Subspace dynamics," Section 2.4.3 of the Abaqus Theory Guide, for the theory behind this method. In this method the modes of the linear system are extracted in an eigenfrequency extraction step ("Natural frequency extraction," Section 6.3.5) prior to the dynamic analysis and are used as a small set of global basis vectors to develop the solution. These modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The method works well when the system exhibits mildly nonlinear behavior, such as small regions of plastic yielding or rotations that are not small but not too large.

This method can be very effective. As with the other direct integration methods, it is more expensive in terms of computer time than the modal methods of purely linear dynamic analysis, but it is often significantly less expensive than the direct integration of all of the equations of motion of the model. However, since the subspace projection method is based on the modes of the system, it will not be accurate if there is extreme nonlinear response that cannot be modeled well by the modes that form the basis of the solution.

Input File Usage: *DYNAMIC, SUBSPACE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Subspace**

Selecting the modes on which to project

You can select the modes of the system on which the subspace projection will be performed. The mode numbers can be listed individually, or they can be generated automatically. If you choose not to select

the modes, all modes extracted in the prior frequency extraction step, including residual modes if they were activated, are used in the subspace projection.

Input File Usage: Use one of the following options:

*SELECT EIGENMODES
*SELECT EIGENMODES, GENERATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Subspace**: **Basic**: **Number of modes to use**: **All** or **Specify**

Numerical implementation

The subspace projection method is implemented in Abaqus/Standard using the explicit (central difference) operator to integrate the equations of motion written in terms of the modes of the linear system. This integration method is particularly effective here because the modes are orthogonal with respect to the mass matrix so that the projected system always has a diagonal mass matrix.

A fixed time increment is used: this increment is the smaller of the time increment that you specify or 80% of the stable time increment, which is $2/\omega_{\max}$ for the linear system, where $\omega_{\max}$ is the highest circular frequency of the modes that are used as the basis of the solution. The 80% factor is intended as a safety factor so that any increase in this highest frequency caused by nonlinear effects is less likely to cause the integration to become unstable. The 80% is rather arbitrary; in some cases it may be nonconservative. You must monitor the response—for example, the energy balance—to ensure that the time increment is not causing instability. Instability is a concern if the nonlinearities can stiffen the system significantly, although in many practical cases such stiffening effects are more prominent in increasing the lower frequencies of the system than in affecting the highest frequencies that are likely to be retained to represent the dynamic behavior accurately.

Accuracy of the subspace projection method

The effectiveness of the subspace projection method depends on the value of the modes of the linear system as a set of global interpolation functions for the problem, which is a matter of judgment on your part—the same sort of judgment as required when deciding if a particular mesh of finite elements is sufficient. The method is valuable for mildly nonlinear systems and for cases where it is easy to extract enough modes that you can be confident that they describe the system adequately.

If nonlinear geometric effects are considered in the subspace dynamics step, it is possible to perform a dynamic simulation for some time, reextract the modes on the current stressed geometry by using another frequency extraction step, and then continue the analysis with the new modes as the subspace basis system. This procedure can improve the accuracy of the method in some cases.

Material damping

You can introduce Rayleigh damping, as explained in "Material damping," Section 26.1.1. This damping will act in addition to numerical damping associated with the time integrator (discussed previously).

Input File Usage: *DAMPING, ALPHA=α_R, BETA=β_R

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damping**: **Alpha** and **Beta**

Initial conditions

"Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions. Initial velocities must be defined in global directions regardless of the use of nodal transformations (see "Transformed coordinate systems," Section 2.1.5).

If initial velocities are specified at nodes for which displacement boundary conditions are also specified, the initial velocities will be ignored at these nodes. However, if a displacement boundary condition refers to an amplitude curve with an analytically defined time variation (i.e., excluding the piecewise linear tabular and equally spaced definitions), Abaqus/Standard will compute the initial velocity for the nodes involved in the boundary condition as the time derivative (evaluated at time zero) of the analytic variation.

When initial velocities are specified for dynamic analysis, they should be consistent with all of the constraints on the model, especially time-dependent boundary conditions. Abaqus/Standard will ensure that initial velocities are consistent with boundary conditions and with multi-point and equation constraints but will not check for consistency with internal constraints such as incompressibility of the material. In case of a conflict, boundary conditions and multi-point constraints take precedence over initial conditions.

Specified initial velocities are used in a dynamic step only if it is the first dynamic step in an analysis. If a dynamic step is not the first dynamic step and there is an immediately preceding dynamic step, the velocities from the end of the preceding step are used as the initial velocities for the current step. If a dynamic step is not the first dynamic step and the immediately preceding step is not a dynamic step, zero initial velocities are assumed for the current step.

Controlling calculation of accelerations at the beginning of a dynamic step

By default, Abaqus/Standard will calculate accelerations at the beginning of the dynamic step for transient fidelity applications. You can choose to bypass these acceleration calculations, in which case Abaqus/Standard will assume that initial accelerations for the current step are zero unless there is an immediately preceding dynamic step. If the immediately preceding step is also a dynamic step, bypassing the acceleration calculations will cause Abaqus/Standard to use the accelerations from the end of the previous step to continue the new step. It is appropriate to bypass the acceleration calculations if the loading has not changed suddenly at the start of the dynamic step, but it is not correct if the loading at the beginning of the first increment is significantly different from that at the end of the previous step. In cases where large loads are applied suddenly, high-frequency noise due to the bypass of the acceleration calculations may greatly increase the half-increment residual.

Input File Usage: *DYNAMIC, INITIAL=NO

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Implicit**: **Other**: **Initial acceleration calculations at beginning of step: Bypass**

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6), to warping degree of freedom 7 in open-section beam elements, to fluid pressure degree of freedom 8 for

hydrostatic fluid elements, or to acoustic pressure degree of freedom 8 for acoustic elements ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Amplitude references can be used to prescribe time-varying boundary conditions in a direct-integration dynamic step. Default amplitude variations are described in "Defining an analysis," Section 6.1.2.

In direct time integration dynamic analysis, when a node with a prescribed motion is used in an equation constraint or a multi-point constraint to control the motion of another node, the equation or multi-point constraint will be imposed correctly for the displacement and velocity of the dependent node. However, the acceleration will not be rigorously transmitted to the dependent node, which may cause some high-frequency noise.

In the subspace projection method it is not currently possible to specify nonzero boundary conditions directly. Instead, acceleration boundary conditions can be approximated by using appropriate combinations of large point masses and concentrated loads. At the node where such a boundary condition is desired, attach a large point mass that is approximatively 10^5–10^6 times larger than the mass of the original model. In addition, a concentrated load of magnitude equal to the product between the large point mass and the desired acceleration must be specified in the direction of the approximated boundary condition. Since the point mass is significantly larger than the mass of the model, the big mass–concentrated load combination will approximate the desired acceleration in the specified direction accurately. Boundary conditions other than accelerations must be converted into acceleration histories before they can be approximated.

Loads

The following loads can be prescribed in a dynamic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."
- Distributed pressure or volumetric accelerations (on acoustic elements) can be applied; these are described in "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1.

Predefined fields

The following predefined fields can be specified in a dynamic analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in stress/displacement elements, nodal temperatures can be specified as a predefined field. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.

Material options

Most material models that describe mechanical behavior are available for use in a dynamic analysis. The following material properties are not active during a dynamic analysis: thermal properties (except for thermal expansion), mass diffusion properties, electrical conductivity properties, and pore fluid flow properties.

Rate-dependent material properties ("Time domain viscoelasticity," Section 22.7.1; "Hysteresis in elastomers," Section 22.8.1; "Rate-dependent yield," Section 23.2.3; and "Two-layer viscoplasticity," Section 23.2.11) can be included in a dynamic analysis.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature, pressure, and electrical potential degrees of freedom) can be used in a dynamic analysis. Inertia effects are ignored in hydrostatic fluid elements, and the inertia of the fluid in pore pressure elements is not taken into account.

Output

In addition to the usual output variables available in Abaqus/Standard (see "Abaqus/Standard output variable identifiers," Section 4.2.1), the following variables are provided specifically for implicit dynamic analysis:

Variables for a specified element set or for the entire model:

XC	Current coordinates of the center of mass.
XC*n*	Coordinate *n* of the center of mass ($n = 1, 2, 3$).
UC	Displacement of the center of mass.
UC*n*	Displacement component *n* of the center of mass ($n = 1, 2, 3$).
URC*n*	Rotation component *n* of the center of mass.
VC	Equivalent rigid body velocity components.
VC*n*	Component *n* of the equivalent rigid body velocity ($n = 1, 2, 3$).
VRC*n*	Component *n* of the equivalent rigid body angular velocity ($n = 1, 2, 3$).
HC	Angular momentum about the center of mass.
HC*n*	Component *n* of the angular momentum about the center of mass ($n = 1, 2, 3$).
HO	Angular momentum about the origin.
HO*n*	Component *n* of the angular momentum about the origin ($n = 1, 2, 3$).
RI	Rotary inertia about the origin.
RI*ij*	ij-component of the rotary inertia about the origin ($i \leq j \leq 3$).
MASS	Mass.
VOL	Current volume.

Input file template

***HEADING**
…
***BOUNDARY**
Data lines to specify zero-valued boundary conditions
***INITIAL CONDITIONS**
Data lines to specify initial conditions
AMPLITUDE, NAME=name*
Data lines to define amplitude variations
******
***STEP (,NLGEOM)**
Once NLGEOM is specified, it will be active in all subsequent steps.
***DYNAMIC**
Data line to control automatic time incrementation

***BOUNDARY**
Data lines to describe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD and/or *INCIDENT WAVE**
Data lines to specify loads
***TEMPERATURE and/or *FIELD**
Data lines to prescribe predefined fields
***CECHARGE and/or *DECHARGE (if electrical potential degrees of freedom are active)**
Data lines to specify charges
***END STEP**

Additional references

- Czekanski, A., N. El-Abbasi, and S. A. Meguid, "Optimal Time Integration Parameters for Elastodynamic Contact Problems," Communications in Numerical Methods in Engineering, vol. 17, pp. 379–384, 2001.
- Hilber, H. M., T. J. R. Hughes, and R. L. Taylor, "Improved Numerical Dissipation for Time Integration Algorithms in Structural Dynamics," Earthquake Engineering and Structural Dynamics, vol. 5, pp. 283–292, 1977.

6.3.3 EXPLICIT DYNAMIC ANALYSIS

Products: Abaqus/Explicit Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- *DYNAMIC
- "Configuring a dynamic, explicit procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

An explicit dynamic analysis:

- is computationally efficient for the analysis of large models with relatively short dynamic response times and for the analysis of extremely discontinuous events or processes;
- allows for the definition of very general contact conditions ("Contact interaction analysis: overview," Section 36.1.1);
- uses a consistent, large-deformation theory—models can undergo large rotations and large deformation;
- can use a geometrically linear deformation theory—strains and rotations are assumed to be small (see "Defining an analysis," Section 6.1.2);
- can be used to perform an adiabatic stress analysis if inelastic dissipation is expected to generate heat in the material (see "Adiabatic analysis," Section 6.5.4);
- can be used to perform quasi-static analyses with complicated contact conditions; and
- allows for either automatic or fixed time incrementation to be used—by default, Abaqus/Explicit uses automatic time incrementation with the global time estimator.

Explicit dynamic analysis

The explicit dynamics procedure performs a large number of small time increments efficiently. An explicit central-difference time integration rule is used; each increment is relatively inexpensive (compared to the direct-integration dynamic analysis procedure available in Abaqus/Standard) because there is no solution for a set of simultaneous equations. The explicit central-difference operator satisfies the dynamic equilibrium equations at the beginning of the increment, t; the accelerations calculated at time t are used to advance the velocity solution to time $t + \Delta t/2$ and the displacement solution to time $t + \Delta t$.

Input File Usage: *DYNAMIC, EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**

Numerical implementation

The explicit dynamics analysis procedure is based upon the implementation of an explicit integration rule together with the use of diagonal ("lumped") element mass matrices. The equations of motion for the body are integrated using the explicit central-difference integration rule

$$\dot{u}^N_{(i+\frac{1}{2})} = \dot{u}^N_{(i-\frac{1}{2})} + \frac{\Delta t_{(i+1)} + \Delta t_{(i)}}{2}\ddot{u}^N_{(i)},$$

$$u^N_{(i+1)} = u^N_{(i)} + \Delta t_{(i+1)}\dot{u}^N_{(i+\frac{1}{2})},$$

where u^N is a degree of freedom (a displacement or rotation component) and the subscript *i* refers to the increment number in an explicit dynamics step. The central-difference integration operator is explicit in the sense that the kinematic state is advanced using known values of $\dot{u}^N_{(i-1/2)}$ and $\ddot{u}^N_{(i)}$ from the previous increment.

The explicit integration rule is quite simple but by itself does not provide the computational efficiency associated with the explicit dynamics procedure. The key to the computational efficiency of the explicit procedure is the use of diagonal element mass matrices because the accelerations at the beginning of the increment are computed by

$$\ddot{u}^N_{(i)} = \left(M^{NJ}\right)^{-1}(P^J_{(i)} - I^J_{(i)}),$$

where M^{NJ} is the mass matrix, P^J is the applied load vector, and I^J is the internal force vector. A lumped mass matrix is used because its inverse is simple to compute and because the vector multiplication of the mass inverse by the inertial force requires only *n* operations, where *n* is the number of degrees of freedom in the model. The explicit procedure requires no iterations and no tangent stiffness matrix. The internal force vector, I^J, is assembled from contributions from the individual elements such that a global stiffness matrix need not be formed.

Nodal mass and inertia

The explicit integration scheme in Abaqus/Explicit requires nodal mass or inertia to exist at all activated degrees of freedom (see "Conventions," Section 1.2.2) unless constraints are applied using boundary conditions. More precisely, a nonzero nodal mass must exist unless all activated translational degrees of freedom are constrained and nonzero rotary inertia must exist unless all activated rotational degrees of freedom are constrained. Nodes that are part of a rigid body do not require mass, but the entire rigid body must possess mass and inertia unless constraints are used. Nodes that belong to Eulerian elements also do not require mass, since the surrounding Eulerian elements may be void at some time during the simulation.

When degrees of freedom at a node are activated by elements with a nonzero mass density (e.g., solid, shell, beam) or mass and inertia elements, a nonzero nodal mass or inertia occurs naturally from the assemblage of lumped mass contributions.

When degrees of freedom at a node are activated by elements with no mass (e.g., spring, dashpot, or connector elements), care must be taken either to constrain the node or to add mass and inertia as appropriate.

Stability

The explicit procedure integrates through time by using many small time increments. The central-difference operator is conditionally stable, and the stability limit for the operator (with no damping) is given in terms of the highest frequency of the system as

$$\Delta t \leq \frac{2}{\omega_{max}}.$$

With damping, the stable time increment is given by

$$\Delta t \leq \frac{2}{\omega_{max}}(\sqrt{1+\xi_{max}^2} - \xi_{max}),$$

where ξ_{max} is the fraction of critical damping in the mode with the highest frequency. Contrary to our usual engineering intuition, introducing damping to the solution reduces the stable time increment. In Abaqus/Explicit a small amount of damping is introduced in the form of bulk viscosity to control high frequency oscillations. Physical forms of damping, such as dashpots or material damping, can also be introduced. Bulk viscosity and material damping are discussed below.

Estimating the stable time increment size

An approximation to the stability limit is often written as the smallest transit time of a dilatational wave across any of the elements in the mesh

$$\Delta t \approx \frac{L_{min}}{c_d},$$

where L_{min} is the smallest element dimension in the mesh and c_d is the dilatational wave speed in terms of λ_0 and μ_0, defined below.

In general, for beams, conventional shells, and membranes the element thickness or cross-sectional dimensions are not considered in determining the smallest element dimension; the stability limit is based upon the midplane or membrane dimensions only. When the transverse shear stiffness is defined for shell elements (see "Shell section behavior," Section 29.6.4), the stable time increment will also be based on the transverse shear behavior.

This estimate for Δt is only approximate and in most cases is not a conservative (safe) estimate. In general, the actual stable time increment chosen by Abaqus/Explicit will be less than this estimate by a factor between $1/\sqrt{2}$ and 1 in a two-dimensional model and between $1/\sqrt{3}$ and 1 in a three-dimensional model. The time increment chosen by Abaqus/Explicit also accounts for any stiffness behavior in a model associated with penalty contact. For further discussion, see "Computational cost" below.

Stable time increment report

Abaqus/Explicit writes a report to the status (`.sta`) file during the data check phase of the analysis that contains an estimate of the minimum stable time increment and a listing of the elements with the smallest stable time increments and their values. The initial stable time increments listed do not include damping (bulk viscosity), mass scaling, or penalty contact effects.

This listing is provided because often a few elements have much smaller stability limits than the rest of the elements in the mesh. The stable time increment can be increased by modifying the mesh to increase the size of the controlling element or by using appropriate mass scaling.

Dilatational wave speed

The current dilatational wave speed, c_d, is determined in Abaqus/Explicit by calculating the effective hypoelastic material moduli from the material's constitutive response. Effective Lamé's constants, $\hat{\lambda}$ and $\hat{G} = 2\hat{\mu}$, are determined in the following manner. Define Δp as the increment in the mean stress, $\Delta \mathbf{S}$ as the increment in the deviatoric stress, $\Delta \epsilon_{vol}$ as the increment of volumetric strain, and $\Delta \mathbf{e}$ as the deviatoric strain increment. We assume a hypoelastic stress-strain rule of the form

$$\Delta p = (3\hat{\lambda} + 2\hat{\mu})\Delta \epsilon_{vol},$$

$$\Delta \mathbf{S} = 2\hat{\mu}\Delta \mathbf{e} \ .$$

The effective moduli can then be computed as

$$3\hat{K} = 3\hat{\lambda} + 2\hat{\mu} = \frac{\Delta p}{\Delta \epsilon_{vol}},$$

$$2\hat{\mu} = \frac{\Delta \mathbf{S} : \Delta \mathbf{e}}{\Delta \mathbf{e} : \Delta \mathbf{e}},$$

$$\hat{\lambda} + 2\hat{\mu} = \frac{1}{3}(3\hat{K} + 4\hat{\mu}) \ .$$

For shell elements defined by a shell cross-section that requires numerical integration (see "Using a shell section integrated during the analysis to define the section behavior," Section 29.6.5), the effective moduli for the section are computed by integrating the effective moduli at the section points through the thickness. These effective moduli represent the element stiffness and determine the current dilatational wave speed in the element as

$$c_d = \sqrt{\frac{\hat{\lambda} + 2\hat{\mu}}{\rho}} \ ,$$

where ρ is the density of the material.

In an isotropic, elastic material the effective Lamé's constants can be defined in terms of Young's modulus, E, and Poisson's ratio, ν, by

$$\hat{\lambda} = \lambda_0 = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

and

$$\hat{\mu} = \mu_0 = \frac{E}{2(1+\nu)}.$$

Time incrementation

The time increment used in an analysis must be smaller than the stability limit of the central-difference operator. Failure to use a small enough time increment will result in an unstable solution. When the solution becomes unstable, the time history response of solution variables such as displacements will usually oscillate with increasing amplitudes. The total energy balance will also change significantly.

If the model contains only one material type, the initial time increment is directly proportional to the size of the smallest element in the mesh. If the mesh contains uniform size elements but contains multiple material descriptions, the element with the highest wave speed will determine the initial time increment.

In nonlinear problems—those with large deformations and/or nonlinear material response—the highest frequency of the model will continually change, which consequently changes the stability limit. Abaqus/Explicit has two strategies for time incrementation control: fully automatic time incrementation (where the code accounts for changes in the stability limit) and fixed time incrementation.

Scaling the time increment

To reduce the chance of a solution going unstable, you can adjust the stable time increment computed by Abaqus/Explicit by a constant scaling factor. This factor can be used to scale the default global time estimate, the element-by-element estimate, or the fixed time increment based on the initial element-by-element estimate; it cannot be used to scale a fixed time increment specified directly by you.

Input File Usage: Use the following option to scale the stable time increment based on the global time estimate:

*DYNAMIC, EXPLICIT, SCALE FACTOR=*f*

Use the following option to scale the stable time increment based on the element-by-element estimate:

*DYNAMIC, EXPLICIT, ELEMENT BY ELEMENT, SCALE FACTOR=*f*

Use the following option to scale the stable time increment based on the fixed time increment on the initial element-by-element estimate:

*DYNAMIC, EXPLICIT, FIXED TIME INCREMENTATION,
SCALE FACTOR=*f*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Time scaling factor**: f

Automatic time incrementation

The default time incrementation scheme in Abaqus/Explicit is fully automatic and requires no user intervention. Two types of estimates are used to determine the stability limit: element by element and global. An analysis always starts by using the element-by-element estimation method and may switch to the global estimation method under certain circumstances, as explained below.

Element-by-element estimation

In an analysis Abaqus/Explicit initially uses a stability limit based on the highest element frequency in the whole model. This element-by-element estimate is determined using the current dilatational wave speed in each element.

The element-by-element estimate is conservative; it will give a smaller stable time increment than the true stability limit that is based upon the maximum frequency of the entire model. In general, constraints such as boundary conditions and kinematic contact have the effect of compressing the eigenvalue spectrum, and the element-by-element estimates do not take this into account.

The concept of the stable time increment as the time required to propagate a dilatational wave across the smallest element dimension is useful for interpreting how the explicit procedure chooses the time increment when element-by-element stability estimation controls the time increment. As the step proceeds, the global stability estimate, if used, will make the time increment less sensitive to element size.

Input File Usage: *DYNAMIC, EXPLICIT, ELEMENT BY ELEMENT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Stable increment estimator**: **Element-by-element**

Global estimation

The stability limit will be determined by the global estimator as the step proceeds unless the element-by-element estimation method is specified, fixed time incrementation is specified, or one of the conditions explained below prevents the use of global estimation. The switch to the global estimation method occurs once the algorithm determines that the accuracy of the global estimation method is acceptable.

The adaptive, global estimation algorithm determines the maximum frequency of the entire model using the current dilatational wave speed. This algorithm continuously updates the estimate for the maximum frequency. The global estimator will usually allow time increments that exceed the element-by-element values.

Abaqus/Explicit monitors the effectiveness of the global estimation algorithm. If the cost for computing the global time estimate is more than its benefit, the code will turn off the global estimation algorithm and simply use the element-by-element estimates to save computation time.

Conditions that will prevent the use of the global time estimator

The global estimation algorithm will not be used when any of the following capabilities are included in the model:

- Fluid elements
- Infinite elements
- Dashpots
- Thick shells (thickness to characteristic length ratio larger than 0.92)
- Thick beams (thickness to length ratio larger than 1.0)
- The JWL equation of state
- Material damping
- Nonisotropic elastic materials with temperature and field variable dependency
- Distortion control
- Adaptive meshing
- Subcycling

"Improved" stable time increment for three-dimensional continuum elements and elements with plane stress formulations

For three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements) an "improved" estimate of the element characteristic length is used by default. This "improved" method usually results in a larger element stable time increment than a more traditional method. For analyses using variable mass scaling, the total mass added to achieve a given stable time increment will be less with the improved estimate.

Input File Usage: Use the following option to activate the "improved" element time estimation method:

*DYNAMIC, EXPLICIT, IMPROVED DT METHOD=YES

Use the following option to deactivate the "improved" element time estimation method:

*DYNAMIC, EXPLICIT, IMPROVED DT METHOD=NO

Abaqus/CAE Usage: The ability to deactivate the "improved" element time estimation method is not supported in Abaqus/CAE.

Fixed time incrementation

A fixed time incrementation scheme is also available in Abaqus/Explicit. The fixed time increment size is determined either by the initial element-by-element stability estimate for the step or by a user-specified time increment.

Fixed time incrementation may be useful when a more accurate representation of the higher mode response of a problem is required. In this case a time increment size smaller than the element-by-element

estimates may be used. The element-by-element estimate can be obtained simply by running a data check analysis (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

When fixed time incrementation is used, Abaqus/Explicit will not check that the computed response is stable during the step. You should ensure that a valid response has been obtained by carefully checking the energy history and other response variables.

Basing the fixed time increment size on the initial element-by-element stability limit

You can use time increments the size of the initial element-by-element stability limit throughout a step. The dilatational wave speed in each element at the beginning of the step is used to compute the fixed time increment size.

Input File Usage: *DYNAMIC, EXPLICIT, FIXED TIME INCREMENTATION

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Fixed**: **Use element-by-element time increment estimator**

Specifying the fixed time increment size directly

Alternatively, you can specify a time increment size directly.

Input File Usage: *DYNAMIC, EXPLICIT, DIRECT USER CONTROL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Incrementation**: **Type: Fixed**: **User-defined time increment**

Advantages of the explicit method

The use of small increments (dictated by the stability limit) is advantageous because it allows the solution to proceed without iterations and without requiring tangent stiffness matrices to be formed. It also simplifies the treatment of contact.

The explicit dynamics procedure is ideally suited for analyzing high-speed dynamic events, but many of the advantages of the explicit procedure also apply to the analysis of slower (quasi-static) processes. A good example is sheet metal forming, where contact dominates the solution and local instabilities may form due to wrinkling of the sheet.

The results in an explicit dynamics analysis are not automatically checked for accuracy as they are in Abaqus/Standard (Abaqus/Standard uses the half-increment residual). In most cases this is not of concern because the stability condition imposes a small time increment such that the solution changes only slightly in any one time increment, which simplifies the incremental calculations. While the analysis may take an extremely large number of increments, each increment is relatively inexpensive, often resulting in an economical solution. It is not uncommon for Abaqus/Explicit to take over 10^5 increments for an analysis. The method is, therefore, computationally attractive for problems where the total dynamic response time that must be modeled is only a few orders of magnitude longer than the stability limit; for example, wave propagation studies or some "event and response" applications.

Computational cost

The computer time involved in running a simulation using explicit time integration with a given mesh is proportional to the time period of the event. The time increment based on the element-by-element stability estimate can be rewritten (ignoring damping) in the form

$$\Delta t \leq \min\left(L_e\sqrt{\frac{\rho}{\hat{\lambda}+2\hat{\mu}}}\right),$$

where the minimum is taken over all elements in the mesh, L_e is a characteristic length associated with an element (see "Explicit dynamic analysis," Section 2.4.5 of the Abaqus Theory Guide), ρ is the density of the material in the element, and $\hat{\lambda}$ and $\hat{\mu}$ are the effective Lamé's constants for the material in the element (defined above).

The time increment from the global stability estimate may be somewhat larger, but for this discussion we will assume that the above inequality always holds (when the inequality does not hold, the solution time will be somewhat faster).

For linear, nonisotropic elastic materials this stability limit is further scaled down by the square root of the ratio of the effective material stiffness to the maximum material stiffness in one particular direction. Since this effectively means that the time increment can be no larger than the time required to propagate a stress wave across an element, the computer time involved in running a quasi-static analysis can be very large: the cost of the simulation is directly proportional to the number of time increments required.

The number of increments, n, required is $n = T/\Delta t$ if Δt remains constant, where T is the time period of the event being simulated. (Even the element-by-element approximation of Δt will not remain constant in general, since element distortion will change L_e and nonlinear material response will change the effective Lamé constants. But the assumption is sufficiently accurate for the purposes of this discussion.) Thus,

$$n \approx T\max\left(\frac{1}{L_e}\sqrt{\frac{\hat{\lambda}+2\hat{\mu}}{\rho}}\right).$$

In a two-dimensional analysis refining the mesh by a factor of two in each direction will increase the run time in the explicit procedure by a factor of eight—four times as many elements and half the original time increment size. Similarly, in a three-dimensional analysis refining the mesh by a factor of two in each direction will increase the run time by a factor of sixteen.

In a quasi-static analysis it is expedient to reduce the computational cost by either speeding up the simulation or by scaling the mass. In either case the kinetic energy should be monitored to ensure that the ratio of kinetic energy to internal energy does not get too large—typically less than 10%.

Reducing the computational cost by speeding up the simulation

To reduce the number of increments required, n, we can speed up the simulation compared to the time of the actual process—that is, we can artificially reduce the time period of the event, T. This will introduce

two possible errors. If the simulation speed is increased too much, the increased inertia forces will change the predicted response (in an extreme case the problem will exhibit wave propagation response). The only way to avoid this error is to choose a speed-up that is not too large.

The other error is that some aspects of the problem other than inertia forces—for example, material behavior—may also be rate dependent. In this case the actual time period of the event being modeled cannot be changed.

Reducing the computational cost by using mass scaling

Artificially increasing the material density, ρ, by a factor f^2 reduces n to n/f, just like decreasing T to T/f. This concept, called "mass scaling," reduces the ratio of the event time to the time for wave propagation across an element while leaving the event time fixed, which allows rate-dependent behavior to be included in the analysis. Mass scaling has exactly the same effect on inertia forces as speeding up the time of simulation.

Mass scaling is attractive because it can be used in rate-dependent problems, but it must be used with care to ensure that the inertia forces do not dominate and change the solution. Either fixed or variable mass scaling can be invoked (see "Mass scaling," Section 11.6.1).

Mass scaling can also be accomplished by altering the density; however, the fixed and variable mass scaling capabilities provide more versatile methods of scaling the mass of the entire model or specific element sets in the model.

Reducing the computational cost by using selective subcycling

One disadvantage in an explicit dynamic analysis is that a few very small elements will force the entire model to be integrated with a small time increment. You can use mixed time integration or "subcycling" methods to reduce this problem. In these methods the equations of motion for the body are still integrated using the explicit central-difference integration rule as shown above, but the different time increments are allowed for different groups of nodes in the finite element model. If most nodes are integrated with a large stable time increment and only a few nodes are integrated with a small time increment, the computational cost may be reduced significantly.

Selective subcycling can be invoked by defining the subcycling zones. See "Selective subcycling," Section 11.7.1 for details.

Bulk viscosity

Bulk viscosity introduces damping associated with volumetric straining. Its purpose is to improve the modeling of high-speed dynamic events (see "Stability" above for a discussion of the effect of damping on the stable time increment). Abaqus/Explicit contains two forms of bulk viscosity: linear and quadratic. Linear bulk viscosity is included by default in an Abaqus/Explicit analysis.

The bulk viscosity parameters b_1 and b_2 defined below can be redefined and can be changed from step to step. If the default values are changed in a step, the new values will be used in subsequent steps until they are redefined. Bulk viscosities defined this way apply to the whole model. For an individual element set the linear and quadratic bulk viscosities can be scaled by a factor by defining section controls (see "Section controls," Section 27.1.4).

Input File Usage: Use the following option to define bulk viscosity for the entire model:

*BULK VISCOSITY

Use the following options to define bulk viscosity for an individual element set:

*BULK VISCOSITY
*SECTION CONTROLS

Abaqus/CAE Usage: Use the following option to define bulk viscosity for the entire model:

Step module: **Create Step**: **General**: **Dynamic, Explicit**: **Other**: **Linear bulk viscosity parameter** and **Quadratic bulk viscosity parameter**

Defining bulk viscosity for an individual element set is not supported in Abaqus/CAE.

Linear bulk viscosity

Linear bulk viscosity is found in all elements and is introduced to damp "ringing" in the highest element frequency. This damping is sometimes referred to as truncation frequency damping. It generates a bulk viscosity pressure that is linear in the volumetric strain rate

$$p_{bv1} = b_1 \rho c_d L_e \dot{\epsilon}_{vol},$$

where b_1 is a damping coefficient (default=.06), ρ is the current material density, c_d is the current dilatational wave speed, L_e is an element characteristic length, and $\dot{\epsilon}_{vol}$ is the volumetric strain rate.

For acoustic elements, the bulk viscosity pressure can be obtained from the above equation by using the relationship of the fluid particle velocity and the pressure rate (see "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide) as

$$p_{bv1} = -\frac{b_1 L_e}{c} \dot{p},$$

where $\dot{p}$ and c are the pressure rate and the speed of sound in the fluid, respectively.

Quadratic bulk viscosity

The second form of bulk viscosity pressure is found only in solid continuum elements (except the plane stress element CPS4R). This form is quadratic in the volumetric strain rate

$$p_{bv2} = \rho (b_2 L_e \dot{\epsilon}_{vol})^2,$$

where b_2 is a damping coefficient (default=1.2) and all other quantities are as defined for the linear bulk viscosity. Quadratic bulk viscosity is applied only if the volumetric strain rate is compressive.

The quadratic bulk viscosity pressure will smear a shock front across several elements and is introduced to prevent elements from collapsing under extremely high velocity gradients. Consider a simple one-element problem in which the nodes on one side of the element are fixed and the nodes on the other side have an initial velocity in the direction of the fixed nodes. If the initial velocity is equal to the dilatational wave speed of the material, without the quadratic bulk viscosity, the element would

collapse to zero volume in one time increment (because the stable time increment size is precisely the transit time of a dilatational wave across the element). The quadratic bulk viscosity pressure will introduce a resisting pressure that will prevent the element from collapsing.

Fraction of critical damping due to bulk viscosity

The bulk viscosity pressure is not included in the material point stresses because it is intended as a numerical effect only—it is not considered part of the material's constitutive response. The bulk viscosity pressures are based upon the dilatational mode of each element. The fraction of critical damping in the dilatational mode of each element is given by

$$\xi = b_1 - b_2^2 \frac{L_e}{c_d} \min(0, \dot{\epsilon}_{vol}) \, .$$

Rotational bulk viscosity for shell elements

For the displacement degrees of freedom, bulk viscosity introduces damping associated with volumetric straining. Linear bulk viscosity or truncation frequency damping is used to damp the high frequency ringing that leads to unwanted noise in the solution or spurious overshoot in the response amplitude. For the same reason, in shells the high frequency ringing in the rotational degrees of freedom is damped with linear bulk viscosity acting on the mean curvature strain rate. This damping generates a bulk viscosity "pressure moment," m, which is linear in the mean curvature strain rate

$$m = b_1 \frac{h_0^2}{12} \rho c_d L \dot{\kappa} \, ,$$

where b_1 is a damping coefficient (default = 0.06), h_0 is the original thickness, ρ is the mass density, c_d is the current dilatational wave speed, L is the characteristic length used for rotary inertia and transverse shear stiffness scaling (see "Finite-strain shell element formulation," Section 3.6.5 of the Abaqus Theory Guide), and $\dot{\kappa} = \dot{\kappa}_{11} + \dot{\kappa}_{22}$ is twice the mean curvature strain rate. The resultant pressure moment mh, where h is the current thickness, is added to the direct components of the moment resultant.

Material damping

Defining inelastic material behavior, dashpots, etc. will introduce energy dissipation into a model. In addition to these mechanisms, general ("Rayleigh") material damping can be introduced (see "Material damping," Section 26.1.1). Adding damping to a model, especially stiffness proportional damping, β_R, may significantly reduce the stable time increment.

Input File Usage: *DAMPING, ALPHA=α_R, BETA=β_R

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damping**: **Alpha** and **Beta**

Obtaining diagnostic information about critical elements

Abaqus/Explicit writes critical elements (elements with the smallest stable time increments) and their stable time increment values to the output database at each summary increment for visualization in Abaqus/CAE. By default, the number of critical elements written to the output database is 10.

Input File Usage: *DIAGNOSTICS, CRITICAL ELEMENTS=*value*

Abaqus/CAE Usage: The ability to control the number of critical elements written to the output database is not supported in Abaqus/CAE.

Obtaining diagnostic information about the deformation speed

The deformation speed in an element is defined as the largest absolute value of all the deformation rate components of an element times the element characteristic length, L_e. You can request diagnostic information about the deformation speed within a step definition, as described below. In a multistep analysis diagnostic requests remain in effect until they are explicitly redefined.

Deformation speed warnings

By default, Abaqus/Explicit will check for a relatively large deformation speed in all the elements since too high a value may cause the element to deform or collapse unrealistically. A warning message is issued if the ratio of deformation speed versus dilatational wave speed in an element reaches the value specified for the "warning ratio." By default, the warning ratio is 0.3. You can redefine this limit.

The first occurrence of the warning message is written to the status (`.sta`) file; subsequent occurrences are written to the message (`.msg`) file. See "Output," Section 4.1.1, for a description of these output files.

Generally when the ratio of deformation speed to dilatational wave speed is greater than 0.3, it is an indication that the purely mechanical material constitutive relationship is no longer valid and that a thermomechanical equation of state material is required.

Input File Usage: *DIAGNOSTICS, WARNING RATIO=*ratio*

Abaqus/CAE Usage: The ability to redefine the warning ratio limit is not supported in Abaqus/CAE.

Deformation speed errors

An error message is issued and the analysis is terminated when the maximum ratio of deformation speed versus current dilatational wave speed for any element is greater than the "cutoff ratio." By default, the cutoff ratio is 1.0. You can redefine this limit.

The check for this cutoff ratio is not applied to any model that has an equation of state material (see "Equation of state," Section 25.2.1) or a user-defined material (see "User-defined mechanical material behavior," Section 26.7.1).

Input File Usage: *DIAGNOSTICS, CUTOFF RATIO=*ratio*

Abaqus/CAE Usage: The ability to redefine the cutoff ratio limit is not supported in Abaqus/CAE.

Obtaining a summary of the deformation speed information

You can request summary diagnostic information to obtain warning and error messages for only the element with the largest ratio of deformation speed to dilatational wave speed.

Input File Usage: *DIAGNOSTICS, DEFORMATION SPEED CHECK=SUMMARY

Abaqus/CAE Usage: A summary of the deformation speed diagnostic information is output by default in Abaqus/CAE.

Obtaining detailed deformation speed information

You can request detailed diagnostic information to obtain warning and error messages for all elements with large deformation speed to dilatational wave speed ratios.

Input File Usage: *DIAGNOSTICS, DEFORMATION SPEED CHECK=DETAIL

Abaqus/CAE Usage: You cannot output detailed diagnostic information about the deformation speed in Abaqus/CAE.

Disabling deformation speed checks

You can choose to completely bypass the checks for large deformation speed.

Input File Usage: *DIAGNOSTICS, DEFORMATION SPEED CHECK=OFF

Abaqus/CAE Usage: You cannot disable the deformation speed checks in Abaqus/CAE.

Monitoring output variables for extreme values

There are some analyses in which it is useful to monitor the value of a variable at every increment. For example, in a force-driven analysis such as hydro-forming, the simulation time that is sufficient to model the completion of the physical process may depend on the magnitude of the displacement of a node or a group of nodes in the model. Another example is a drop test simulation where the postfailure response is not of interest. Monitoring the values of critical variables and halting the analysis when those variables exceed a given criterion can reduce computational expense and turnaround time.

For such problems Abaqus/Explicit allows output variables to be monitored during an analysis to verify whether or not their values have exceeded or fallen below user-specified values in specified element or node sets. Comparisons of specified element integration point variables, element section variables, or nodal variables with user-specified values are performed at every increment. At the first occurrence of a variable exceeding the user-specified bounds, the variable name, the associated element or node number, and the increment number are written to the status (`.sta`) file. In addition, you can request that the analysis be stopped and/or the output state be written in the increment following the one in which the variable has exceeded the user-specified bound. At the end of each step in which variables are monitored, the maximum, minimum, or absolute maximum value that each variable attains during the course of the analysis, along with the number of the element or node where the extreme value occurred, will be written to the status file.

Defining the element and nodal variables to be monitored

The element output variables that can be monitored include all the element integration point variables and element section point variables that are available for history-type output to the output database. Similarly, the nodal output variables that can be monitored include all the nodal variables that are available for history output to the output database. The keys identifying the output variables are defined in "Abaqus/Explicit output variable identifiers," Section 4.2.2.

Input File Usage: Use the first option with one or both of the following options in the history portion of the input file:

*EXTREME VALUE
*EXTREME ELEMENT VALUE, ELSET=*element_set_name*
*EXTREME NODE VALUE, NSET=*nset_set_name*

The *EXTREME VALUE option can be repeated in the same step, and the *EXTREME ELEMENT VALUE and *EXTREME NODE VALUE options can be repeated as many times as necessary.

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Halting the analysis when the extreme value criterion is met

You can choose to halt the analysis when the extreme value criterion is met. The analysis will stop at the end of the increment following the one in which any of the specified element or nodal variables exceeded the prescribed bounds.

Input File Usage: Use the following options:

*EXTREME VALUE, HALT=YES
*EXTREME ELEMENT VALUE *and/or* *EXTREME NODE VALUE

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Obtaining output when the extreme value criterion is met

You can obtain field-type output to the output database and an additional restart state when any of the selected variables fall outside the specified bounds for the first time during the analysis. The output will be written in the increment following the one in which such an occurrence took place. Since output is automatically written when the analysis terminates, this request has an effect only if you have not chosen to halt the analysis when the extreme value criterion is met as described above.

Input File Usage: Use either or both of the following options in conjunction with the *EXTREME VALUE option:

*EXTREME ELEMENT VALUE, ELSET=*element_set_name*, OUTPUT=YES
*EXTREME NODE VALUE, NSET=*node_set_name*, OUTPUT=YES

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Monitoring variables in a multistep analysis

In a multistep analysis the monitoring requests you specify remain in effect until they are redefined. You must redefine all requests to add or change any variables, element or node sets, or maxima or minima.

Stopping the monitoring of variables in a new step

You can stop monitoring variables in a new step.

Input File Usage: Use the *EXTREME VALUE option without the *EXTREME ELEMENT VALUE and *EXTREME NODE VALUE options.

Abaqus/CAE Usage: Extreme value output monitoring is not supported in Abaqus/CAE.

Initial conditions

"Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the initial conditions that are available for an explicit dynamic analysis.

Boundary conditions

Boundary conditions can be defined as explained in "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1. Boundary conditions applied during an explicit dynamic response step should use appropriate amplitude references ("Amplitude curves," Section 34.1.2). If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference will be ignored, and a zero velocity boundary condition will be enforced.

Loads

The loading types available for an explicit dynamic analysis are explained in "Applying loads: overview," Section 34.4.1. Concentrated nodal forces or moments can be applied to the displacement or rotation degrees of freedom (1–6). Distributed pressure forces or body forces can also be applied; the distributed load types available with particular elements are described in Part VI, "Elements."

As with boundary conditions, loads applied during a dynamic response step should use appropriate amplitude references ("Amplitude curves," Section 34.1.2). If loads are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step.

Predefined fields

The following predefined fields can be specified, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in explicit dynamic analysis, nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion,"

Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material options

Any of the material models in Abaqus/Explicit can be used in a general explicit dynamic analysis (see "Combining material behaviors," Section 21.1.3).

Elements

All of the elements available in Abaqus/Explicit can be used in an explicit dynamic analysis. The elements are listed in Part VI, "Elements."

If coupled temperature-displacement elements are used in an explicit dynamic analysis, the temperature degrees of freedom will be ignored.

Output

The element output available for a dynamic analysis includes stress; strain; energies; and the values of state, field, and user-defined variables. The nodal output available includes displacements, velocities, accelerations, reaction forces, and coordinates. All of the output variable identifiers are outlined in "Abaqus/Explicit output variable identifiers," Section 4.2.2. The types of output available are described in "Output," Section 4.1.1.

When an Abaqus/Explicit analysis encounters a fatal error, the preselected variables applicable to the current procedure are added automatically to the output database as field data for the last increment.

Energy output is particularly important in checking the accuracy of the solution in an explicit dynamic analysis. In general, the total energy (ETOTAL) should be a constant or close to a constant; the "artificial" energies, such as the artificial strain energy (ALLAE), the damping dissipation (ALLVD), and the mass scaling work (ALLMW) should be negligible compared to "real" energies such as the strain energy (ALLSE) and the kinetic energy (ALLKE).

In a quasi-static analysis the value of the kinetic energy (ALLKE) should not exceed a small fraction of the value of the strain energy (ALLIE).

It is a good practice to output the constraint penalty work (ALLCW) and the contact penalty work (ALLPW) in analyses involving constraints (such as ties and fasteners) and contact. The value of these energies should be close to zero.

Input file template

```
*HEADING
 …
*MATERIAL, NAME=name
*ELASTIC
 …
```

***DENSITY**
Data lines to define density
***DAMPING, ALPHA =**α**, BETA=**β
Data lines to define Rayleigh damping
…
***BOUNDARY**
Data lines to specify zero-valued boundary conditions
INITIAL CONDITIONS, TYPE=type*
Data lines to specify initial conditions
AMPLITUDE, NAME=name*
Data lines to define amplitude variations
****
***STEP**
***DYNAMIC, EXPLICIT**
Data line to specify the time period of the step
***DIAGNOSTICS, DEFORMATION SPEED CHECK=SUMMARY**
BOUNDARY, AMPLITUDE=name*
Data lines to describe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD**
Data lines to specify loading
***TEMPERATURE and/or *FIELD**
Data lines to specify predefined fields
FILE OUTPUT, NUMBER INTERVAL=n*
***EL FILE**
Data line specifying element output variables
***NODE FILE**
Data line specifying node output variables
***ENERGY FILE**
OUTPUT, FIELD, NUMBER INTERVAL=n*
***ELEMENT OUTPUT**
Data line specifying element output variables
***NODE OUTPUT**
Data line specifying node output variables
OUTPUT, HISTORY, TIME INTERVAL=t*
ELEMENT OUTPUT, ELSET=element set name*
Data line specifying element output variables
NODE OUTPUT, NSET=node set name*
Data line specifying node output variables
***ENERGY OUTPUT**
Data line specifying energy output variables
***END STEP**
****

```
*STEP
*DYNAMIC, EXPLICIT, ELEMENT BY ELEMENT
 …
*BULK VISCOSITY
```
Data line to define linear and/or quadratic bulk viscosity in this step
```
 …
*END STEP
```

6.3.4 DIRECT-SOLUTION STEADY-STATE DYNAMIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Dynamic analysis procedures: overview," Section 6.3.1
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Subspace-based steady-state dynamic analysis," Section 6.3.9
- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- *STEADY STATE DYNAMICS
- "Configuring a direct-solution steady-state dynamic procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Creating and modifying prescribed conditions," Section 16.4 of the Abaqus/CAE User's Guide

Overview

A direct-solution steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is a linear perturbation procedure;
- calculates the response directly in terms of the physical degrees of freedom of the model;
- is an alternative to mode-based steady-state dynamic analysis, in which the response of the system is calculated on the basis of the eigenmodes;
- is more expensive computationally than mode-based or subspace-based steady-state dynamics;
- is more accurate than mode-based or subspace-based steady-state dynamics, in particular if significant frequency-dependent material damping or viscoelastic material behavior is present in the structure; and
- is able to bias the excitation frequencies toward the approximate values that generate a response peak.

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system due to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep by applying the loading at a series of different frequencies and recording the response; in Abaqus/Standard the direct-solution steady-state dynamic procedure conducts this frequency sweep. In a direct-solution steady-state analysis the steady-state harmonic response is calculated directly in terms of the physical degrees of freedom of the model using the mass, damping, and stiffness matrices of the system.

When defining a direct-solution steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below ("Selecting the frequency spacing"). Logarithmic frequency spacing is the default. Frequencies are given in cycles/time.

Those frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (described below).

The direct-solution steady-state analysis procedure can be used in the following cases for which the eigenvalues cannot be extracted (and, thus, the mode-based steady-state dynamics procedures are not applicable):

- for nonsymmetric stiffness;
- when any form of damping other than modal damping must be included; and
- when viscoelastic material properties must be taken into account.

While the response in this procedure is linear, the prior response can be nonlinear. Initial stress effects (stress stiffening) as well as load stiffness effects will be included in the steady-state dynamics response if nonlinear geometric effects ("General and linear perturbation procedures," Section 6.1.3) were included in any general analysis step prior to the direct-solution steady-state dynamic procedure.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**

Ignoring damping

If damping terms can be ignored, you can specify that a real, rather than a complex, system matrix be factored, which can significantly reduce computational time. Damping is discussed below.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT, REAL ONLY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Compute real response only**

Selecting the type of frequency interval for which output is requested

Three types of frequency intervals are permitted for output from a direct-solution steady-state dynamic step. If an eigenvalue extraction step precedes the direct-solution steady-state dynamic step, you can select either the range or the eigenfrequency type of frequency interval; otherwise, only the range type can be used.

Dividing the specified frequency range using the user-defined number of points and the optional bias function

For the range type of frequency interval (the default), the specified frequency range of interest is divided using the user-defined number of points and the optional bias function.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Specifying the frequency ranges by using the system's eigenfrequencies

If the direct-solution steady-state dynamic analysis is preceded by an eigenfrequency extraction step, you can select the eigenfrequency type of frequency interval. The following intervals then exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function. Figure 6.3.4–1 illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Use eigenfrequencies to subdivide each frequency range**

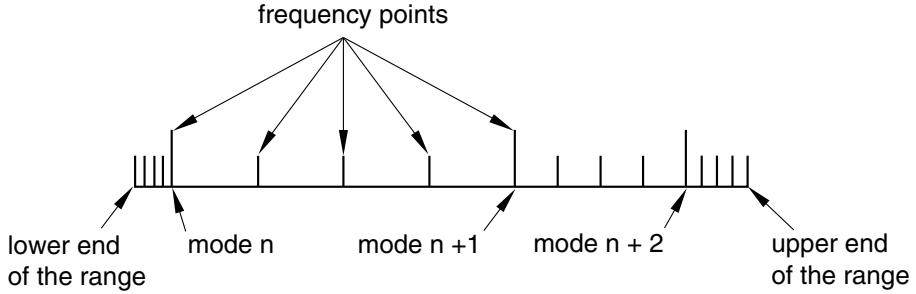


Figure 6.3.4–1 Division of range for the eigenfrequency type of interval and 5 calculation points.

Specifying the frequency ranges by the frequency spread

If the direct-solution steady-state dynamic analysis is preceded by an eigenfrequency extraction step, you can select the spread type of frequency interval. In this case intervals exist around each

eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. Figure 6.3.4–2 illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

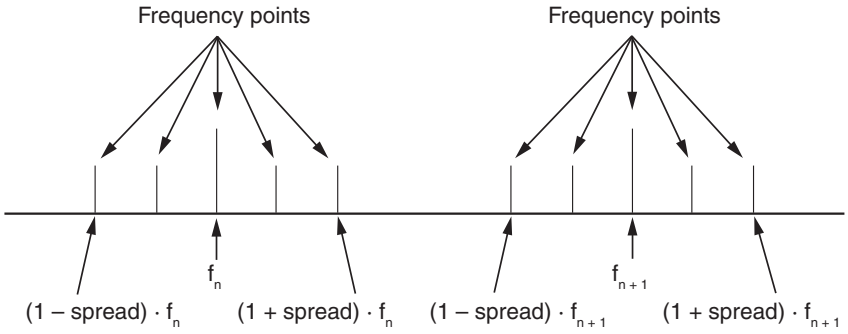


Figure 6.3.4–2 Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

Selecting the frequency spacing

Two types of frequency spacing are permitted for a direct-solution steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use either of the following options:

*STEADY STATE DYNAMICS, DIRECT,
FREQUENCY SCALE=LOGARITHMIC
*STEADY STATE DYNAMICS, DIRECT, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Scale: Logarithmic** or **Linear**

Requesting multiple frequency ranges

You can request multiple frequency ranges or multiple single frequency points for a direct-solution steady-state dynamic step.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...
Repeat the data lines as often as necessary.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Data**: enter data in table, and add rows as necessary

The bias parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. Figure 6.3.4–3 shows a few examples of the effect of the bias parameter on the frequency spacing.

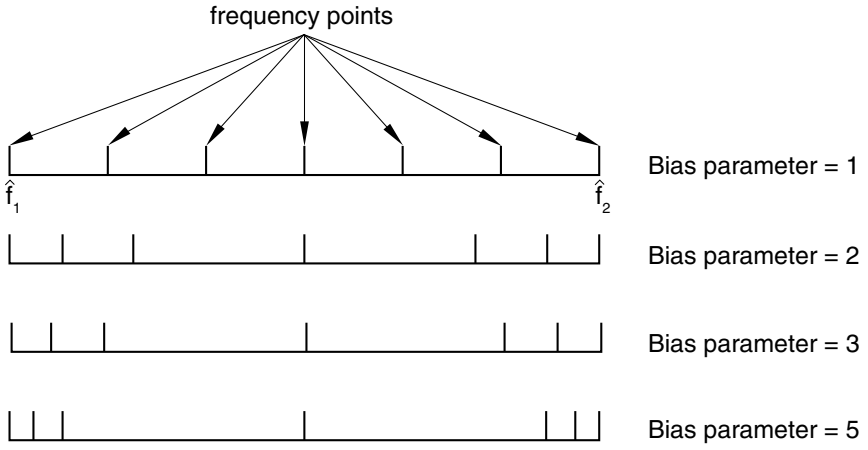


Figure 6.3.4–3 Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used in direct-solution steady-state dynamics is

$$\hat{f}_k = \frac{1}{2}(\hat{f}_1 + \hat{f}_2) + \frac{1}{2}(\hat{f}_2 - \hat{f}_1)|y|^{1/p}\text{sign}(y),$$

where

y	$= -1 + 2(k-1)/(n-1)$;
n	is the number of frequency points at which results are to be given;
k	is one such frequency point ($k = 1, 2, \ldots, n$);
$\hat{f}_1$	is the lower limit of the frequency range;
$\hat{f}_2$	is the upper limit of the range;
$\hat{f}_k$	is the frequency at which the kth results are given;
p	is the bias parameter value; and
$\hat{f}$	is the frequency or the logarithm of the frequency, depending on the value chosen for the frequency scale.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 1.0 for a range frequency interval and 3.0 for an eigenfrequency interval.

The frequency scale factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see "Specifying the frequency ranges by using the system's eigenfrequencies," above).

Damping

If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in "Material damping," Section 26.1.1.

In direct-solution steady-state dynamics damping can be created by the following:

- dashpots (see "Dashpots," Section 32.2.1),
- "Rayleigh" damping associated with materials and elements (see "Material damping," Section 26.1.1),
- damping associated with acoustic elements (see "Acoustic medium," Section 26.3.1; "Infinite elements," Section 28.3.1; and "Acoustic and shock loads," Section 34.4.6),
- structural damping (see "Damping in dynamic analysis" in "Dynamic analysis procedures: overview," Section 6.3.1), and
- viscoelasticity included in the material definitions (see "Frequency domain viscoelasticity," Section 22.7.2).

When a real-only system matrix is factored, all forms of damping are ignored, including quiet boundaries on infinite elements and nonreflecting boundaries on acoustic elements.

Contact conditions with sliding friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom are not constrained and the effect of friction results in an unsymmetric contribution to the stiffness matrix. At other contact nodes the tangential degrees of freedom are constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see "Coulomb friction," Section 5.2.3 of the Abaqus Theory Guide. Direct-solution steady-state dynamics analysis allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: *STEADY STATE DYNAMICS, DIRECT, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Direct**: **Include friction-induced damping effects**

Initial conditions

The base state is the current state of the model at the end of the last general analysis step prior to the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary conditions

In a steady-state dynamic analysis the real and imaginary parts of any degree of freedom are either restrained or unrestrained simultaneously; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will automatically restrain both the real and imaginary parts of a degree of freedom even if only one part is prescribed specifically. The unspecified part will be assumed to have a perturbation magnitude of zero.

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6) in a direct-solution steady-state analysis. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1. These boundary conditions will vary sinusoidally with time. You specify the real (in-phase) part of a boundary condition and the imaginary (out-of-phase) part of a boundary condition separately.

Input File Usage: Use either of the following options to define the real (in-phase) part of the boundary condition:

*BOUNDARY
*BOUNDARY, REAL

Use the following option to define the imaginary (out-of-phase) part of the boundary condition:

*BOUNDARY, IMAGINARY

Abaqus/CAE Usage: Load module: boundary condition editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**

Frequency-dependent boundary conditions

An amplitude definition can be used to specify the amplitude of a boundary condition as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

*AMPLITUDE, NAME=*name*
*BOUNDARY, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:** *name*

Load module: boundary condition editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**: **Amplitude:** *name*

Loads

The following loads can be prescribed in a steady-state dynamic analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."
- Incident wave loads can be applied; see "Acoustic and shock loads," Section 34.4.6.

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Coriolis distributed loading adds an imaginary antisymmetric contribution to the overall system of equations. This contribution is currently accounted for in solid and truss elements only and is activated by using the unsymmetric matrix storage and solution scheme for the step ("Defining an analysis," Section 6.1.2).

Incident wave loads can be used to model sound waves from distinct planar or spherical sources or from diffuse fields.

Fluid flux loading cannot be used in a steady-state dynamic analysis.

Input File Usage: Use any of the following options to define the real (in-phase) part of the load:

*CLOAD *or* *DLOAD
*CLOAD *or* *DLOAD, REAL

Use either of the following options to define the imaginary (out-of-phase) part of the load:

*CLOAD *or* *DLOAD, IMAGINARY

Abaqus/CAE Usage: Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**

Frequency-dependent loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

∗AMPLITUDE, NAME=*name*
∗CLOAD *or* ∗DLOAD, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:** *name*

Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**: **Amplitude:** *name*

Predefined fields

Predefined temperature fields can be specified in direct-solution steady-state dynamic analysis (see "Predefined fields," Section 34.6.1) and will produce harmonically varying thermal strains if thermal expansion is included in the material definition ("Thermal expansion," Section 26.1.2). Other predefined fields are ignored.

Material options

As in any dynamic analysis procedure, mass or density ("Density," Section 21.2.1) must be assigned to some regions of any separate parts of the model where dynamic response is required. If an analysis is desired in which the inertia effects are neglected, the density should be set to a very small number. The following material properties are not active during steady-state dynamic analyses: plasticity and other inelastic effects, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see "General and linear perturbation procedures," Section 6.1.3.

Viscoelastic effects can be included in direct-solution steady-state harmonic response analysis. The linearized viscoelastic response is considered to be a perturbation about a nonlinear preloaded state, which is computed on the basis of purely elastic behavior (long-term response) in the viscoelastic components. Therefore, the vibration amplitude must be sufficiently small so that the material response in the dynamic phase of the problem can be treated as a linear perturbation about the predeformed state. Viscoelastic frequency domain response is described in "Frequency domain viscoelasticity," Section 22.7.2.

Elements

Any of the following elements available in Abaqus/Standard can be used in a steady-state dynamic procedure:

- stress/displacement elements (other than generalized axisymmetric elements with twist);

- acoustic elements;
- piezoelectric elements; or
- hydrostatic fluid elements.

See “Choosing the appropriate element for an analysis type,” Section 27.1.3.

Output

In direct-solution steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see “Abaqus/Standard output variable identifiers,” Section 4.2.1). In the case of data file output the first printed line gives the magnitudes while the second lists the phase angle.

The following variables are provided specifically for steady-state dynamic analysis:

Element integration point variables:

PHS	Magnitude and phase angle of all stress components.
PHE	Magnitude and phase angle of all strain components.
PHEPG	Magnitude and phase angles of the electrical potential gradient vector.
PHEFL	Magnitude and phase angles of the electrical flux vector.
PHMFL	Magnitude and phase angle of the mass flow rate in fluid link elements.
PHMFT	Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF	Magnitude and phase angle of connector total forces.
PHCEF	Magnitude and phase angle of connector elastic forces.
PHCVF	Magnitude and phase angle of connector viscous forces.
PHCRF	Magnitude and phase angle of connector reaction forces.
PHCSF	Magnitude and phase angle of connector friction forces.
PHCU	Magnitude and phase angle of connector relative displacements.
PHCCU	Magnitude and phase angle of connector constitutive displacements.
PHCV	Magnitude and phase angle of connector relative velocities.
PHCA	Magnitude and phase angle of connector relative accelerations.

Nodal variables:

PU	Magnitude and phase angle of all displacement/rotation components at a node.
PPOR	Magnitude and phase angle of the fluid, pore, or acoustic pressure at a node.
PHPOT	Magnitude and phase angle of the electrical potential at a node.
PRF	Magnitude and phase angle of all reaction forces/moments at a node.
PHCHG	Magnitude and phase angle of the reactive charge at a node.

The total kinetic energy of an element (ELKE) is not available for output in a direct-solution steady-state dynamic analysis.

The elastic strain energy density (SENER) is not available for output in a SIM-based steady-state dynamic analysis.

Whole model variables such as ALLIE (total strain energy) are available for direct-solution steady-state dynamic analysis by requesting energy output to the data, results, or output database files (see "Output to the data and results files," Section 4.1.2, and "Output to the output database," Section 4.1.3).

Input file template

```
*HEADING
 …
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamic step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
…
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, DIRECT
Data lines to specify frequency ranges and bias parameters
*BOUNDARY, REAL
Data lines to specify real (in-phase) boundary conditions
*BOUNDARY, IMAGINARY
Data lines to specify imaginary (out-of-phase) boundary conditions
*CLOAD, AMPLITUDE=loadamp
Data lines to specify sinusoidally varying, frequency-dependent, concentrated loads
*CLOAD and/or *DLOAD
Data lines to specify sinusoidally varying loads
…
*END STEP
```

6.3.5 NATURAL FREQUENCY EXTRACTION

Products: Abaqus/Standard Abaqus/CAE Abaqus/AMS

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Dynamic analysis procedures: overview," Section 6.3.1
- *FREQUENCY
- "Configuring a frequency procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The frequency extraction procedure:

- performs eigenvalue extraction to calculate the natural frequencies and the corresponding mode shapes of a system;
- will include initial stress and load stiffness effects due to preloads and initial conditions if geometric nonlinearity is accounted for in the base state, so that small vibrations of a preloaded structure can be modeled;
- will compute residual modes if requested;
- is a linear perturbation procedure;
- can be performed using the traditional Abaqus software architecture if appropriate, but the high-performance SIM architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1) is used as the default; and
- solves the eigenfrequency problem only for symmetric mass and stiffness matrices; the complex eigenfrequency solver must be used if unsymmetric contributions, such as the load stiffness, are needed.

Eigenvalue extraction

The eigenvalue problem for the natural frequencies of an undamped finite element model is

$$(-\omega^2 M^{MN} + K^{MN})\phi^N = 0,$$

where

M^{MN} is the mass matrix (which is symmetric and positive definite);

K^{MN} is the stiffness matrix (which includes initial stiffness effects if the base state included the effects of nonlinear geometry);

ϕ^N is the eigenvector (the mode of vibration); and

M and N are degrees of freedom.

When K^{MN} is positive definite, all eigenvalues are positive. Rigid body modes and instabilities cause K^{MN} to be indefinite. Rigid body modes produce zero eigenvalues. Instabilities produce negative eigenvalues and occur when you include initial stress effects. Abaqus/Standard solves the eigenfrequency problem only for symmetric matrices.

Selecting the eigenvalue extraction method

Abaqus/Standard provides three eigenvalue extraction methods:

- Lanczos
- Automatic multi-level substructuring (AMS), an add-on analysis capability for Abaqus/Standard
- Subspace iteration

In addition, you must consider the software architecture that will be used for the subsequent modal superposition procedures. The choice of architecture has minimal impact on the frequency extraction procedure, but the default SIM architecture offers significant performance improvements over the traditional architecture for subsequent mode-based steady-state or transient dynamic procedures (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1). The architecture that you use for the frequency extraction procedure is used for all subsequent mode-based linear dynamic procedures; you cannot switch architectures during an analysis. The software architectures used by the different eigensolvers are outlined in Table 6.3.5–1.

Table 6.3.5–1 Software architectures available with different eigensolvers.

Software Architecture	Eigensolver		
	Lanczos	AMS	Subspace Iteration
Traditional	√		√
SIM	√	√	√

The Lanczos solver with the SIM architecture is the default eigenvalue extraction method because it has the most general capabilities. However, the Lanczos method is generally slower than the AMS method. The increased speed of the AMS eigensolver is particularly evident when you require a large number of eigenmodes for a system with many degrees of freedom. However, the AMS method has the following limitations:

- All restrictions imposed on SIM-based linear dynamic procedures also apply to mode-based linear dynamic analyses based on mode shapes computed by the AMS eigensolver. See "Using the

SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1, for details.

- The AMS eigensolver does not compute composite modal damping factors.
- You cannot use the AMS eigensolver in an analysis that contains piezoelectric elements.
- You cannot request output to the results (`.fil`) file in an AMS frequency extraction step.
- The SIM-based architecture does not support the following capabilities: cyclic symmetry, fluid properties associated with an incident wave, transient dynamic procedures used for coupled structural-acoustic problems, symmetric model generation, and imperfections based on eigenmode data. It is recommended that you run random response analyses using the non-SIM architecture.
- At least one output request is required to run analyses using the SIM-based architecture.

If your model has many degrees of freedom and these limitations are acceptable, you should use the AMS eigensolver. Otherwise, you should use the Lanczos eigensolver. The Lanczos eigensolver and the subspace iteration method are described in "Eigenvalue extraction," Section 2.5.1 of the Abaqus Theory Guide.

Lanczos eigensolver

For the Lanczos method you need to provide the maximum frequency of interest or the number of eigenvalues required; Abaqus/Standard will determine a suitable block size (although you can override this choice, if needed). If you specify both the maximum frequency of interest and the number of eigenvalues required and the actual number of eigenvalues is underestimated, Abaqus/Standard will issue a corresponding warning message; the remaining eigenmodes can be found by restarting the frequency extraction.

You can also specify the minimum frequencies of interest; Abaqus/Standard will extract eigenvalues until either the requested number of eigenvalues has been extracted in the given range or all the frequencies in the given range have been extracted.

See "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1, for information on using the SIM architecture with the Lanczos eigensolver.

Input File Usage: *FREQUENCY, EIGENSOLVER=LANCZOS

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver**: **Lanczos**

Choosing a block size for the Lanczos method

In general, the block size for the Lanczos method should be as large as the largest expected multiplicity of eigenvalues (that is, the largest number of modes with the same frequency). A block size larger than 10 is not recommended. If the number of eigenvalues requested is n, the default block size is the minimum of (7, n). The choice of 7 for block size proves to be efficient for problems with rigid body modes. The number of block Lanczos steps within each Lanczos run is usually determined by Abaqus/Standard but can be changed by you. In general, if a particular type of eigenproblem converges slowly, providing more block Lanczos steps will reduce the analysis cost. On the other hand, if you know that a particular type of problem converges quickly, providing fewer block Lanczos steps will reduce the amount of in-core memory used. The default values are

Block size	Maximum number of block Lanczos steps
1	80
2	50
3	45
≥ 4	35

Automatic multi-level substructuring (AMS) eigensolver

For the AMS method you need only specify the maximum frequency of interest (the global frequency), and Abaqus/Standard will extract all the modes up to this frequency. You can also specify the minimum frequencies of interest and/or the number of requested modes. However, specifying these values will not affect the number of modes extracted by the eigensolver; it will affect only the number of modes that are stored for output or for a subsequent modal analysis.

The execution of the AMS eigensolver can be controlled by specifying three parameters: AMS_{cutoff_1}, AMS_{cutoff_2}, and AMS_{cutoff_3}. These three parameters multiplied by the maximum frequency of interest define three cutoff frequencies. AMS_{cutoff_1} (default value of 5) controls the cutoff frequency for substructure eigenproblems in the reduction phase, while AMS_{cutoff_2} and AMS_{cutoff_3} (default values of 1.7 and 1.1, respectively) control the cutoff frequencies used to define a starting subspace in the reduced eigensolution phase. Generally, increasing the value of AMS_{cutoff_2} and AMS_{cutoff_3} improves the accuracy of the results but may affect the performance of the analysis.

Requesting eigenvectors at all nodes

By default, the AMS eigensolver computes eigenvectors at every node of the model.

Input File Usage: *FREQUENCY, EIGENSOLVER=AMS

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver**: **AMS**

Requesting eigenvectors only at specified nodes

Alternatively, you can specify a node set, and eigenvectors will be computed and stored only at the nodes that belong to that node set. The node set that you specify must include all nodes at which loads are applied or output is requested in any subsequent modal analysis (this includes any restarted analysis). If element output is requested or element-based loading is applied, the nodes attached to the associated elements must also be included in this node set. Computing eigenvectors at only selected nodes improves performance and reduces the amount of stored data. Therefore, it is recommended that you use this option for large problems. Abaqus/Standard can automatically select all the nodes that need to be included in the node set. These nodes are

- nodes at which a concentrated load is applied in the following mode-based procedures,
- nodes at which output is requested in the eigenvalue extraction analysis or in the following mode-based procedures,

- nodes at which residual vectors are requested,
- nodes of elements at which a distributed load is applied,
- nodes of elements with frequency-dependent material properties, and
- nodes of elements at which output is requested in the eigenvalue extraction analysis or in the following mode-based procedures.

Input File Usage: Use the following option to specify a node set:

*FREQUENCY, EIGENSOLVER=AMS, NSET=*name*

Use the following option to allow Abaqus/Standard to select the nodes automatically:

*FREQUENCY, EIGENSOLVER=AMS, NSET

Abaqus/CAE Usage: You can only request eigenvectors at specific nodes by specifying a node set in Abaqus/CAE.

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver**: **AMS**: **Limit region of saved eigenvectors**, select node set

Controlling the AMS eigensolver

The AMS method consists of the following three phases:

- **Reduction phase:** In this phase Abaqus/Standard uses a multi-level substructuring technique to reduce the full system in a way that allows a very efficient eigensolution of the reduced system. The approach combines a sparse factorization based on a multi-level supernode elimination tree and a local eigensolution at each supernode.

  Starting from the lowest level supernodes, we use a Craig-Bampton substructure reduction technique to successively reduce the size of the system as we progress upward in the elimination tree. At each supernode a local eigensolution is obtained based on fixing the degrees of freedom connected to the next higher level supernode (these are the local retained or "fixed-interface" degrees of freedom). At the end of the reduction phase the full system has been reduced such that the reduced stiffness matrix is diagonal and the reduced mass matrix has unit diagonal values but contains off-diagonal blocks of nonzero values representing the coupling between the supernodes.

  The cost of the reduction phase depends on the system size and the number of eigenvalues extracted (the number of eigenvalues extracted is controlled indirectly by specifying the highest eigenfrequency desired). You can make trade-offs between cost and accuracy during the reduction phase through the AMS_{cutoff_1} parameter. This parameter multiplied by the highest eigenfrequency specified for the full model yields the highest eigenfrequency that is extracted in the local supernode eigensolutions. Increasing the value of AMS_{cutoff_1} increases the accuracy of the reduction since more local eigenmodes are retained. However, increasing the number of retained modes also increases the cost of the reduced eigensolution phase, which is discussed next.
- **Reduced eigensolution phase:** In this phase Abaqus/Standard computes the eigensolution of the reduced system that comes from the previous phase. Although the reduced system typically is two orders of magnitude smaller in size than the original system, generally it still is too large to solve directly. Thus, the system is further reduced mainly by truncating the retained eigenmodes

and then solved using a single subspace iteration step. The two AMS parameters, AMS_{cutoff_2} and AMS_{cutoff_3}, define a starting subspace of the subspace iteration step. The default values of these parameters are carefully chosen and provide accurate results in most cases. When a more accurate solution is needed, the recommended procedure is to increase both parameters proportionally from their respective default values.

- **Recovery phase:** In this phase the eigenvectors of the original system are recovered using eigenvectors of the reduced problem and local substructure modes. If you request recovery at specified nodes, the eigenvectors are computed only at those nodes.

Subspace iteration method

For the subspace iteration procedure you need only specify the number of eigenvalues required; Abaqus/Standard chooses a suitable number of vectors for the iteration. If the subspace iteration technique is requested, you can also specify the maximum frequency of interest; Abaqus/Standard extracts eigenvalues until either the requested number of eigenvalues has been extracted or the last frequency extracted exceeds the maximum frequency of interest.

Input File Usage: *FREQUENCY, EIGENSOLVER=SUBSPACE

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver**: **Subspace**

Structural-acoustic coupling

Structural-acoustic coupling affects the natural frequency response of systems. In Abaqus the AMS eigensolver and the Lanczos eigensolver can extract coupled modes to fully include this effect. The subspace eigensolver neglects the effect of coupling for the purpose of computing the modes and frequencies; the modes and frequencies are computed using natural boundary conditions at the structural-acoustic coupling surface. By default, the same is done for the AMS eigensolver; the coupling is projected onto the modal space and stored for later use.

Structural-acoustic coupling using the Lanczos eigensolver

If structural-acoustic coupling is present in the model and the Lanczos method is used, Abaqus/Standard extracts the coupled modes by default. Because these modes fully account for coupling, they represent the mathematically optimal basis for subsequent modal procedures. The effect is most noticeable in strongly coupled systems such as steel shells and water. However, coupled structural-acoustic modes cannot be used in subsequent random response or response spectrum analyses. You can define the coupling using either acoustic-structural interaction elements (see "Acoustic interface elements," Section 32.13.1) or the surface-based tie constraint (see "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1). It is possible to ignore coupling when extracting acoustic and structural modes; in this case the coupling boundary is treated as traction-free on the structural side and rigid on the acoustic side.

For frequency extractions that use the Lanczos eigensolver based on the SIM architecture, it is also possible to project structural-acoustic coupling operators onto the subspace of eigenvectors. The modes are computed using traction-free boundary conditions on the structural side of the coupling boundary and rigid boundary conditions on the acoustic side. Structural-acoustic coupling operators

(see "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1) are projected by default onto the subspace of eigenvectors. Contributions to these global operators, which come from surface-based tie constraints defined between structural and acoustic surfaces, are assembled into global matrices that are projected onto the mode shapes and used in subsequent SIM-based modal dynamic procedures.

Input File Usage: Use the following option to account for structural-acoustic coupling during the frequency extraction:

*FREQUENCY, EIGENSOLVER=LANCZOS,
ACOUSTIC COUPLING=ON (default)

Use the following option to project structural-acoustic coupling on the uncoupled eigenmodes during the frequency extraction:

*FREQUENCY, EIGENSOLVER=LANCZOS, SIM,
ACOUSTIC COUPLING=PROJECTION (only if the Lanczos eigensolver is based on the SIM architecture)

Use the following option to ignore structural-acoustic coupling during the frequency extraction:

*FREQUENCY, EIGENSOLVER=LANCZOS,
ACOUSTIC COUPLING=OFF

Abaqus/CAE Usage: Use the following option to account for structural-acoustic coupling during the frequency extraction:

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: Lanczos**, toggle on **Include acoustic-structural coupling where applicable**

Use the following option to project structural-acoustic coupling on the uncoupled eigenmodes during the frequency extraction:

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: Lanczos**, toggle on **Use SIM-based linear dynamics procedures**, toggle on **Project acoustic-structural coupling where applicable**

Use the following option to ignore structural-acoustic coupling during the frequency extraction:

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: Lanczos**, toggle off **Include acoustic-structural coupling where applicable**

Structural-acoustic coupling using the AMS eigensolver

For frequency extractions that use the AMS eigensolver, the modes are computed by default using traction-free boundary conditions on the structural side of the coupling boundary and rigid boundary conditions on the acoustic side. Structural-acoustic coupling operators (see "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1) are projected by default onto the subspace of eigenvectors. Contributions to these global operators, which come from surface-based tie constraints

defined between structural and acoustic surfaces, are assembled into global matrices that are projected onto the mode shapes and used in subsequent SIM-based modal dynamic procedures.

For frequency extractions that use the AMS eigensolver, Abaqus/Standard can also extract the coupled modes. Because these modes fully account for coupling, they represent the mathematically optimal basis for subsequent modal procedures. The effect is most noticeable in strongly coupled systems such as steel shells and water. However, extracting the coupled structural-acoustic modes using the AMS eigensolver is computationally more expensive than the default option, where the coupling operators are projected onto the subspace of uncoupled eigenvectors.

User-defined acoustic-structural interaction elements (see "Acoustic interface elements," Section 32.13.1) cannot be used in an AMS eigenvalue extraction analysis.

Input File Usage: Use the following option to project structural-acoustic coupling operators onto the subspace of eigenvectors:

```
*FREQUENCY, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=PROJECTION (default)
```

Use the following option to disable the projection of structural-acoustic coupling operators:

```
*FREQUENCY, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=OFF
```

Use the following option to extract coupled structural-acoustic eigenmodes:

```
*FREQUENCY, EIGENSOLVER=AMS,
ACOUSTIC COUPLING=ON
```

Abaqus/CAE Usage: Use the following option to project structural-acoustic coupling operators onto the subspace of eigenvectors:

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, toggle on **Project acoustic-structural coupling where applicable**

Use the following option to disable the projection of structural-acoustic coupling operators:

Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, toggle off **Project acoustic-structural coupling where applicable**

Requesting the coupled structural-acoustic modes during the AMS eigenanalysis is not supported in Abaqus/CAE.

Specifying a frequency range for the acoustic modes

Because structural-acoustic coupling can be ignored during the AMS and SIM-based Lanczos eigenanalysis, the computed resonances will, in principle, be higher than those of the fully coupled system. This may be understood as a consequence of neglecting the mass of the fluid in the structural phase and vice versa. For the common metal and air case, the structural resonances may be relatively unaffected; however, some acoustic modes that are significant in the coupled response may be omitted due to the air's upward frequency shift during eigenanalysis. Therefore, Abaqus allows you to specify

a multiplier, so that the maximum acoustic frequency in the analysis is taken to be higher than the structural maximum.

Input File Usage: Use either of the following options:

```
*FREQUENCY, EIGENSOLVER=AMS
, , , , , , acoustic range factor
```

or

```
*FREQUENCY, EIGENSOLVER=LANCZOS, SIM,
ACOUSTIC COUPLING=PROJECTION
, , , , , , acoustic range factor
```

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Basic**: **Eigensolver: AMS**, **Acoustic range factor**: *acoustic range factor*

Specifying a frequency range for the acoustic modes during Lanczos eigenanalysis is not supported in Abaqus/CAE.

Effects of fluid motion on natural frequency analysis of acoustic systems

To extract natural frequencies from an acoustic-only or coupled structural-acoustic system in which fluid motion is prescribed using an acoustic flow velocity, either the Lanczos method or the complex eigenvalue extraction procedure can be used. In the former case Abaqus extracts real-only eigenvalues and considers the fluid motion's effects only on the acoustic stiffness matrix. Thus, these results are of primary interest as a basis for subsequent linear perturbation procedures. When the complex eigenvalue extraction procedure is used, the fluid motion effects are included in their entirety; that is, the acoustic stiffness and damping matrices are included in the analysis.

Frequency shift

For the Lanczos and subspace iteration eigensolvers you can specify a positive or negative shifted squared frequency, S. This feature is useful when a particular frequency is of concern or when the natural frequencies of an unrestrained structure or a structure that uses secondary base motions (large mass approach) are needed. In the latter case a shift from zero (the frequency of the rigid body modes) will avoid singularity problems or round-off errors for the large mass approach; a negative frequency shift is normally used. The default is no shift.

If the Lanczos eigensolver is in use and the user-specified shift is outside the requested frequency range, the shift will be adjusted automatically to a value close to the requested range.

Normalization

For the Lanczos and subspace iteration eigensolvers both displacement and mass eigenvector normalization are available. If the non-SIM architecture is used, displacement normalization is the default. Mass normalization is the only option available for SIM-based natural frequency extraction.

The choice of eigenvector normalization type has no influence on the results of subsequent modal dynamic steps (see "Linear analysis of a rod under dynamic loading," Section 1.4.9 of the Abaqus

Benchmarks Guide). The normalization type determines only the manner in which the eigenvectors are represented.

In addition to extracting the natural frequencies and mode shapes, the Lanczos and subspace iteration eigensolvers automatically calculate the generalized mass, the participation factor, the effective mass, and the composite modal damping for each mode; therefore, these variables are available for use in subsequent linear dynamic analyses. The AMS eigensolver computes the generalized mass and the participation factor.

Displacement normalization

If displacement normalization is selected, the eigenvectors are normalized so that the largest displacement entry in each vector is unity. If the displacements are negligible, as in a torsional mode, the eigenvectors are normalized so that the largest rotation entry in each vector is unity. In a coupled acoustic-structural extraction, if the displacements and rotations in a particular eigenvector are small when compared to the acoustic pressures, the eigenvector is normalized so that the largest acoustic pressure in the eigenvector is unity. The normalization is done before the recovery of dependent degrees of freedom that have been previously eliminated with multi-point constraints or equation constraints. Therefore, it is possible that such degrees of freedom may have values greater than unity.

Input File Usage: *FREQUENCY, NORMALIZATION=DISPLACEMENT, SIM=NO

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Other**: **Normalize eigenvectors by**: **Displacement**

Mass normalization

Alternatively, the eigenvectors can be normalized so that the generalized mass for each vector is unity.

The "generalized mass" associated with mode α is

$$m_\alpha = \phi_\alpha^N M^{NM} \phi_\alpha^M \quad \text{(no sum on } \alpha\text{)},$$

where M^{NM} is the structure's mass matrix and ϕ_α^N is the eigenvector for mode α. The superscripts N and M refer to degrees of freedom of the finite element model.

If the eigenvectors are normalized with respect to mass, all the eigenvectors are scaled so that m_α=1. For coupled acoustic-structural analyses, an acoustic contribution fraction to the generalized mass is computed as well.

Input File Usage: *FREQUENCY, NORMALIZATION=MASS

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Other**: **Normalize eigenvectors by**: **Mass**

Modal participation factors

The participation factor for mode α in direction i, $\Gamma_{\alpha i}$, is a variable that indicates how strongly motion in the global x-, y-, or z-direction or rigid body rotation about one of these axes is represented in the

eigenvector of that mode. The six possible rigid body motions are indicated by $i = 1, 2, \ldots, 6$. The participation factor is defined as

$$\Gamma_{\alpha i} = \frac{1}{m_\alpha} \phi_\alpha^N M^{NM} T_i^M \quad \text{(no sum on } \alpha\text{)},$$

where T_i^N defines the magnitude of the rigid body response of degree of freedom N in the model to imposed rigid body motion (displacement or infinitesimal rotation) of type i. For example, at a node with three displacement and three rotation components, T_i^N is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & (z-z_0) & -(y-y_0) \\ 0 & 1 & 0 & -(z-z_0) & 0 & (x-x_0) \\ 0 & 0 & 1 & (y-y_0) & -(x-x_0) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \\ \hat{e}_4 \\ \hat{e}_5 \\ \hat{e}_6 \end{Bmatrix},$$

where $\hat{e}_i$ is unity and all other $\hat{e}_p$ are zero; x, y, and z are the coordinates of the node; and x_0, y_0, and z_0 represent the coordinates of the center of rotation. The participation factors are, thus, defined for the translational degrees of freedom and for rotation around the center of rotation. For coupled acoustic-structural eigenfrequency analysis, an additional acoustic participation factor is computed as outlined in "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide.

Modal effective mass

The effective mass for mode α associated with kinematic direction i ($i = 1, 2, \ldots, 6$) is defined as

$$m_{\alpha i}^{\text{eff}} = \left(\Gamma_{\alpha i}\right)^2 m_\alpha \quad \text{(no sum on } \alpha\text{)}.$$

If the effective masses of all modes are added in any global translational direction, the sum should give the total mass of the model (except for mass at kinematically restrained degrees of freedom). Thus, if the effective masses of the modes used in the analysis add up to a value that is significantly less than the model's total mass, this result suggests that modes that have significant participation in a certain excitation direction have not been extracted.

For coupled acoustic-structural eigenfrequency analysis, an additional acoustic effective mass is computed as outlined in "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide.

Composite modal damping

Composite modal damping allows you to define a damping factor for each material or element in the model as a fraction of critical damping. These factors are then combined into a damping factor for each mode as weighted averages of the mass matrix associated with each material or element; when using the SIM architecture, you can also include the weighted averages of the stiffness matrix. For more information, see "Defining composite modal damping" in "Dynamic analysis procedures: overview," Section 6.3.1.

Obtaining residual modes for use in mode-based procedures

Several analysis types in Abaqus/Standard are based on the eigenmodes and eigenvalues of the system. For example, in a mode-based steady-state dynamic analysis the mass and stiffness matrices and load vector of the physical system are projected onto a set of eigenmodes resulting in a diagonal system in terms of modal amplitudes (or generalized degrees of freedom). The solution to the physical system is obtained by scaling each eigenmode by its corresponding modal amplitude and superimposing the results (for more information, see "Linear dynamic analysis using modal superposition," Section 2.5.3 of the Abaqus Theory Guide).

Due to cost, usually only a small subset of the total possible eigenmodes of the system are extracted, with the subset consisting of eigenmodes corresponding to eigenfrequencies that are close to the excitation frequency. Since excitation frequencies typically fall in the range of the lower modes, it is usually the higher frequency modes that are left out. Depending on the nature of the loading, the accuracy of the modal solution may suffer if too few higher frequency modes are used. Thus, a trade-off exists between accuracy and cost. To minimize the number of modes required for a sufficient degree of accuracy, the set of eigenmodes used in the projection and superposition can be augmented with additional modes known as *residual modes*. The residual modes help correct for errors introduced by mode truncation. In Abaqus/Standard a residual mode, $\boldsymbol{R}$, represents the static response of the structure subjected to a nominal (or unit) load, $\boldsymbol{P}$, corresponding to the actual load that will be used in the mode-based analysis orthogonalized against the extracted eigenmodes,

$$R^N = (\delta^{NJ} - \phi_\alpha^N \frac{1}{m_\alpha} \phi_\alpha^I M^{IJ})(K^{-1})^{JK} P^K,$$

followed by an orthogonalization of the residual modes against each other.

This orthogonalization is required to retain the orthogonality properties of the modes (residual and eigen) with respect to mass and stiffness. As a consequence of the mass and stiffness matrices being available, the orthogonalization can be done efficiently during the frequency extraction. Hence, if you wish to include residual modes in subsequent mode-based procedures, you must activate the residual mode calculations in the frequency extraction step. If the static responses are linearly dependent on each other or on the extracted eigenmodes, Abaqus/Standard automatically eliminates the redundant responses for the purpose of computing the residual modes.

For the Lanczos eigensolver you must ensure that the static perturbation response of the load that will be applied in the subsequent mode-based analysis (i.e., $K^{-1}P$) is available by specifying that load in a static perturbation step immediately preceding the frequency extraction step. If multiple load cases are specified in this static perturbation analysis, one residual mode is calculated for each load case; otherwise, it is assumed that all loads are part of a single load case, and only one residual mode will be calculated. When residual modes are requested, the boundary conditions applied in the frequency extraction step must match those applied in the preceding static perturbation step. In addition, in the immediately preceding static perturbation step Abaqus/Standard requires that (1) if multiple load cases are used, the boundary conditions applied in each load case must be identical, and (2) the boundary condition magnitudes are zero. When generating dynamic substructures (see "Generating a reduced structural damping matrix for a substructure" in "Defining substructures," Section 10.1.2), residual modes usually

will provide the most benefit if the loading patterns defined in each of the load cases in the preceding static perturbation step match the loading patterns defined under the corresponding substructure load cases in the substructure generation step.

If you use the AMS eigensolver, you do not need to specify the loads in a preceding static perturbation step. Residual modes are computed at all degrees of freedom at which a concentrated load is applied in the following mode-based procedures. You can request additional residual modes by specifying degrees of freedom. One residual mode is computed for every requested degree of freedom.

As an outcome of the orthogonalization process, a pseudo-eigenvalue corresponding to each residual mode, α, is computed and given by

$$\frac{[R]_\alpha^T[K][R]_\alpha}{[R]_\alpha^T[M][R]_\alpha} \quad \text{(no sum on } \alpha\text{)}.$$

Henceforth, and in other Abaqus/Standard documentation, the term eigenvalue is used generally to refer to actual eigenvalues and pseudo-eigenvalues. All data (e.g., participation factors, etc.; see "Output") associated with the modes (eigenmodes and residual modes) are ordered by increasing eigenvalue. Therefore, both eigenmodes and residual modes are assigned mode numbers. In the printed output file Abaqus/Standard clearly identifies which modes are eigenmodes and which modes are residual modes so that you can easily distinguish between them. By default, if you activate residual modes, all the calculated eigenmodes and residual modes will be used in subsequent mode-based procedures, unless:

- You choose to obtain a new set of eigenmodes and residual modes in a new frequency extraction step.
- You choose to select a subset of the available eigenmodes and residual modes in the mode-based procedure (selection of modes is described in each of the mode-based analysis type sections).

Residual modes cannot be calculated if the cyclic symmetric modeling capability is used. In addition, the Lanczos or AMS eigensolver must be used if you wish to activate residual mode calculations.

Input File Usage: *FREQUENCY, RESIDUAL MODES

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Basic**: **Include residual modes**

Evaluating frequency-dependent material properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in the frequency extraction procedure. This evaluation is necessary because the stiffness cannot be modified during the eigenvalue extraction procedure. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness associated with frequency-dependent springs and dashpots at zero frequency and does not consider the stiffness contributions from frequency domain viscoelasticity. If you do specify a frequency, only the real part of the stiffness contributions from frequency domain viscoelasticity is considered.

Evaluating the properties at a specified frequency is particularly useful in analyses in which the eigenfrequency extraction step is followed by a subspace projection steady-state dynamic step (see "Subspace-based steady-state dynamic analysis," Section 6.3.9). In these analyses the eigenmodes extracted in the frequency extraction step are used as global basis functions to compute the steady-state

dynamic response of a system subjected to harmonic excitation at a number of output frequencies. The accuracy of the results in the subspace projection steady-state dynamic step is improved if you choose to evaluate the material properties at a frequency in the vicinity of the center of the range spanned by the frequencies specified for the steady-state dynamic step.

Input File Usage: *FREQUENCY, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: **Step→Create**: **Frequency**: **Other**: **Evaluate dependent properties at frequency**

Initial conditions

If the frequency extraction procedure is the first step in an analysis, the initial conditions form the *base state* for the procedure (except for initial stresses, which cannot be included in the frequency extraction if it is the first step). Otherwise, the base state is the current state of the model at the end of the last general analysis step ("General and linear perturbation procedures," Section 6.1.3). Initial stress stiffness effects (specified either through defining initial stresses or through loading in a general analysis step) will be included in the eigenvalue extraction only if geometric nonlinearity is considered in a general analysis procedure prior to the frequency extraction procedure.

If initial stresses must be included in the frequency extraction and there is not a general nonlinear step prior to the frequency extraction step, a "dummy" static step—which includes geometric nonlinearity and which maintains the initial stresses with appropriate boundary conditions and loads—must be included before the frequency extraction step.

"Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions.

Boundary conditions

Nonzero magnitudes of boundary conditions in a frequency extraction step will be ignored; the degrees of freedom specified will be fixed ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Boundary conditions defined in a frequency extraction step will not be used in subsequent general analysis steps (unless they are respecified).

In a frequency extraction step involving piezoelectric elements, the electric potential degree of freedom must be constrained at least at one node to remove numerical singularities arising from the dielectric part of the element operator.

Defining primary and secondary bases for modal superposition procedures

If displacements or rotations are to be prescribed in subsequent dynamic modal superposition procedures, boundary conditions must be applied in the frequency extraction step; these degrees of freedom are grouped into "bases." The bases are then used for prescribing motion in the modal superposition procedure—see "Transient modal dynamic analysis," Section 6.3.7.

Boundary conditions defined in the frequency extraction step supersede boundary conditions defined in previous steps. Hence, degrees of freedom that were fixed prior to the frequency extraction step will

be associated with a specific base if they are redefined with reference to such a base in the frequency extraction step.

The primary base

By default, all degrees of freedom listed for a boundary condition will be assigned to an unnamed "primary" base. If the same motion will be prescribed at all fixed points, the boundary condition is defined only once; and all prescribed degrees of freedom belong to the primary base.

Unless removed in the frequency extraction step, boundary conditions from the last general analysis step become fixed boundary conditions for the frequency step and belong to the primary base.

If all rigid body motions are not suppressed by the boundary conditions that make up the primary base, you must apply a suitable frequency shift to avoid numerical problems.

Input File Usage: *BOUNDARY

The *BOUNDARY option without the BASE NAME parameter can appear only once in a frequency extraction step.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**

Secondary bases

If the modal superposition procedure will have more than one independent base motion, the driven nodes must be grouped together into "secondary" bases in addition to the primary base. The secondary bases must be named. (See "Base motions in modal-based procedures," Section 2.5.9 of the Abaqus Theory Guide.) Secondary bases are used only in modal dynamic and steady-state dynamic (not direct) procedures.

The degrees of freedom associated with secondary bases are not suppressed; instead, a "big" mass is added to each of them. To provide six digits of numerical accuracy, Abaqus/Standard sets each "big" mass equal to 10^6 times the total mass of the structure and each "big" rotary inertia equal to 10^6 times the total moment of inertia of the structure. Hence, an artificial low frequency mode is introduced for every degree of freedom in a secondary base. To keep the requested range of frequencies unchanged, Abaqus/Standard automatically increases the number of eigenvalues extracted. Consequently, the cost of the eigenvalue extraction step will increase as more degrees of freedom are included in the secondary bases. To reduce the analysis cost, keep the number of degrees of freedom associated with secondary bases to a minimum. This can sometimes be done by reducing several secondary bases that all have the same prescribed motion to a single node by using BEAM type MPCs ("General multi-point constraints," Section 35.2.2).

For the Lanczos and subspace iteration methods a negative shift must be used with either the rigid body modes or secondary bases.

The "big" masses are not included in the model statistics, and the total mass of the structure and the printed messages about masses and inertia for the entire model are not affected. However, the presence of the masses will be noticeable in the output tables printed for the eigenvalue extraction step, as well as in the information for the generalized masses and effective masses. See "Double cantilever subjected to multiple base motions," Section 1.4.12 of the Abaqus Benchmarks Guide, for an example of the use of the base motion feature.

More than one secondary base can be defined by repeating the boundary condition definition and assigning different base names.

Input File Usage: *BOUNDARY, BASE NAME=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *frequency_step*; **Category: Mechanical**; **Types for Selected Step: Secondary base**; **Constrained degrees-of-freedom**: **Region**: *select region*, **U1**, **U2**, **U3**, **UR1**, **UR2**, and/or **UR3**

Controlling damping for low frequency modes

You can control damping of the low frequency modes (rigid body modes and low frequency modes with secondary base motions) in transient modal analyses. For more information, see "Controlling damping" in "Transient modal dynamic analysis," Section 6.3.7.

Loads

Applied loads ("Applying loads: overview," Section 34.4.1) are ignored during a frequency extraction analysis. If loads were applied in a previous general analysis step and geometric nonlinearity was considered for that prior step, the load stiffness determined at the end of the previous general analysis step is included in the eigenvalue extraction ("General and linear perturbation procedures," Section 6.1.3).

Predefined fields

Predefined fields cannot be prescribed during natural frequency extraction.

Material options

The density of the material must be defined ("Density," Section 21.2.1). The following material properties are not active during a frequency extraction: plasticity and other inelastic effects, rate-dependent material properties, thermal properties, mass diffusion properties, electrical properties (although piezoelectric materials are active), and pore fluid flow properties—see "General and linear perturbation procedures," Section 6.1.3.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement or acoustic elements in Abaqus/Standard (including those with temperature, pressure, or electrical degrees of freedom) can be used in a frequency extraction procedure.

Output

The eigenvalues (EIGVAL), eigenfrequencies in cycles/time (EIGFREQ), generalized masses (GM), composite modal damping factors (CD), participation factors for displacement degrees of freedom 1–6 (PF1–PF6) and acoustic pressure (PF7), and modal effective masses for displacement degrees of freedom

1–6 (EM1–EM6) and acoustic pressure (EM7) are written automatically to the output database as history data. Output variables such as stress, strain, and displacement (which represent mode shapes) are also available for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values.

The eigenvalues and corresponding frequencies (in both radians/time and cycles/time) will also be automatically listed in the printed output file, along with the generalized masses, composite modal damping factors, participation factors, and modal effective masses.

The only energy density available in eigenvalue extraction procedures is the elastic strain energy density, SENER. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

The AMS eigensolver does not compute composite modal damping factors, participation factors, or modal effective masses. In addition, you cannot request output to the results (`.fil`) file.

You can restrict output to the results, data, and output database files by selecting the modes for which output is desired (see "Output to the data and results files," Section 4.1.2, and "Output to the output database," Section 4.1.3).

Input File Usage: Use one of the following options:

*EL FILE, MODE, LAST MODE
*EL PRINT, MODE, LAST MODE
*OUTPUT, MODE LIST

Abaqus/CAE Usage: Step module: **Output→Field Output Requests→Create**: **Frequency**: **Specify modes**

Input file template

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
**
*STEP (,NLGEOM)
If NLGEOM is used, initial stress and preload stiffness effects
will be included in the frequency extraction step
*STATIC
...
*CLOAD and/or *DLOAD
Data lines to specify loads
*TEMPERATURE and/or *FIELD
Data lines to specify values of predefined fields
*BOUNDARY
Data lines to specify zero-valued or nonzero boundary conditions
```

```
*END STEP
**
*STEP, PERTURBATION
*STATIC
…
*LOAD CASE, NAME=load case name
Keywords and data lines to define loading for this load case
*END LOAD CASE
…
*END STEP
**
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS, RESIDUAL MODES
Data line to control eigenvalue extraction
*BOUNDARY

*BOUNDARY, BASE NAME=name
Data lines to assign degrees of freedom to a secondary base
*END STEP
```

6.3.6 COMPLEX EIGENVALUE EXTRACTION

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- *COMPLEX FREQUENCY
- "Configuring a complex frequency procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The complex eigenvalue extraction procedure:

- performs eigenvalue extraction to calculate the complex eigenvalues and the corresponding complex mode shapes of a system;
- is a linear perturbation procedure;
- requires that an eigenfrequency extraction procedure ("Natural frequency extraction," Section 6.3.5) be performed prior to the complex eigenvalue extraction;
- can use the high-performance SIM software architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1);
- will include initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the base state step definition ("General and linear perturbation procedures," Section 6.1.3);
- can include friction, damping, and unsymmetric load stiffness contributions;
- can include unsymmetric damping and stiffness contributions in acoustic finite elements due to underlying mean flow ("Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1); and
- cannot be used in a model defined as a cyclic symmetric structure ("Analysis of models that exhibit cyclic symmetry," Section 10.4.3).

Complex eigenvalue extraction

The complex eigenvalue extraction procedure uses a projection method to extract the complex eigenvalues of the current system. The eigenvalue problem of the finite element model is formulated in the following manner:

$$(\mu^2 M^{MN} + \mu C^{MN} + K^{MN})\phi_R^N = 0,$$

where

M^{MN} is the mass matrix (which is symmetric and, in general, is semi-positive definite);

C^{MN} is the damping matrix;

K^{MN} is the stiffness matrix (which can include initial stress stiffness and friction effects and, therefore, in general is unsymmetric);

μ is the complex eigenvalue;

ϕ_R^N is the right complex eigenvector;

ϕ_L^N is the left complex eigenvector and is defined as follows:

$$(\phi_L^N)^*(\mu^2 M^{MN} + \mu C^{MN} + K^{MN}) = 0;$$

where $(\phi_L^N)^*$ is a transpose conjugate left eigenvector

M and N are degrees of freedom.

The complex eigenvalue extraction procedure in Abaqus/Standard uses a subspace projection method; thus, the eigenmodes of the undamped system with the symmetrized stiffness matrix must be extracted using the eigenfrequency extraction procedure prior to the complex eigenvalue extraction step. By default, the entire subspace is used as the base vector; this subspace can be reduced as described below. Abaqus/Standard always computes all the complex eigenmodes available in the projection subspace (taking into account any user-specified modifications to the subspace). The user-specified number of requested eigenmodes and frequency range for the complex eigenvalue extraction procedure do not influence the number of computed complex eigenmodes. It determines only the number of reported modes, which cannot be higher than the dimension of the projected subspace. To modify the number of computed eigenmodes, reduce the projection subspace as described below or change the number of eigenmodes extracted in the prior natural frequency extraction step accordingly. If you do not specify the number of requested complex modes or the frequency range, all the computed modes will be reported.

To take into account the unsymmetric effects, the unsymmetric matrix solution and storage scheme is used automatically for a complex eigenvalue extraction step. The unsymmetric effects will be disregarded if you specify that the symmetric solution and storage scheme should be used (see "Defining an analysis," Section 6.1.2).

Input File Usage: *COMPLEX FREQUENCY
number of complex eigenmodes, frequency_min, frequency_max

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Number of eigenvalues requested: All** or **Value**, **Minimum frequency of interest (cycles/time):** *value*, **Maximum frequency of interest (cycles/time):** *value*

Shift point

You can specify a shift point, S, in cycles per time, for the complex eigenvalue extraction procedure ($S \geq 0$). Abaqus/Standard reports the complex eigenmodes, μ_n, in order of increasing $|Im(\mu_n) - S|$ so

that the modes with the imaginary part closest to a given shift point are reported first. This feature is useful when a particular frequency range is of concern. The default is no shift.

Input File Usage: *COMPLEX FREQUENCY
, , , S

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Frequency shift (cycles/time):** S

Normalization

For complex eigenvalue extraction analysis both displacement and modal complex eigenvector normalization are available. Displacement normalization is the default in SIM-based analysis. Modal normalization is the only option available if the SIM-based architecture is not used.

If displacement normalization is selected, the complex eigenvectors are normalized so that the largest value in each vector is unity and the imaginary part is zero. If modal normalization is selected, only the complex eigenvectors of the projected system (GU) are normalized using the displacement method and no normalization of the complex eigenvectors in the finite element subspace is performed. For large eigenproblems the displacement normalization can become time consuming; therefore, modal normalization is recommended.

Input File Usage: Use the following option to select displacement normalization (available only if the SIM-based architecture is used):

*COMPLEX FREQUENCY, NORMALIZATION=DISPLACEMENT

Use the following option to select modal normalization (the only option if the SIM-based architecture is not used):

*COMPLEX FREQUENCY, NORMALIZATION=MODAL

Abaqus/CAE Usage: You cannot select the normalization method of the complex eigenvectors in Abaqus/CAE; the default method is used.

Selecting the eigenmodes on which to project

You can select eigenmodes of the undamped system with the symmetrized stiffness matrix on which the subspace projection will be performed. You can select them by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the eigenmodes that belong to specified frequency ranges. If you do not select the eigenmodes, all modes extracted in the prior eigenfrequency extraction step are used in the modal superposition.

Input File Usage: Use one of the following options to select the eigenmodes by specifying mode numbers:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS
*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to define the eigenmodes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the eigenmodes in Abaqus/CAE; all modes extracted are used in the subspace projection.

Evaluating frequency-dependent material properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in the complex eigenvalue extraction procedure. This evaluation is necessary because the operators cannot be modified during the eigenvalue extraction procedure. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness and damping associated with frequency-dependent springs and dashpots at zero frequency and does not consider the stiffness and damping contributions from frequency-domain viscoelasticity. If you do specify a frequency, the stiffness and damping contributions from frequency-domain viscoelasticity are considered.

Input File Usage: *COMPLEX FREQUENCY, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: **Create Step**: **Complex Frequency**: **Other**: **Evaluate dependent properties at frequency:** *value*

Right and left complex eigenvectors

For complex eigenvalue extraction analysis right or left complex eigenvectors can be requested. By default, the right eigenvectors are extracted. The left eigenvectors are available only in analyses that are based on the SIM architecture. You can extract both right and left complex eigenvectors in the same analysis, but they must be requested in separate steps. You should select modal normalization of the complex eigenvectors if you want to extract both the right and left eigenvectors.

Input File Usage: Use the following option to extract the right complex eigenvectors:

*COMPLEX FREQUENCY, RIGHT EIGENVECTORS (default, the only option if the SIM architecture is not used)

Use the following option to extract the left complex eigenvectors:

*COMPLEX FREQUENCY, LEFT EIGENVECTORS (only if the SIM architecture is used)

Abaqus/CAE Usage: Only the right complex eigenvectors are extracted in Abaqus/CAE.

Contact conditions with sliding friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom will not be constrained and the effect of friction will result in an unsymmetric contribution to the stiffness matrix. At other nodes in contact the tangential degrees of freedom will be constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing and is also known as "negative damping." For more details, see "Coulomb friction," Section 5.2.3 of the Abaqus Theory Guide. The complex eigensolver allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: *COMPLEX FREQUENCY, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Complex frequency**: **Include friction-induced damping effects**

Damping

In complex eigenvalue extraction analysis damping can be defined by dashpots (see "Dashpots," Section 32.2.1), by "Rayleigh" damping associated with materials and elements (see "Material damping," Section 26.1.1), and by quiet boundaries on infinite elements or acoustic elements. In addition, as described in "Contact conditions with sliding friction" above, friction-induced damping can be included.

Structural damping, damping contributions from frequency-domain viscoelasticity, and all types of modal damping (except composite modal damping) are supported in complex eigenvalue extraction using the high-performance SIM architecture.

Prescribing motion, transport velocity, and acoustic flow velocity

Motion, transport velocity, and acoustic flow velocity affect complex frequency analyses. Motion and transport velocity must be specified in a preceding steady-state transport general step, and their effects are included in the complex frequency step. The acoustic flow velocity has no effect in steady-state transport steps, and acoustic flow velocities specified in a steady-state transport step are not propagated to perturbation steps. The acoustic flow velocity must be specified in each linear perturbation step where it is desired.

Initial conditions

Initial conditions cannot be specified for complex eigenvalue extraction.

Boundary conditions

Boundary conditions cannot be defined during complex eigenvalue extraction. The boundary conditions will be the same as in the prior natural frequency extraction analysis.

Loads

Applied loads (“Applying loads: overview,” Section 34.4.1) are ignored during a complex eigenvalue extraction. If loads were applied in a previous general analysis step in which nonlinear geometric effects were included, the load stiffness determined at the end of the previous general analysis step is included in the complex eigenvalue extraction (see “General and linear perturbation procedures,” Section 6.1.3).

Coriolis distributed loading adds an unsymmetric contribution to the damping operator, which is currently accounted for only in solid and truss elements.

Predefined fields

Predefined fields cannot be prescribed during complex eigenvalue extraction.

Material options

The density of the material must be defined (see “Density,” Section 21.2.1). The following material properties are not active during complex eigenvalue extraction:

- plasticity and other inelastic effects;
- rate-dependent material properties, excluding friction, which can be rate dependent if the velocity differential on the contact interface exists;
- thermal properties;
- mass diffusion properties;
- electrical properties (although piezoelectric materials are active); and
- pore fluid flow properties.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature or pressure degrees of freedom) can be used in complex eigenvalue extraction.

Output

The real (EIGREAL) and imaginary (EIGIMAG) parts of the eigenvalues, (α and ω); frequencies in cycles/time (EIGFREQ); and effective damping ratios (DAMPRATIO = $-2 * \alpha/|\omega|$) are written automatically to the data (**.dat**) file and to the output database (**.odb**) file as history data. In addition, you can request that the generalized displacements (GU), which are the modes of the projected system, be written to the output database file (see “Output to the output database,” Section 4.1.3). Output variables such as stress, strain, and displacement (which represent mode shapes) are also available for each eigenvalue; these quantities are perturbation values and represent mode shapes, not absolute values.

The only energy density available in eigenvalue extraction procedures is the elastic strain energy density, SENER. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

You can restrict output to the data file and output database file by selecting the modes for which output is desired (see "Output to the data and results files," Section 4.1.2) or "Output to the output database," Section 4.1.3). Output to the results (**.fil**) file is not available for the complex eigenvalue extraction procedure.

Setting the cutoff value for complex eigenmodes

You can also set the cutoff value for complex eigenmodes, so only complex modes with the real part of the eigenvalue higher than the cutoff value are written to the output database file. The default cutoff value is 0.0. If the cutoff value is not set, all complex modes are output.

Input File Usage: Use one of the following options to select complex eigenmodes for output:

∗COMPLEX FREQUENCY, UNSTABLE MODES ONLY
∗COMPLEX FREQUENCY, UNSTABLE MODES ONLY=*value*

The SIM architecture

The complex eigenvalue extraction analysis can be performed using the SIM architecture. The advantages of performing the complex eigenvalue extraction procedure using the SIM architecture are as follows:

- structural damping, including damping defined with viscoelastic material, is taken into account;
- modal damping can be specified;
- matrices representing the stiffness, mass, and damping can be defined (both symmetric and unsymmetric matrices are supported); and
- the AMS eigensolver can be used to generate the projection subspace for the complex eigenvalue extraction.

When the AMS eigensolver is used for computing the projection subspace, you should increase the accuracy of the AMS eigensolution by increasing the values of the AMS parameters and by increasing the highest frequency of interest.

Input file template

```
*HEADING
...
*SURFACE INTERACTION
*FRICTION
Specify zero friction coefficient
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS
Data lines to specify initial conditions
```

```
**
*STEP (,NLGEOM)
```
If NLGEOM is used, initial stress and preload stiffness effects will be included in the eigenvalue extraction steps
```
*STATIC
…
*CLOAD and/or *DLOAD
```
Data lines to specify loads
```
*TEMPERATURE and/or *FIELD
```
Data lines to specify values of predefined fields
```
*BOUNDARY
```
Data lines to specify zero-valued or nonzero boundary conditions
```
*END STEP
**
*STEP(,NLGEOM)
*STATIC
```
Data line to define incrementation
```
*CHANGE FRICTION
*FRICTION
```
Data lines to redefine friction coefficient
```
*MOTION, ROTATION or TRANSLATION
```
Data lines to define the velocity differential
```
*END STEP
**
*STEP
*FREQUENCY
```
Data line to control eigenvalue extraction
```
*END STEP
**
*STEP
*COMPLEX FREQUENCY
```
Data line to control complex eigenvalue extraction
```
*SELECT EIGENMODES
```
Data lines to define applicable mode ranges
```
*END STEP
```

6.3.7 TRANSIENT MODAL DYNAMIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Dynamic analysis procedures: overview," Section 6.3.1
- *MODAL DYNAMIC
- "Configuring a modal dynamics procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A modal dynamic analysis:

- is used to analyze transient linear dynamic problems using modal superposition;
- can be performed only after a frequency extraction procedure since it bases the structure's response on the modes of the system;
- can use the high-performance SIM software architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1);
- can include nondiagonal damping effects (i.e., from material or element damping) only when using the SIM architecture; and
- is a linear perturbation procedure.

Modal dynamic analysis

Transient modal dynamic analysis gives the response of the model as a function of time based on a given time-dependent loading. The structure's response is based on a subset of the modes of the system, which must first be extracted using an eigenfrequency extraction procedure ("Natural frequency extraction," Section 6.3.5). The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

The modal amplitudes are integrated through time, and the response is synthesized from these modal responses. For linear systems the modal dynamic procedure is much less expensive computationally than the direct integration of the entire system of equations performed in the dynamic procedure ("Implicit dynamic analysis using direct integration," Section 6.3.2).

As long as the system is linear and is represented correctly by the modes being used (which are generally only a small subset of the total modes of the finite element model), the method is also very accurate because the integration operator used is exact whenever the forcing functions vary piecewise linearly with time. You should ensure that the forcing function definition and the choice of time increment

are consistent for this purpose. For example, if the forcing is a seismic record in which acceleration values are given every millisecond and it is assumed that the acceleration varies linearly between these values, the time increment used in the modal dynamic procedure should be a millisecond.

The user-specified maximum number of increments is ignored in a modal dynamic step. The number of increments is based on both the time increment and the total time chosen for the step.

While the response in this procedure is for linear vibrations, the prior response can be nonlinear and stress stiffening (initial stress) effects will be included in the response if nonlinear geometric effects were included in the step definition for the base state of the eigenfrequency extraction procedure, as explained in "Natural frequency extraction," Section 6.3.5.

Selecting the modes and specifying damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS
*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying modal damping

Damping is almost always specified for a mode-based procedure; see "Material damping," Section 26.1.1. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

*MODAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

∗MODAL DAMPING, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Damping**

Defining damping by specifying frequency ranges is not supported in Abaqus/CAE.

Example of specifying damping

Figure 6.3.7–1 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
```
f_1, d_1
f_2, d_2
f_2, d_3
f_3, d_3
f_4, d_4

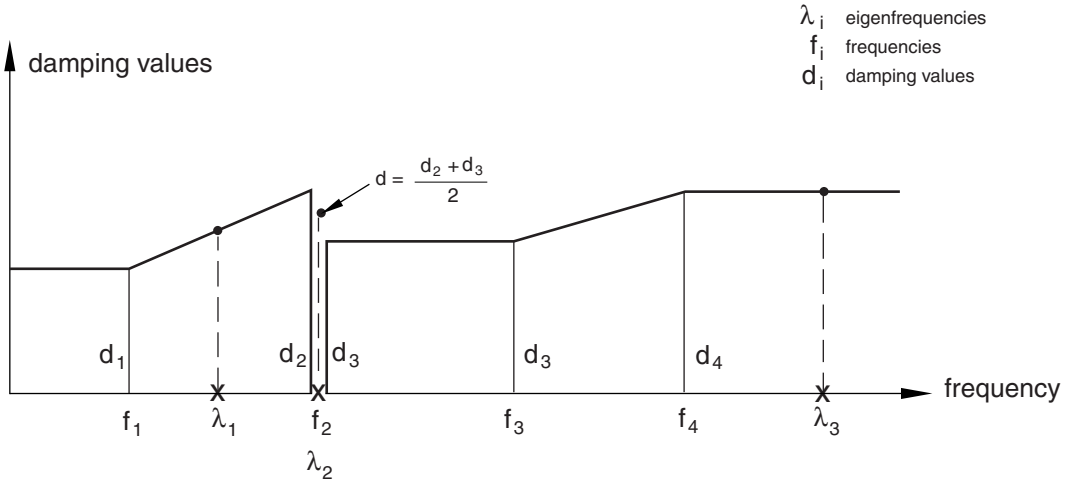


Figure 6.3.7–1 Damping coefficients specified by frequency range.

Rules for selecting modes and specifying damping coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.

- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Specifying global damping

For convenience you can specify constant global damping factors for all selected eigenmodes for mass and stiffness proportional viscous factors, as well as stiffness proportional structural damping. Structural damping is a commonly used damping model that represents damping as complex stiffness. This representation causes no difficulty for frequency domain analysis such as steady-state dynamics for which the solution is already complex. However, the solution must remain real-valued in the time domain. To allow users to apply their structural damping model in the time domain, a method has been developed to convert structural damping to an equivalent viscous damping. This technique was designed so that the viscous damping applied in the frequency domain is identical to the structural damping if the projected damping matrix is diagonal. For further details, see "Modal dynamic analysis," Section 2.5.5 of the Abaqus Theory Guide.

Input File Usage: ∗GLOBAL DAMPING, ALPHA=*factor*, BETA=*factor*, STRUCTURAL=*factor*

Abaqus/CAE Usage: Defining damping by global factors is not supported in Abaqus/CAE.

Material damping

Structural and viscous material damping (see "Material damping," Section 26.1.1) is taken into account in a SIM-based transient modal analysis. Since the projection of damping onto the mode shapes is performed only one time during the frequency extraction step, significant performance advantages can be achieved by using the SIM-based transient modal procedure (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1).

If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

Controlling damping

You can deactivate the structural or viscous damping in a transient modal procedure if desired.

You can also control damping of the low frequency eigenmodes in transient modal analyses. This control is useful for free structures and models with secondary base motions, and it controls all sources of damping including the modal damping. To include low frequency eigenmodes, set the low frequency cutoff value to a small negative value. To exclude them, either provide a low frequency cutoff value or

allow Abaqus to calculate it; it will be six orders of magnitude smaller than the eigenfrequency of the first deformable eigenmode.

Input File Usage: Use the following option to deactivate structural and viscous damping in a specific transient modal dynamic step:

*DAMPING CONTROLS, STRUCTURAL=NONE, VISCOUS=NONE

Use the following option to control damping of the low frequency eigenmodes in a specific transient modal dynamic step:

*DAMPING CONTROLS, LOW FREQUENCY CUTOFF

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Initial conditions

By default, the modal dynamic step will begin with zero initial displacements. If initial velocities have been defined ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1), they will be used; otherwise, the initial velocities will be zero.

Alternatively, you can force the modal dynamic step to carry over the initial conditions from the immediately preceding step, which must be either another modal dynamic step or a static perturbation step:

- In most cases if the immediately preceding step is a modal dynamic step, both the displacements and velocities are carried over from the end of that step and used as initial conditions for the current step. For a SIM-based analysis, you should use secondary base motion instead of primary base motion (see "Prescribed motions in modal superposition procedures") to carry over the initial conditions; Abaqus issues a warning message if primary base motion is used.
- If the immediately preceding step is a static perturbation step, the displacements are carried over from that step. If initial velocities have been defined ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1), they will be used; otherwise, the initial velocities will be zero.

Input File Usage: Use the following option to begin the modal dynamic step with zero initial displacements:

*MODAL DYNAMIC, CONTINUE=NO

Use the following option to force the modal dynamic step to carry over the initial conditions from the immediately preceding step:

*MODAL DYNAMIC, CONTINUE=YES

Abaqus/CAE Usage: Use the following option to begin the modal dynamic step with zero initial displacements:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Basic**: **Zero initial conditions**

Use the following option to force the modal dynamic step to carry over the initial conditions from the immediately preceding step:

Step module: **Create Step**: **Linear perturbation**: **Modal dynamics**: **Basic**: **Use initial conditions**

Boundary conditions

It is not possible to prescribe nonzero displacements and rotations (or acoustic pressure) directly as boundary conditions ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) in mode-based dynamic response procedures. In these procedures the motion for nodes can be specified only as base motion, as described below. Nonzero displacement or acceleration history definitions given as boundary conditions are ignored in modal superposition procedures, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors.

Prescribed motions in modal superposition procedures

Boundary conditions must be applied during the eigenfrequency extraction step to the degrees of freedom that will be prescribed in the modal dynamic procedure. These degrees of freedom are grouped into one or more "bases" (see "Natural frequency extraction," Section 6.3.5). The unnamed base is called the "primary" base. Named "secondary" bases must be defined by specifying boundary conditions in the frequency extraction step. A different motion can be prescribed for each base.

The far-field nodes of infinite elements are always added to the primary base in modal procedures. The far-field nodes are added to the primary base even if they are not explicitly constrained via boundary conditions.

Specifying the degree of freedom and the time history of the motion

The displacements and rotations that are associated with a base are prescribed during the modal dynamic response procedure. The base motions are fully defined by at most three global translations and three global rotations. Thus, at most one base motion can be defined for each translation and rotation component. Base motions are always specified in global directions, regardless of the use of nodal transformations. You specify the global direction (1–6) for which the base motion is being defined. If a rotation is specified about an origin that is not the origin of the coordinates, you must specify the center of rotation.

The time history of a motion must be defined by an amplitude curve ("Amplitude curves," Section 34.1.2).

Input File Usage: *BASE MOTION, DOF=*n*, AMPLITUDE=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Scaling the amplitude of the base motion

The amplitude curve used to define the time history of the motion can be scaled. By default, the scaling factor is 1.0.

Input File Usage: *BASE MOTION, DOF=*n*, AMPLITUDE=*name*, SCALE=*n*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*; **Amplitude scale factor:** *n*

Specifying the type of base motion

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. Furthermore, if the displacement or velocity histories have nonzero initial values, Abaqus/Standard will make corrections to the initial accelerations as described in "Modal dynamic analysis," Section 2.5.5 of the Abaqus Theory Guide. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

Input File Usage: Use one of the following options:

*BASE MOTION, DOF=*n*, AMPLITUDE=*name*, TYPE=ACCELERATION
*BASE MOTION, DOF=*n*, AMPLITUDE=*name*, TYPE=VELOCITY
*BASE MOTION, DOF=*n*, AMPLITUDE=*name*, TYPE=DISPLACEMENT

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**

Specifying secondary base motion

The primary base motion is specified by defining a base motion without referring to a base. If the base motion is to be applied to a secondary base, it must refer to the name of the base defined in the eigenfrequency extraction step.

For named bases you can also use degree of freedom 8 as the acoustic pressure degree of freedom. Acoustic pressure variation can be used in secondary bases only with big mass scaling. In such cases an acoustic mass is used to scale the big mass value. To specify an acoustic pressure, use the displacement base motion; to specify the first and the second derivative of acoustic pressure, use the velocity and acceleration, respectively.

Input File Usage: *BASE MOTION, DOF=*n*, AMPLITUDE=*name*, BASE NAME=*secondary base*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *modal_dynamic_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; toggle on **Secondary base:** *boundary_condition_name*

Example

To illustrate the concept of primary and secondary bases, consider a single-bay frame with supports at nodes 1 and 4. If the input prior to the eigenfrequency extraction step includes the following boundary conditions:

- degrees of freedom 1 through 6 constrained at node 1
- degree of freedom 1 constrained at node 4
- degrees of freedom 3 through 6 constrained at node 4

and different base motions are assigned to degree of freedom 2 at nodes 1 and 4, the following step definitions could be used:

- an eigenfrequency extraction step that includes a boundary condition associated with **BASE2** constraining degree of freedom 2 at node 4; and
- a modal dynamic step that includes two base motion definitions: the primary base motion assigned to degree of freedom 2 that does not refer to a base and the secondary base motion assigned to degree of freedom 2 that refers to **BASE2**.

If boundary conditions were not given prior to the eigenfrequency extraction step, you would have to define them in the eigenfrequency extraction step. Again, the secondary base would be defined by a boundary condition with a base name.

Calculating the response of the structure

The degrees of freedom associated with the primary base are set to zero in the eigenfrequency extraction step, and primary base motions are introduced by multiplying the base acceleration with the modal participation factors. Hence, Abaqus/Standard calculates the response of the structure with respect to the primary base. If the rotational degrees of freedom are references in the primary base motion definition, the rotation is defined, as default, about the origin of the coordinate system unless you provide the center of rotation.

The degrees of freedom associated with the secondary bases are not set to zero in the eigenfrequency extraction step; instead, a "big" mass is added to each of them. Any degree of freedom in a secondary base that was constrained by a regular boundary condition in a previous general step will be released, and a big mass will be added to that degree of freedom. Secondary base motions are introduced by nodal forces, obtained by multiplying the base acceleration (or acoustic pressure) with the big mass (or acoustic big mass). Although the secondary base motions are defined in absolute terms, the response calculated at the secondary bases is relative to the motion of the primary base for the translational degrees of freedom. The rotational secondary bases are defined about the nodes included in the node sets specified in the base name definition. Therefore, you cannot change the center of rotation for secondary bases.

For a more detailed description of the base motion procedure, see "Base motions in modal-based procedures," Section 2.5.9 of the Abaqus Theory Guide.

Loads

The following loads can be prescribed in modal dynamic analysis, as described in "Concentrated loads," Section 34.4.2:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

Predefined temperature fields are not allowed in transient modal dynamic analysis. Other predefined fields are ignored.

Material options

The density of the material must be defined ("Density," Section 21.2.1). The following material properties are not active during a modal dynamic analysis: plasticity and other inelastic effects, rate-dependent material properties, thermal properties, mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties. See "General and linear perturbation procedures," Section 6.1.3.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard (including those with temperature and pressure degrees of freedom) can be used in a modal dynamic analysis.

Output

All the output variables in Abaqus/Standard are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1. The values of nodal solution variables U, V, and A in modal dynamics in the time domain are relative to the motion of the primary base. Hence, the sum of the relative motion and the base motion of the primary base yields the total motion; this total motion is available by requesting output variables TU, TV, and TA. In the absence of primary base motions, the relative and total motions are identical.

The following modal variables can be output to the data or results files (see "Output to the data and results files," Section 4.1.2):

GU	Generalized displacements for all modes.
GV	Generalized velocities for all modes.
GA	Generalized accelerations for all modes.
SNE	Elastic strain energy for the entire model per each mode.

KE	Kinetic energy for the entire model per each mode.
T	External work for the entire model per each mode.
BM	Base motion.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in modal dynamic analysis. However, whole model variables such as ALLIE (total strain energy) are available for mode-based procedures as output to the data or results files (see "Output to the data and results files," Section 4.1.2).

The computational expense of a modal dynamic analysis can be decreased significantly by reducing the amount of output requested.

Input file template

```
*HEADING
…
*AMPLITUDE, NAME=amplitude
Data lines to define amplitude variations
**
*STEP
*FREQUENCY
Data line to specify the number of modes to be extracted
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*BOUNDARY, BASE NAME=base
Data lines to assign degrees of freedom to a secondary base
*END STEP
**
*STEP
*MODAL DYNAMIC
Data line to control time incrementation
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data line to define modal damping
*BASE MOTION, DOF=dof, AMPLITUDE=amplitude
*BASE MOTION, DOF=dof, AMPLITUDE=amplitude, BASE NAME=base
*END STEP
```

6.3.8 MODE-BASED STEADY-STATE DYNAMIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Dynamic analysis procedures: overview," Section 6.3.1
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Subspace-based steady-state dynamic analysis," Section 6.3.9
- *STEADY STATE DYNAMICS
- "Configuring a mode-based steady-state dynamic analysis" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A mode-based steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is a linear perturbation procedure;
- calculates the response based on the system's eigenfrequencies and modes;
- requires that an eigenfrequency extraction procedure be performed prior to the steady-state dynamic analysis;
- can use the high-performance SIM software architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1);
- can include nondiagonal damping effects (i.e., from material or element damping) only when using the SIM architecture;
- is an alternative to direct-solution steady-state dynamic analysis, in which the system's response is calculated in terms of the physical degrees of freedom of the model;
- can include computation of acoustic contribution factors to help determine the major contributors to acoustic noise;
- is computationally cheaper than direct-solution or subspace-based steady-state dynamics;

- is less accurate than direct-solution or subspace-based steady-state analysis, in particular if significant material damping is present, and
- is able to bias the excitation frequencies toward the values that generate a response peak.

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system due to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep by applying the loading at a series of different frequencies and recording the response; in Abaqus/Standard the steady-state dynamic analysis procedure is used to conduct the frequency sweep.

In a mode-based steady-state dynamic analysis the response is based on modal superposition techniques; the modes of the system must first be extracted using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

When defining a mode-based steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below ("Selecting the frequency spacing"). Logarithmic frequency spacing is the default. Frequencies are given in cycles/time.

These frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (see "The bias parameter," below).

While the response in this procedure is for linear vibrations, the prior response can be nonlinear. Initial stress effects (stress stiffening) will be included in the steady-state dynamics response if nonlinear geometric effects ("General and linear perturbation procedures," Section 6.1.3) were included in any general analysis step prior to the eigenfrequency extraction step preceding the steady-state dynamic procedure.

Input File Usage: *STEADY STATE DYNAMICS

The DIRECT and SUBSPACE PROJECTION parameters must be omitted from the *STEADY STATE DYNAMICS option to conduct a mode-based steady-state dynamic analysis.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**

Selecting the type of frequency interval for which output is requested

Three types of frequency intervals are permitted for output from a mode-based steady-state dynamic step.

Specifying the frequency ranges by using the system's eigenfrequencies

By default, the eigenfrequency type of frequency interval is used; in this case the following intervals exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function (which is discussed below and allows the sampling points on the frequency scale to be spaced closer together at eigenfrequencies in the frequency range). Thus, detailed definition of the response close to resonance frequencies is allowed. Figure 6.3.8–1 illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

Input File Usage: *STEADY STATE DYNAMICS, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Use eigenfrequencies to subdivide each frequency range**

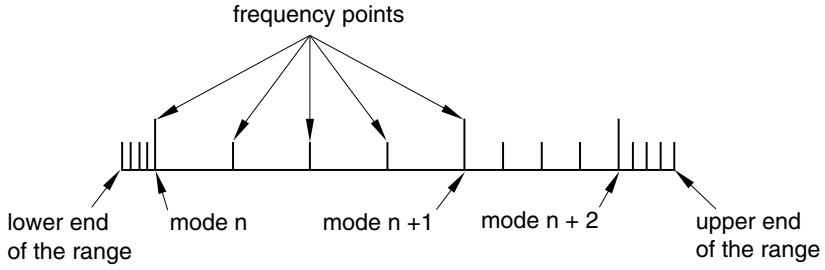


Figure 6.3.8–1 Division of range for the eigenfrequency type of interval and 5 calculation points.

Specifying the frequency ranges by the frequency spread

If the spread type of frequency interval is selected, intervals exist around each eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. Figure 6.3.8–2 illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

Input File Usage: *STEADY STATE DYNAMICS, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

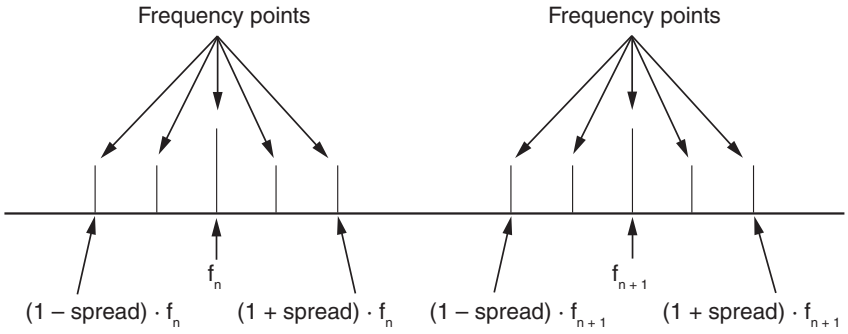


Figure 6.3.8–2 Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Specifying the frequency ranges directly

If the alternative range type of frequency interval is chosen, there is only one interval in the specified frequency range spanning from the lower to the upper limit of the range. This interval is divided using the user-defined number of points and the optional bias function, which can be used to space the sampling frequency points closer to the range limits. For the range type of frequency interval, the peak responses around the system's eigenfrequencies may be missed since the sampling frequencies at which output will be reported will not be biased toward the eigenfrequencies.

Input File Usage: *STEADY STATE DYNAMICS, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Selecting the frequency spacing

Two types of frequency spacing are permitted for a mode-based steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use either of the following options:

*STEADY STATE DYNAMICS, FREQUENCY SCALE=LOGARITHMIC
*STEADY STATE DYNAMICS, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Scale: Logarithmic** or **Linear**

Requesting multiple frequency ranges

You can request multiple frequency ranges or multiple single frequency points for a mode-based steady-state dynamic step.

Input File Usage: *STEADY STATE DYNAMICS
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...
Repeat the data lines as often as necessary.

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Modal**: **Data**: enter data in table, and add rows as necessary

The bias parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. Figure 6.3.8–3 shows a few examples of the effect of the bias parameter on the frequency spacing.

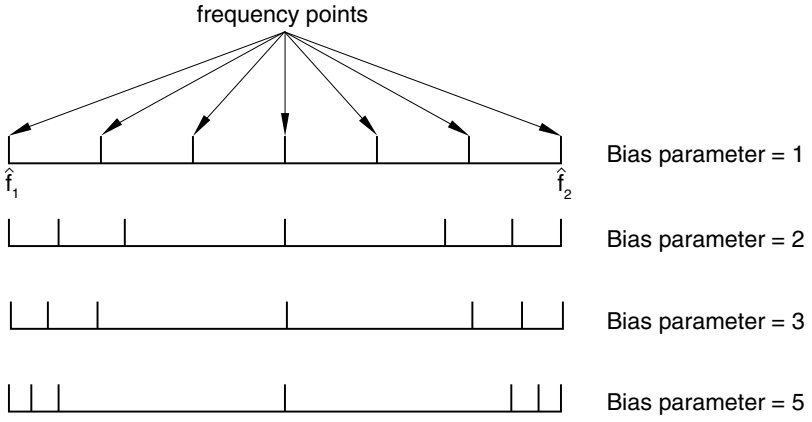


Figure 6.3.8–3 Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used to calculate the frequency at which results are presented is as follows:

$$\hat{f}_k = \frac{1}{2}(\hat{f}_1 + \hat{f}_2) + \frac{1}{2}(\hat{f}_2 - \hat{f}_1)|y|^{1/p}\text{sign}(y),$$

where

y $= -1 + 2(k-1)/(n-1)$;

n is the number of frequency points at which results are to be given within a frequency interval (discussed above);

k is one such frequency point ($k = 1, 2, \ldots, n$);
$\hat{f}_1$ is the lower limit of the frequency interval;
$\hat{f}_2$ is the upper limit of the frequency interval;
$\hat{f}_k$ is the frequency at which the kth results are given;
p is the bias parameter value; and
$\hat{f}$ is the frequency or the logarithm of the frequency, depending on the value used for the frequency scale parameter.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 3.0 for an eigenfrequency interval and 1.0 for a range frequency interval.

The frequency scale factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see "Specifying the frequency ranges by using the system's eigenfrequencies," above).

Selecting the modes and specifying damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS
*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying modal damping

Damping is almost always specified for a steady-state analysis (see "Material damping," Section 26.1.1). If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal

to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in "Material damping," Section 26.1.1. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

*MODAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

*MODAL DAMPING, DEFINITION=FREQUENCY RANGE

Use the following option to define damping by global factors:

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**:
Steady-state dynamics, Modal: **Damping**

Defining damping by specifying frequency ranges is not supported in Abaqus/CAE.

Example of specifying damping

Figure 6.3.8–4 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
```
f_1, d_1
f_2, d_2
f_2, d_3
f_3, d_3
f_4, d_4

Rules for selecting modes and specifying damping coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.

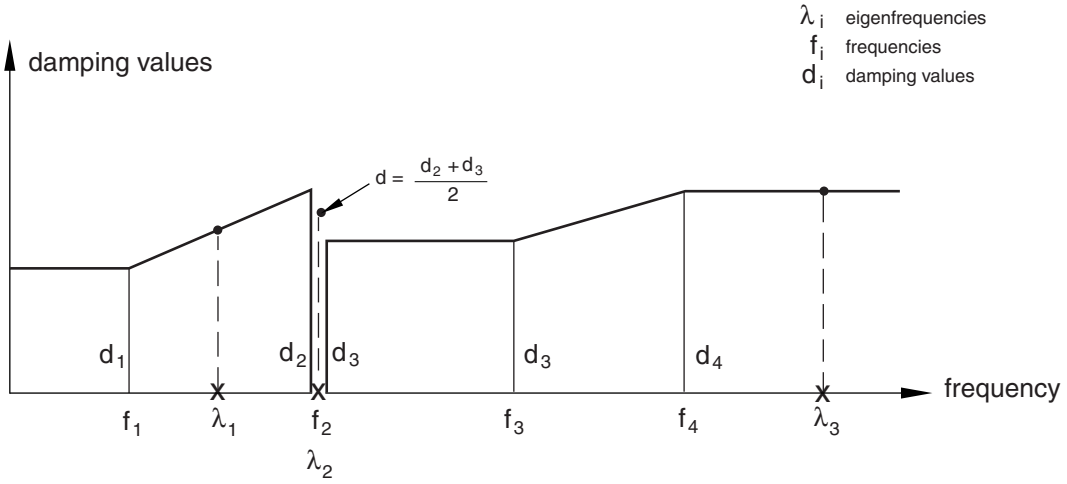


Figure 6.3.8–4 Damping values specified by frequency range.

- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Specifying global damping

For convenience you can specify constant global damping factors for all selected eigenmodes for mass and stiffness proportional viscous factors, as well as stiffness proportional structural damping. For further details, see "Damping in dynamic analysis" in "Dynamic analysis procedures: overview," Section 6.3.1.

Input File Usage: ∗GLOBAL DAMPING, ALPHA=*factor*, BETA=*factor*, STRUCTURAL=*factor*

Abaqus/CAE Usage: Defining damping by global factors is not supported in Abaqus/CAE.

Material damping

Structural and viscous material damping (see "Material damping," Section 26.1.1) is taken into account in a SIM-based steady-state dynamic analysis. Since the projection of damping onto the mode shapes is performed only one time during the frequency extraction step, significant performance advantages can be achieved by using the SIM-based steady-state dynamic procedure (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1).

If the damping operators depend on frequency, they will be evaluated at the frequency specified for property evaluation during the frequency extraction procedure.

You can deactivate the structural or viscous damping in a mode-based steady-state dynamic procedure if desired.

Input File Usage: Use the following option to deactivate structural and viscous damping in a specific steady-state dynamic step:

*DAMPING CONTROLS, STRUCTURAL=NONE, VISCOUS=NONE

Abaqus/CAE Usage: Damping controls are not supported in Abaqus/CAE.

Initial conditions

The base state is the current state of the model at the end of the last general analysis step prior to the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary conditions

In a mode-based steady-state dynamic analysis both the real and imaginary parts of any degree of freedom are either restrained or unrestrained; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will automatically restrain both the real and imaginary parts of a degree of freedom even if only one part is restrained.

Base motion

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) in mode-based dynamic response procedures. Therefore, in a mode-based steady-state dynamic analysis, the motion of nodes can be specified only as base motion; nonzero displacement or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. The method for prescribing base motion in modal superposition procedures is described in "Transient modal dynamic analysis," Section 6.3.7.

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

When secondary bases are used, low frequency eigenmodes will be extracted for each "big" mass applied in the model. Use care when choosing the frequency lower limit range in such cases. The "big" mass modes are important in the modal superposition; however, the response at zero or arbitrarily low frequency level should not be requested since it forces Abaqus/Standard to calculate responses at frequencies between these "big" mass eigenfrequencies, which is not desirable.

Frequency-dependent base motion

An amplitude definition can be used to specify the amplitude of a base motion as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

∗AMPLITUDE, NAME=*name*
∗BASE MOTION, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *step_name*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Loads

The following loads can be prescribed in a mode-based steady-state dynamic analysis, as described in "Concentrated loads," Section 34.4.2:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in Part VI, "Elements."

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Fluid flux loading cannot be used in a steady-state dynamic analysis.

Input File Usage: Use either of the following input lines to define the real (in-phase) part of the load:

∗CLOAD *or* ∗DLOAD
∗CLOAD *or* ∗DLOAD, REAL

Use the following input line to define the imaginary (out-of-phase) part of the load:

∗CLOAD *or* ∗DLOAD, IMAGINARY

Abaqus/CAE Usage: Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**

Frequency-dependent loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

∗AMPLITUDE, NAME=*name*
∗CLOAD *or* ∗DLOAD, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:** *name*

Load module: load editor: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**: **Amplitude:** *name*

Predefined fields

Predefined temperature fields are not allowed in mode-based steady-state dynamic analysis. Other predefined fields are ignored.

Material options

As in any dynamic analysis procedure, mass or density ("Density," Section 21.2.1) must be assigned to some regions of any separate parts of the model where dynamic response is required. The following material properties are not active during mode-based steady-state dynamic analyses: plasticity and other inelastic effects, viscoelastic effects, thermal properties, mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see "General and linear perturbation procedures," Section 6.1.3.

Elements

Any of the following elements available in Abaqus/Standard can be used in a steady-state dynamics procedure:

- stress/displacement elements (other than generalized axisymmetric elements with twist);
- acoustic elements;
- piezoelectric elements; or
- hydrostatic fluid elements.

See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

In mode-based steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see "Abaqus/Standard output variable identifiers," Section 4.2.1). In this case the first printed line in the data file gives the magnitude while the second gives the phase angle.

The following variables are provided specifically for steady-state dynamic analysis:

Element integration point variables:

PHS	Magnitude and phase angle of all stress components.
PHE	Magnitude and phase angle of all strain components.
PHEPG	Magnitude and phase angles of the electrical potential gradient vector.
PHEFL	Magnitude and phase angles of the electrical flux vector.
PHMFL	Magnitude and phase angle of the mass flow rate in fluid link elements.
PHMFT	Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF	Magnitude and phase angle of connector total forces.
PHCEF	Magnitude and phase angle of connector elastic forces.
PHCVF	Magnitude and phase angle of connector viscous forces.
PHCRF	Magnitude and phase angle of connector reaction forces.
PHCSF	Magnitude and phase angle of connector friction forces.
PHCU	Magnitude and phase angle of connector relative displacements.
PHCCU	Magnitude and phase angle of connector constitutive displacements.

Nodal variables:

PU	Magnitude and phase angle of all displacement/rotation components at a node.
PPOR	Magnitude and phase angle of the fluid or acoustic pressure at a node.
PHPOT	Magnitude and phase angle of the electrical potential at a node.
PRF	Magnitude and phase angle of all reaction forces/moments at a node.
PHCHG	Magnitude and phase angle of the reactive charge at a node.

Element energy densities (such as the elastic strain energy density, SENER) and whole element energies (such as the total kinetic energy of an element, ELKE) are not available for output in a mode-based steady-state dynamic analysis.

The standard output variables U, V, A, and the variable PU listed above correspond to motions relative to the motion of the primary base in a mode-based analysis. Total values, which include the motion of the primary base, are also available:

TU	Magnitude of all components of total displacement/rotation at a node.
TV	Magnitude of all components of total velocity at a node.
TA	Magnitude of all components of total acceleration at a node.
PTU	Magnitude and phase angle of all total displacement/rotation components at a node.

The following modal variables are also available for mode-based steady-state dynamic analysis and can be output to the data, results, and/or output database files (see “Output to the data and results files,” Section 4.1.2, and “Output to the output database,” Section 4.1.3):

GU	Generalized displacements for all modes.
GV	Generalized velocities for all modes.
GA	Generalized accelerations for all modes.
GPU	Phase angle of generalized displacements for all modes.
GPV	Phase angle of generalized velocities for all modes.
GPA	Phase angle of generalized acceleration for all modes.
SNE	Elastic strain energy for the entire model per mode.

KE	Kinetic energy for the entire model per mode.
T	External work for the entire model per mode.
BM	Base motion.

Whole model variables such as ALLIE (total strain energy) are available for mode-based steady-state dynamics as output to the data, results, and/or output database files (see "Output to the data and results files," Section 4.1.2).

Acoustic contribution factors

Computation of the acoustic contribution factors helps you determine the major noise sources. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. For more information, see "Acoustic contribution factors in mode-based and subspace-based steady-state dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1.

Input file template

```
*HEADING
…
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
*AMPLITUDE, NAME=base
Data lines to define an amplitude curve to be used to prescribe base motion
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamics step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
*END STEP
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
```

***BOUNDARY, BASE NAME=base2**
Data lines to assign degrees of freedom to a secondary base
***END STEP**
******
***STEP**
***STEADY STATE DYNAMICS**
Data lines to specify frequency ranges and bias parameters
***SELECT EIGENMODES**
Data lines to define the applicable mode ranges
***ACOUSTIC CONTRIBUTION**
***MODAL DAMPING**
Data lines to define the modal damping factors
BASE MOTION, DOF=dof***, AMPLITUDE=base**
BASE MOTION, DOF=dof***, AMPLITUDE=base, BASE NAME=base2**
***CLOAD and/or *DLOAD, AMPLITUDE=loadamp**
Data lines to specify sinusoidally varying, frequency-dependent loads
…
***END STEP**

6.3.9 SUBSPACE-BASED STEADY-STATE DYNAMIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Dynamic analysis procedures: overview," Section 6.3.1
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- *STEADY STATE DYNAMICS
- "Configuring a subspace-based steady-state dynamic analysis" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A subspace-based steady-state dynamic analysis:

- is used to calculate the steady-state dynamic linearized response of a system to harmonic excitation;
- is based on projection of the steady-state dynamic equations on a subspace of selected modes of the undamped system;
- is a linear perturbation procedure;
- provides a cost-effective way to include frequency-dependent effects (such as frequency-dependent damping and viscoelastic effects) in the model;
- allows for nonsymmetric stiffness;
- requires that an eigenfrequency extraction procedure be performed prior to the steady-state dynamic analysis;
- can use the high-performance SIM software architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1);
- is an alternative to direct-solution steady-state dynamic analysis, in which the system's response is calculated in terms of the physical degrees of freedom of the model;
- can include computation of acoustic contribution factors to help determine the major contributors to acoustic noise;
- is computationally cheaper than direct-solution steady-state dynamics but more expensive than mode-based steady-state dynamics;

- is less accurate than direct-solution steady-state analysis, in particular if significant material damping or viscoelasticity with a high loss modulus is present; and
- is able to bias the excitation frequencies toward the values that generate a response peak.

Introduction

Steady-state dynamic analysis provides the steady-state amplitude and phase of the response of a system subjected to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep, by applying the loading at a series of different frequencies and recording the response. In Abaqus/Standard the subspace-based steady-state dynamic analysis procedure is used to conduct the frequency sweep.

In a subspace-based steady-state dynamic analysis the response is based on direct solution of the steady-state dynamic equations projected onto a subspace of modes. The modes of the undamped, symmetric system must first be extracted using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The procedure is based on the assumption that the forced steady-state vibration can be represented accurately by a number of modes of the undamped system that are in the range of the excitation frequencies of interest. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part. The projection of the dynamic equilibrium equations onto a subspace of selected modes leads to a small system of complex equations that is solved for modal amplitudes, which are then used to compute nodal displacements, stresses, etc.

When defining a subspace-based steady-state dynamic step, you specify the frequency ranges of interest and the number of frequencies at which results are required in each range (including the bounding frequencies of the range). In addition, you can specify the type of frequency spacing (linear or logarithmic) to be used, as described below ("Selecting the frequency spacing"). Logarithmic frequency spacing is the default if the frequency ranges are specified directly or by eigenfrequencies. If the frequency ranges are specified by the frequency spread, only linear spacing can be used. Frequencies should be given in cycles/time.

The frequency points for which results are required can be spaced equally along the frequency axis (on a linear or a logarithmic scale), or they can be biased toward the ends of the user-defined frequency range by introducing a bias parameter (see "The bias parameter" below).

The subspace-based steady-state dynamic analysis procedure can be used:

- for nonsymmetric stiffness;
- when any form of damping (except modal damping) is included; and
- when viscoelastic material properties must be taken into account.

While the response in this procedure is for linear vibrations, the prior response can be nonlinear. Initial stress effects (stress stiffening) will be included in the steady-state dynamic response if nonlinear geometric effects ("General and linear perturbation procedures," Section 6.1.3) were included in any general analysis step prior to the eigenfrequency extraction step preceding the subspace-based steady-state dynamic procedure.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**

Ignoring damping

If damping terms can be ignored, you can specify that a real, rather than a complex, system matrix be generated and projected, which can significantly reduce computational time, at the cost of ignoring the damping effects.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION, REAL ONLY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Compute real response only**

Selecting the type of frequency interval for which output is requested

Three types of frequency intervals are permitted for output from a subspace-based steady-state dynamic step.

Specifying the frequency ranges by using the system's eigenfrequencies

By default, the eigenfrequency type of frequency interval is used; in this case the following intervals exist in each frequency range:

- First interval: extends from the lower limit of the frequency range given to the first eigenfrequency in the range.
- Intermediate intervals: extend from eigenfrequency to eigenfrequency.
- Last interval: extends from the highest eigenfrequency in the range to the upper limit of the frequency range.

For each of these intervals the frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval) and the optional bias function (which is discussed below and allows the sampling points on the frequency scale to be spaced closer together at eigenfrequencies in the frequency range). Thus, detailed definition of the response close to resonance frequencies is allowed. Figure 6.3.9–1 illustrates the division of the frequency range for 5 calculation points and a bias parameter equal to 1.

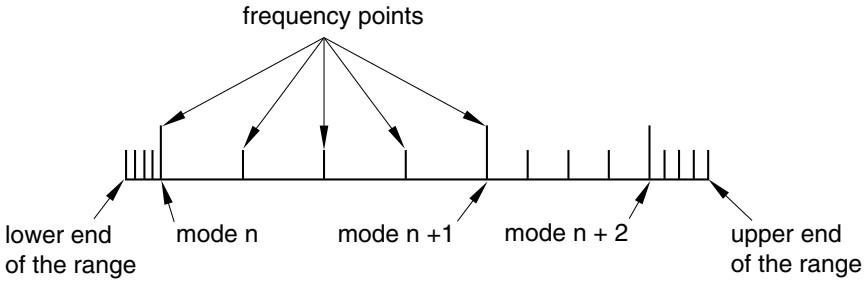


Figure 6.3.9–1 Division of range for the eigenfrequency type of interval and 5 calculation points.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION, INTERVAL=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Use eigenfrequencies to subdivide each frequency range**

Specifying the frequency ranges by the frequency spread

If the spread type of frequency interval is selected, intervals exist around each eigenfrequency in the frequency range. For each of the intervals the equally spaced frequencies at which results are calculated are determined using the user-defined number of points (which includes the bounding frequencies for the interval). The minimum number of frequency points is 3. If the user-defined value is less than 3 (or omitted), the default value of 3 points is assumed. Figure 6.3.9–2 illustrates the division of the frequency range for 5 calculation points.

The bias parameter is not supported with the spread type of frequency interval.

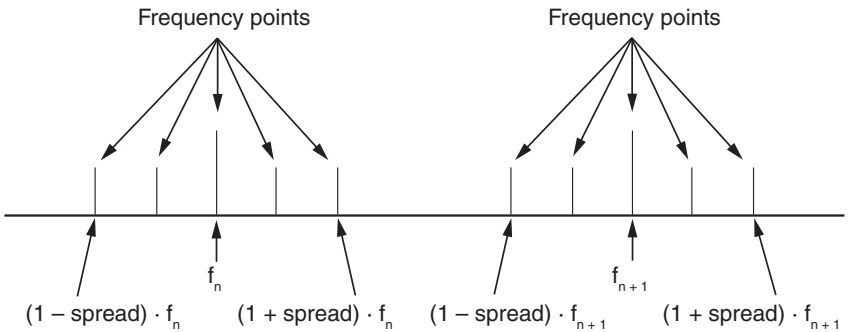


Figure 6.3.9–2 Division of range for the spread type of interval and 5 calculation points. f_n and f_{n+1} are eigenfrequencies of the system.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION, INTERVAL=SPREAD
lwr_freq, upr_freq, numpts, bias_param, freq_scale_factor, spread

Abaqus/CAE Usage: You cannot specify frequency ranges by frequency spread in Abaqus/CAE.

Specifying the frequency ranges directly

If the alternative range type of frequency interval is chosen, there is only one interval in the specified frequency range spanning from the lower to the upper limit of the range. This interval is divided using the user-defined number of points and the optional bias function, which can be used to space the sampling frequency points closer to the range limits. For the range type of frequency interval, the peak responses

around the system's eigenfrequencies may be missed since the sampling frequencies at which output will be reported will not be biased toward the eigenfrequencies.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION, INTERVAL=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: toggle off **Use eigenfrequencies to subdivide each frequency range**

Selecting the frequency spacing

Two types of frequency spacing are permitted for a subspace-based steady-state dynamic step. For the logarithmic frequency spacing (the default), the specified frequency ranges of interest are divided using a logarithmic scale. Alternatively, a linear frequency spacing can be used if a linear scale is desired.

Input File Usage: Use the following option to specify logarithmic frequency spacing:

*STEADY STATE DYNAMICS, SUBSPACE PROJECTION, FREQUENCY SCALE=LOGARITHMIC (default)

Use the following option to specify linear frequency spacing:

*STEADY STATE DYNAMICS, SUBSPACE PROJECTION, FREQUENCY SCALE=LINEAR

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Scale: Logarithmic** or **Linear**

Requesting multiple frequency ranges

You can request multiple frequency ranges for a subspace-based steady-state dynamic step. When both frequency ranges and additional single frequency points are requested, the frequency ranges must be specified first.

Input File Usage: Repeat the data lines as often as necessary to request multiple frequency ranges or multiple single frequency points:

*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
lwr_freq1, upr_freq1, numpts1, bias_param1, freq_scale_factor1
lwr_freq2, upr_freq2, numpts2, bias_param2, freq_scale_factor2
...
single_freq1
single_freq2
...

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Data**: enter data in table, and add rows as necessary

The bias parameter

The bias parameter can be used to provide closer spacing of the results points either toward the middle or toward the ends of each frequency interval. Figure 6.3.9–3 shows a few examples of the effect of the bias parameter on the frequency spacing.

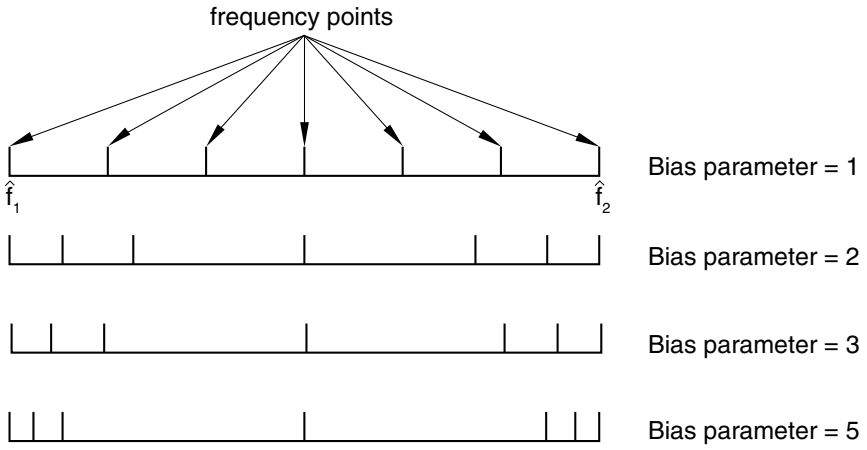


Figure 6.3.9–3 Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used in subspace-based steady-state dynamics is

$$\hat{f}_k = \frac{1}{2}(\hat{f}_1 + \hat{f}_2) + \frac{1}{2}(\hat{f}_2 - \hat{f}_1)|y|^{1/p}\text{sign}(y),$$

where

y	$= -1 + 2(k-1)/(n-1)$;
n	is the number of frequency points at which results are to be given within a frequency interval (discussed above);
k	is one such frequency point ($k = 1, 2, \ldots, n$);
$\hat{f}_1$	is the lower limit of the frequency interval;
$\hat{f}_2$	is the upper limit of the frequency interval;
$\hat{f}_k$	is the frequency at which the kth results are given;
p	is the bias parameter value; and
$\hat{f}$	is the frequency or the logarithm of the frequency, depending on the value chosen for the frequency scale.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of the frequency interval, while values of p that are less than 1.0 provide closer spacing toward the middle of the frequency interval. The default bias parameter is 3.0 for an eigenfrequency interval and 1.0 for a range frequency interval.

The frequency scale factor

The frequency scale factor can be used to scale frequency points. All the frequency points, except the lower and upper limit of the frequency range, are multiplied by this factor. This scale factor can be used only when the frequency interval is specified by using the system's eigenfrequencies (see "Specifying the frequency ranges by using the system's eigenfrequencies," above).

Damping

If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in "Material damping," Section 26.1.1.

In subspace-based steady-state dynamic analysis damping can be created by the following:

- dashpots (see "Dashpots," Section 32.2.1),
- "Rayleigh" damping associated with materials and elements (see "Material damping," Section 26.1.1),
- structural damping (see "Damping in dynamic analysis" in "Dynamic analysis procedures: overview," Section 6.3.1),
- viscoelasticity included in the material definitions (see "Frequency domain viscoelasticity," Section 22.7.2),
- contributions from infinite elements (see "Infinite elements," Section 28.3.1) or defined impedance conditions (see "Acoustic and shock loads," Section 34.4.6) on acoustic elements, and
- "volumetric drag" (viscous Rayleigh damping) in acoustic elements (see "Acoustic medium," Section 26.3.1).

If you specify that a real-only system matrix be generated and projected (see "Ignoring damping" above), all forms of damping are ignored, including quiet boundaries on infinite elements and nonreflecting boundaries on acoustic elements.

Contact conditions with sliding friction

Abaqus/Standard automatically detects the contact nodes that are slipping due to velocity differences imposed by the motion of the reference frame or the transport velocity in prior steps. At those nodes the tangential degrees of freedom are not constrained and the effect of friction results in an unsymmetric contribution to the stiffness matrix. At other contact nodes the tangential degrees of freedom are constrained.

Friction at contact nodes at which a velocity differential is imposed can give rise to damping terms. There are two kinds of friction-induced damping effects. The first effect is caused by the friction forces stabilizing the vibrations in the direction perpendicular to the slip direction. This effect exists only in three-dimensional analysis. The second effect is caused by a velocity-dependent friction coefficient. If the friction coefficient decreases with velocity (which is usually the case), the effect is destabilizing

and is also known as "negative damping." For more details, see "Coulomb friction," Section 5.2.3 of the Abaqus Theory Guide. Subspace-based steady-state dynamics analysis allows you to include these friction-induced contributions to the damping matrix.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION, FRICTION DAMPING=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Include friction-induced damping effects**

Selecting the modes on which to project

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use the following option to select the modes by specifying mode numbers individually:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS

Use the following option to request that Abaqus/Standard generate the mode numbers automatically:

*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Selecting the subspace projection frequency

You can control the frequency of the subspace projections. By default, the dynamic equations are projected onto the subspace at each frequency you request. However, considerable computational savings can be obtained if the projection onto the subspace is performed only at selected frequency points.

Projecting the subspace at each frequency requested

By default, the dynamic equations are projected onto the subspace at each frequency you requested. This is the most computationally expensive method. If coupled acoustic-structural modes are extracted in the preceding eigenfrequency extraction step, this is the only method allowed.

Input File Usage: Use either of the following options:

*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
*STEADY STATE DYNAMICS,
SUBSPACE PROJECTION=ALL FREQUENCIES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Evaluate at each frequency**

Projecting the subspace using model properties at the center frequency of all ranges

You can perform only one projection using model properties evaluated at the center frequency of all ranges and individual frequency points specified. The center frequency is determined on a logarithmic or linear scale depending on the spacing requested.

This method is the least expensive. However, it should be chosen only when the material properties do not depend strongly on frequency.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION=CONSTANT

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Constant**

Projecting the subspace at each extracted eigenfrequency

You can perform the projections at each extracted eigenfrequency in the requested frequency range and at eigenfrequencies immediately outside the range. The projected mass, stiffness, and damping matrices are then interpolated at each frequency point requested. The interpolation is performed on a linear or logarithmic scale depending on the spacing requested.

Input File Usage: *STEADY STATE DYNAMICS,
SUBSPACE PROJECTION=EIGENFREQUENCY

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Interpolate at eigenfrequencies**

Projecting the subspace based on material property changes as a function of frequency

You can select how often subspace projections are performed based on material property changes as a function of frequency. You specify the relative change in material stiffness and damping properties allowed before a new projection is performed. In the beginning of the subspace-based steady-state dynamic step Abaqus/Standard computes a table of relative changes in material stiffness and damping properties, and projections are performed based on the strictest of the two criteria. The projections are then interpolated at each requested frequency point as described above. The default value for the allowable stiffness or damping change is 0.1.

Input File Usage: *STEADY STATE DYNAMICS,
SUBSPACE PROJECTION=PROPERTY CHANGE,
DAMPING CHANGE=*percentage*, STIFFNESS CHANGE=*percentage*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **As a function of property changes**, **Max. damping change**: *percentage*, **Max. stiffness change**: *percentage*

Projecting the subspace at the limits of each frequency range

You can select how often subspace projections are performed based on the limits of each frequency range. The projections onto the modal subspace of the dynamic equations are performed at the lower limit of each frequency range and at the upper limit of the last frequency range. The interpolation of the projected mass, stiffness, and damping matrices is performed on a linear scale. This method can be used only with the SIM architecture.

This method should be chosen when the frequency dependence of material properties is close to linear within a frequency range.

Input File Usage: *STEADY STATE DYNAMICS, SUBSPACE PROJECTION=RANGE

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Steady-state dynamics, Subspace**: **Projection**: **Interpolate at lower and upper frequency limits**

Initial conditions

The base state is the current state of the model at the end of the last general analysis step prior to the steady-state dynamic step. If the first step of an analysis is a perturbation step, the base state is determined from the initial conditions (“Initial conditions in Abaqus/Standard and Abaqus/Explicit,” Section 34.2.1). Initial condition definitions that directly define solution variables, such as velocity, cannot be used in a steady-state dynamic analysis.

Boundary conditions

In a subspace-based steady-state dynamic analysis both the real and imaginary parts of any degree of freedom are either restrained or unrestrained; it is physically impossible to have one part restrained and the other part unrestrained. Abaqus/Standard will restrain both the real and imaginary parts of a degree of freedom automatically even if only one part is restrained.

Base motion

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions (“Boundary conditions in Abaqus/Standard and Abaqus/Explicit,” Section 34.3.1) in subspace-based steady-state dynamic analysis. Instead, prescribed motion can be specified as base motion; nonzero displacement or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. The method for prescribing base motion in modal superposition procedures is described in “Transient modal dynamic analysis,” Section 6.3.7.

Base motions can be defined by a displacement, a velocity, or an acceleration history. For an acoustic pressure the displacement is used to describe an acoustic pressure history. If the prescribed excitation record is given in the form of a displacement or velocity history, Abaqus/Standard differentiates it to obtain the acceleration history. The default is to give an acceleration history for mechanical degrees of freedom and to give a displacement for an acoustic pressure.

When secondary bases are used, low frequency eigenmodes are extracted for each “big” mass applied in the model. Use care when choosing the lower limit range for the frequency in such cases.

The "big" mass modes are important in the modal superposition. However, you should not request the response at zero or an arbitrarily low frequency level because this forces Abaqus/Standard to calculate the responses at frequencies between these "big" mass eigenfrequencies, which is not desirable.

Frequency-dependent base motion

An amplitude definition can be used to specify the amplitude of a base motion as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

∗AMPLITUDE, NAME=*name*
∗BASE MOTION, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *step_name*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Basic** tabbed page: **Degree-of-freedom: U1**, **U2**, **U3**, **UR1**, **UR2**, or **UR3**; **Amplitude:** *name*

Loads

The following loads can be prescribed in a subspace-based steady-state dynamic analysis, as described in "Concentrated loads," Section 34.4.2:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in Part VI, "Elements."
- Incident wave loads can be applied; see "Acoustic and shock loads," Section 34.4.6. Incident wave loads can be used to model sound waves from distinct planar or spherical sources or from diffuse fields.

These loads are assumed to vary sinusoidally with time over a user-specified range of frequencies. Loads are given in terms of their real and imaginary components.

Input File Usage: Use either of the following input lines to define the real (in-phase) part of the load:

∗CLOAD *or* ∗DLOAD
∗CLOAD *or* ∗DLOAD, REAL

Use the following input line to define the imaginary (out-of-phase) part of the load:

∗CLOAD *or* ∗DLOAD, IMAGINARY

Abaqus/CAE Usage: You can only define the real (in phase) part of the load in Abaqus/CAE.

Load module: load editor: *real (in-phase) part*

Frequency-dependent loading

An amplitude definition can be used to specify the amplitude of a load as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

*AMPLITUDE, NAME=*name*
*CLOAD *or* *DLOAD, REAL or IMAGINARY, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name:** *name*

Load module: load editor: **Amplitude:** *name*

Loading limitations

Coriolis distributed loading adds an imaginary antisymmetric contribution to the overall system of equations. This contribution is currently accounted for in solid and truss elements only and is activated by requesting the unsymmetric matrix storage and solution scheme for the step.

Fluid flux loading cannot be used in subspace-based steady-state dynamic analysis.

Predefined fields

Predefined temperature fields can be specified in subspace-based steady-state dynamic analysis (see "Predefined fields," Section 34.6.1) and will produce harmonically varying thermal strains if thermal expansion is included in the material definition ("Thermal expansion," Section 26.1.2). Other predefined fields are ignored.

Material options

As in any dynamic analysis procedure, mass or density ("Density," Section 21.2.1) must be assigned to some regions of any separate parts of the model where dynamic response is required. If an analysis is desired in which the inertia effects are neglected, the density should be set to a very small number. Natural damping, as well as individual dashpots, can be included in this procedure.

Viscoelastic effects can be included in subspace-based steady-state dynamic analysis. The linearized viscoelastic response is considered to be a perturbation about a nonlinear preloaded state, which is computed on the basis of purely elastic behavior (long-term response) in the viscoelastic components. Therefore, the vibration amplitude must be sufficiently small so that the material response in the dynamic phase of the problem can be treated as a linear perturbation about the predeformed state. Viscoelastic frequency domain response is described in "Frequency domain viscoelasticity," Section 22.7.2.

The following material properties are not active during subspace-based steady-state dynamic analyses: plasticity and other inelastic effects, thermal properties (except for thermal expansion), mass diffusion properties, electrical properties (except for the electrical potential, ϕ, in piezoelectric analysis), and pore fluid flow properties—see "General and linear perturbation procedures," Section 6.1.3.

Numerical investigations show that in general the accuracy of the results in the subspace-based steady-state dynamic step is improved if in the previous eigenfrequency extraction step the material properties are evaluated at a frequency in the vicinity of the center of the range spanned by the frequencies specified for the steady-state dynamic step (see "Natural frequency extraction," Section 6.3.5). In this case the modes extracted in the previous eigenfrequency extraction step for the undamped system will reflect most accurately the modes of the damped system at frequencies located in

the proximity of the frequency at which the material properties are evaluated. Thus, if the steady-state dynamic response is sought for a large span of frequencies and the specified material properties vary significantly over this span, the results will be more accurate if the range is divided into smaller ranges and several separate analyses are run over these smaller ranges with the material properties evaluated at appropriate frequencies.

Elements

Any of the following elements available in Abaqus/Standard can be used in a subspace-based steady-state dynamic analysis:

- stress/displacement elements (other than generalized axisymmetric elements with twist);
- acoustic elements;
- piezoelectric elements; and
- hydrostatic fluid elements.

See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

In subspace-based steady-state dynamic analysis the value of an output variable such as strain (E) or stress (S) is a complex number with real and imaginary components. In the case of data file output the first printed line gives the real components while the second lists the imaginary components. Results and data file output variables are also provided to obtain the magnitude and phase of many variables (see "Abaqus/Standard output variable identifiers," Section 4.2.1). In this case the first printed line in the data file gives the magnitude while the second gives the phase angle.

The following variables are provided specifically for subspace-based steady-state dynamic analysis:

Element integration point variables:

PHS	Magnitude and phase angle of all stress components.
PHE	Magnitude and phase angle of all strain components.
PHEPG	Magnitude and phase angles of the electrical potential gradient vector.
PHEFL	Magnitude and phase angles of the electrical flux vector.
PHMFL	Magnitude and phase angle of the mass flow rate in fluid link elements.
PHMFT	Magnitude and phase angle of the total mass flow in fluid link elements.

For connector elements, the following element output variables are available:

PHCTF	Magnitude and phase angle of connector total forces.
PHCEF	Magnitude and phase angle of connector elastic forces.
PHCVF	Magnitude and phase angle of connector viscous forces.
PHCRF	Magnitude and phase angle of connector reaction forces.
PHCSF	Magnitude and phase angle of connector friction forces.

PHCU	Magnitude and phase angle of connector relative displacements.
PHCCU	Magnitude and phase angle of connector constitutive displacements.
PHCV	Magnitude and phase angle of connector relative velocities.
PHCA	Magnitude and phase angle of connector relative accelerations.

Nodal variables:

PU	Magnitude and phase angle of all displacement/rotation components at a node.
PPOR	Magnitude and phase angle of the fluid or acoustic pressure at a node.
PHPOT	Magnitude and phase angle of the electrical potential at a node.
PRF	Magnitude and phase angle of all reaction forces/moments at a node.
PHCHG	Magnitude and phase angle of the reactive charge at a node.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in a SIM-based, subspace-based steady-state dynamic analysis.

The standard output variables U, V, A, and the variable PU listed above correspond to motions relative to the motion of the primary base in a subspace-based steady-state dynamic analysis. Total values, which include the motion of the primary base, are also available:

TU	Components of total displacement/rotation at a node.
TV	Components of total velocity at a node.
TA	Components of total acceleration at a node.
PTU	Magnitude and phase angle of all total displacement/rotation components at a node.

The specified base motion is available for subspace-based steady-state dynamic analysis and can be output to the data, results, and/or output database files (see “Output to the data and results files,” Section 4.1.2, and “Output to the output database,” Section 4.1.3).

BM	Base motion.

Whole model variables such as ALLIE (total strain energy) are available for subspace-based steady-state dynamic analysis as output to the data, results, and/or output database files (see “Output to the data and results files,” Section 4.1.2, and “Output to the output database,” Section 4.1.3).

Acoustic contribution factors

Computation of the acoustic contribution factors helps you determine the major noise sources. The procedure for computing the acoustic contribution factors is based on the modal analysis formulation of acoustic-structural problems with uncoupled modes. For more information, see “Acoustic contribution factors in mode-based and subspace-based steady-state dynamic analyses” in “Dynamic analysis procedures: overview,” Section 6.3.1.

Input file template

```
*HEADING
…
*AMPLITUDE, NAME=loadamp
Data lines to define an amplitude curve as a function of frequency (cycles/time)
*AMPLITUDE, NAME=base
Data lines to define an amplitude curve to be used to prescribe base motion
**
*STEP, NLGEOM
Include the NLGEOM parameter so that stress stiffening effects will
be included in the steady-state dynamics step
*STATIC
**Any general analysis procedure can be used to preload the structure
…
*CLOAD and/or *DLOAD
Data lines to prescribe preloads
*TEMPERATURE and/or *FIELD
Data lines to define values of predefined fields for preloading the structure
*BOUNDARY
Data lines to specify boundary conditions to preload the structure
*END STEP
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*BOUNDARY, BASE NAME=base2
Data lines to assign degrees of freedom to a secondary base
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, SUBSPACE PROJECTION
Data lines to specify frequency ranges and bias parameters
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*BASE MOTION, DOF=dof, AMPLITUDE=base
*BASE MOTION, DOF=dof, AMPLITUDE=base, BASE NAME=base2
*CLOAD and/or *DLOAD, AMPLITUDE=loadamp
```

Data lines to specify sinusoidally varying, frequency-dependent loads

…

***END STEP**

6.3.10 RESPONSE SPECTRUM ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Dynamic analysis procedures: overview," Section 6.3.1
- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- *RESPONSE SPECTRUM
- *SPECTRUM
- "Configuring a response spectrum procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a spectrum," Section 57.11 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A response spectrum analysis:

- provides an estimate of the peak linear response of a structure to dynamic motion of fixed points ("base motion") or dynamic force;
- is typically used to analyze response to a seismic event;
- assumes that the system's response is linear so that it can be analyzed in the frequency domain using its natural modes, which must be extracted in a previous eigenfrequency extraction step ("Natural frequency extraction," Section 6.3.5);
- can use the high-performance SIM software architecture (see "Using the SIM architecture for modal superposition dynamic analyses" in "Dynamic analysis procedures: overview," Section 6.3.1); and
- is a linear perturbation procedure and is, therefore, not appropriate if the excitation is so severe that nonlinear effects in the system are important.

Response spectrum analysis

Response spectrum analysis can be used to estimate the peak response (displacement, stress, etc.) of a structure to a particular base motion or force. The method is only approximate, but it is often a useful, inexpensive method for preliminary design studies.

The response spectrum procedure is based on using a subset of the modes of the system, which must first be extracted by using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part.

In cases with repeated eigenvalues and eigenvectors, the modal summation results must be interpreted with care. You should add insignificant mass to the structure or perturb the symmetric geometry such that the eigenvalues become unique.

While the response in the response spectrum procedure is for linear vibrations, the prior response may be nonlinear. Initial stress effects (stress stiffening) will be included in the response spectrum analysis if nonlinear geometric effects ("General and linear perturbation procedures," Section 6.1.3) were included in a general analysis step prior to the eigenfrequency extraction step.

The problem to be solved can be stated as follows: given a set of base motions, $\ddot{u}_j^B(t)$ $(j = 1, 2, 3)$, specified in orthogonal directions defined by direction cosines t_k^j $(k = 1, 2, 3)$, estimate the peak value over all time of the response of any variable in a finite element model that is simultaneously subjected to these multiple base motions. The peak response is first computed independently for each direction of excitation for each natural mode of the system as a function of frequency and damping. These independent responses are then combined to create an estimate of the actual peak response of any variable chosen for output, as a function of frequency and damping.

The acceleration history (base motion) is not given directly in a response spectrum analysis; it must first be converted into a spectrum.

Specifying a spectrum

The response spectrum method is based on first finding the peak response to each base motion excitation of a one degree of freedom system that has a natural frequency equal to the frequency of interest. The single degree of freedom system is characterized by its undamped natural frequency, ω_α, and the fraction of critical damping present in the system, ξ_α, at each mode α. The equations of motion of the system are integrated through time to find peak values of relative displacement, relative velocity, and relative or absolute acceleration for the linear, one degree of freedom system. This process is repeated for all frequency and damping values in the range of interest. Plots of these responses are known as displacement, velocity, and acceleration spectra: $S^D(\omega_\alpha, \xi_\alpha)$, $S^V(\omega_\alpha, \xi_\alpha)$, and $S^A(\omega_\alpha, \xi_\alpha)$. The response spectrum can be obtained directly from measured data, as described in "Defining a spectrum using values of S as a function of frequency and damping," below. You can also use a Fortran program to define a spectrum; an example of defining a spectrum from an acceleration record in this way is provided in "Analysis of a cantilever subject to earthquake motion," Section 1.4.13 of the Abaqus Benchmarks Guide.

Alternatively, you can create the required spectrum by specifying an amplitude (time history record), the frequency range, and the damping values for which the spectrum will be built, as described in "Creating a spectrum from a given time history record," below. The spectrum can be used in the subsequent response spectrum analysis, or it can be written to a file for future use.

For each damping value the magnitude of the response spectrum must be given over the entire range of frequencies needed, in ascending value of frequency. Abaqus/Standard interpolates linearly between the values given on a log-log scale. Outside the extremes of the frequency range given, the magnitude is assumed to be constant, corresponding to the end value given. (See "Material data definition," Section 21.1.2, for an explanation of data interpolation.)

Any number of spectra can be defined, and each spectrum must be named. The response spectrum procedure allows up to three spectra to be applied simultaneously to the model in orthogonal physical directions defined by their direction cosines.

Defining a spectrum using values of S as a function of frequency and damping

You can define a spectrum by specifying values for the magnitude of the spectrum; frequency, in cycles per time, at which the magnitude is used; and associated damping, given as a ratio of critical damping.

Input File Usage: To define the spectrum on the data lines:

*SPECTRUM, NAME=*spectrum name*

Repeat this option to define multiple spectra for an analysis.

Abaqus/CAE Usage: To define a spectrum, do the following:

Step, Interaction, or Load module: **Tools→Amplitude→Create**; **Name**: *spectrum name*, **Type**: **Spectrum**

To apply a spectrum to the model, do the following:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Use response spectrum**: select spectrum name for each physical direction in which it should be applied

Specifying the type of spectrum

You can indicate whether a displacement, velocity, or acceleration spectrum is given. The default is an acceleration spectrum.

Alternatively, an acceleration spectrum can be given in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to define a displacement, velocity, or acceleration spectrum:

*SPECTRUM, NAME=*name*, TYPE=DISPLACEMENT
*SPECTRUM, NAME=*name*, TYPE=VELOCITY
*SPECTRUM, NAME=*name*, TYPE=ACCELERATION

Use the following option to define an acceleration spectrum given in *g*-units:

*SPECTRUM, NAME=*name*, TYPE=G, G=*g*

Abaqus/CAE Usage: Use one of the following options to define a displacement, velocity, or acceleration spectrum:

Step, Interaction, or Load module: **Tools→Amplitude→Create**; **Type**: **Spectrum**; **Specification units**: **Displacement**, **Velocity**, or **Acceleration**

Use the following option to define an acceleration spectrum given in *g*-units:

Step, Interaction, or Load module: **Tools→Amplitude→Create**; **Type**: **Spectrum**; **Specification units**: **Gravity**, **Gravity**: *g*

Reading the data defining the spectrum from an alternate input file

The data for the spectrum can be specified in an alternate input file and read into the Abaqus/Standard input file.

Input File Usage: *SPECTRUM, NAME=*name*, INPUT=*file name*

Abaqus/CAE Usage: Step, Interaction, or Load module: **Tools→Amplitude→Create**; **Type**: **Spectrum**; click mouse button 3 while holding the cursor over the data table, and select **Read from File**

Creating a spectrum from a given time history record

If you have a time history of a dynamic event (e.g., acceleration, velocity, displacement), you can build your own spectrum by specifying the record type and the amplitude name that this record represents. If the amplitude record is given with an arbitrarily changing time increment, linear interpolation will be needed for the implicit integration scheme for the dynamic equation of motion for a single degree of freedom system subjected to this record. You can specify the frequency range for the integration scheme and the frequency increment. You can build a spectrum for every fraction of critical damping indicated in the list of damping values.

Input File Usage: *SPECTRUM, CREATE, AMPLITUDE=*amplitude name*,
NAME=*spectrum name*, TIME INCREMENT=*dt*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Specifying the type of spectrum to be created

You can indicate whether a displacement, velocity, or acceleration spectrum is to be created. The default is an acceleration spectrum.

Alternatively, an acceleration spectrum can be created in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to create a displacement, velocity, or acceleration spectrum:

*SPECTRUM, CREATE, TYPE=DISPLACEMENT
*SPECTRUM, CREATE, TYPE=VELOCITY
*SPECTRUM, CREATE, TYPE=ACCELERATION

Use the following option to create an acceleration spectrum in *g*-units:

*SPECTRUM, CREATE, TYPE=G, G=*g*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Specifying the record type that the time history represents

You can indicate whether a displacement, velocity, or acceleration amplitude is specified. The default is an acceleration amplitude.

Alternatively, an acceleration amplitude can be given in *g*-units. In this case you must also specify the value of the acceleration of gravity.

Input File Usage: Use one of the following options to indicate that the amplitude is defined in displacement, velocity, or acceleration units:

```
*SPECTRUM, CREATE, EVENT TYPE=DISPLACEMENT
*SPECTRUM, CREATE, EVENT TYPE=VELOCITY
*SPECTRUM, CREATE, EVENT TYPE=ACCELERATION
```

Use the following option to indicate that an acceleration amplitude is given in *g*-units:

```
*SPECTRUM, CREATE, EVENT TYPE=G, G=g
```

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Creating an absolute or relative acceleration spectrum

When you create an acceleration spectrum from a given time history record, you can create an absolute or relative response spectrum. The default is an absolute spectrum.

Input File Usage:

```
*SPECTRUM, CREATE, TYPE=ACCELERATION, ABSOLUTE
*SPECTRUM, CREATE, TYPE=ACCELERATION, RELATIVE
```

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Generating the list of damping values for the fraction of critical damping

You must provide a list of damping values for the fraction of critical damping to create a spectrum. However, if the damping is evenly spaced between its lower and upper bound, you can automatically generate the list of damping values by providing the start value, end value, and increment for the fraction of critical damping.

Input File Usage:

```
*SPECTRUM, CREATE, DAMPING GENERATE
```

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Writing the generated spectra to an independent file

You can write the generated spectra to an independent file. Otherwise, the generated spectra can be used only within the currently submitted job in subsequent response spectra procedures. You can inspect the generated spectra if you request that model definition data be printed to the data file (see “Model and history definition summaries” in “Output,” Section 4.1.1).

Input File Usage: *SPECTRUM, CREATE, FILE=*file name*

Abaqus/CAE Usage: Creating a spectrum from a given time history record is not supported in Abaqus/CAE.

Estimating the peak values of the modal responses

Since the response spectrum procedure uses modal methods to define a model's response, the value of any physical variable is defined from the amplitudes of the modal responses (the "generalized coordinates"), q_α. The first stage in the response spectrum procedure is to estimate the peak values of these modal responses. For mode α and spectrum k this is

$$(q_\alpha^{max})_k = c_k S_k^D(\omega_\alpha, \xi_\alpha) \sum_j t_j^k \Gamma_{\alpha j},$$

where

q_α is the modal amplitude for mode α;

c_k is a scaling parameter introduced as part of the response spectrum procedure definition for spectrum $S_k^D(\omega_\alpha, \xi_\alpha)$;

$S_k^D(\omega_\alpha, \xi_\alpha)$ is the user-defined value of the spectrum (see "Specifying a spectrum") in direction k interpolated, if necessary, at natural frequency ω_α and the fraction of critical damping ξ_α in mode α;

t_j^k is the jth direction cosine for the kth spectrum; and

$\Gamma_{\alpha j}$ is the participation factor for mode α in direction j (see "Natural frequency extraction," Section 6.3.5).

Similar expressions for $(\dot{q}_\alpha^{max})_k$ and $(\ddot{q}_\alpha^{max})_k$ can be obtained by substituting velocity or acceleration spectra in the above equation.

Combining the individual peak responses

The individual peak responses to the excitations in different directions will occur at different times and, therefore, must be combined into an overall peak response. Two combinations must be performed, and both introduce approximations into the results:

1. The multidirectional excitations must be combined into one overall response. This combination is controlled by the directional summation method, as described below in "Directional summation methods."
2. The peak modal responses must be combined to estimate the peak physical response. This combination is controlled by the modal summation method, as described below in "Modal summation methods."

Depending on the type of base excitation, either modal responses or directional responses are combined first.

Directional summation methods

You choose the method for combining the multidirectional excitations depending on the nature of the excitations.

The algebraic method

If the input spectra in the different directions are components of a base excitation that is approximately in a single direction in space, then for each mode the peak responses in the different spatial directions are summed algebraically by

$$q_\alpha^{max} = \sum_k (q_\alpha^{max})_k.$$

After this summation is performed, the modal responses are summed. (Choosing the method used for modal summation is described below in "Modal summation methods.") Since the directional components are summed first, the subscript k is not relevant and can be ignored in the modal summation equations that follow.

Input File Usage: *RESPONSE SPECTRUM, COMP=ALGEBRAIC, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Single direction** or **Multiple direction absolute sum**

The square root of the sum of the squares directional summation method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in "Modal summation methods." Then, the responses in different excitation directions are combined by

$$(R^i)^{max} = \sqrt{\sum_k \left((R^i)_k^{max}\right)^2}.$$

Input File Usage: *RESPONSE SPECTRUM, COMP=SRSS, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction square root of the sum of squares**

The forty-percent method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in "Modal summation methods." Then, the responses in different excitation directions are combined by the 40% rule recommended by the ASCE 4–98 standard for Seismic Analysis of Safety-Related Nuclear Structures and Commentary, Section 3.2.7.1.2. This method combines the response for all possible combinations of the three components, including variations in sign (plus/minus), assuming that when the maximum response from one component occurs, the response from the other two components is 40% of their maximum value, using one of the following:

$$(R^i)^{max} = \pm[(R^i)_1^{max} \pm 0.4(R^i)_2^{max} \pm 0.4(R^i)_3^{max}],$$

$$(R^i)^{max} = \pm[(R^i)_2^{max} \pm 0.4(R^i)_1^{max} \pm 0.4(R^i)_3^{max}],$$

$$(R^i)^{max} = \pm[(R^i)_3^{max} \pm 0.4(R^i)_2^{max} \pm 0.4(R^i)_1^{max}].$$

Input File Usage: *RESPONSE SPECTRUM, COMP=R40, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction forty percent rule**

The thirty-percent method

If the spectra in different directions represent independent excitations, the modal summation is performed first, as explained below in "Modal summation methods." Then, the responses in different excitation directions are combined by the 30% rule recommended by the ASCE 4–98 standard for Seismic Analysis of Safety-Related Nuclear Structures and Commentary, Section 3.2.7.1.2. This method combines the response for all possible combinations of the three components, including variations in sign (plus/minus), assuming that when the maximum response from one component occurs, the response from the other two components is 30% of their maximum value, using one of the following:

$$(R^i)^{max} = \pm[(R^i)_1^{max} \pm 0.3(R^i)_2^{max} \pm 0.3(R^i)_3^{max}],$$

$$(R^i)^{max} = \pm[(R^i)_2^{max} \pm 0.3(R^i)_1^{max} \pm 0.3(R^i)_3^{max}],$$

$$(R^i)^{max} = \pm[(R^i)_3^{max} \pm 0.3(R^i)_2^{max} \pm 0.3(R^i)_1^{max}].$$

Input File Usage: *RESPONSE SPECTRUM, COMP=R30, SUM=*sum*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Excitations: Multiple direction thirty percent rule**

Modal summation methods

The peak response of some physical variable R^i (a component *i* of displacement, stress, section force, reaction force, etc.) caused by the motion in the αth natural mode excited by the given response spectra in direction k at frequency ω_α with damping ξ_α is given by

$$(R_\alpha^i)_k^{max} = \Phi_\alpha^i (q_\alpha^{max})_k,$$

where Φ^i_α is the ith component of mode α, and there is no sum on α. (In the case of algebraic summation the subscript k is not relevant and can be ignored in this equation and in those that follow.)

There are several methods for combining these peak physical responses in the individual modes, $(R^i_\alpha)^{max}_k$, into estimates of the total peak response, $\left(R^i\right)^{max}_k$. Most of the methods implemented in Abaqus/Standard follow the ASCE 4–98 standard for Seismic Analysis of Safety Related Nuclear Structures and Commentary. The updated documents, "Reevaluation of Regulatory Guidance on Modal Response Combination Methods for Seismic Response Spectrum Analysis" issued in 1999 (NUREG/CR-6645, BNL-NUREG-52276) and "Draft Regulatory Guide" (DG-1127) issued in 2005 contain new recommendations. You are advised to read the new recommendations before choosing a modal summation method from among those described below.

The absolute value method

The absolute value method is the most conservative method for combining the modal responses. It is obtained by summing the absolute values resulting from each mode:

$$(R^i)^{max}_k = \sum_\alpha |(R^i_\alpha)^{max}_k|.$$

This method implies that all of the responses peak simultaneously. It will overpredict the peak response of most systems; therefore, it may be too conservative to help in design.

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=ABS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Absolute values**

The square root of the sum of the squares modal summation method

The square root of the sum of the squares method is less conservative than the absolute value method. It is also usually more accurate if the natural frequencies of the system are well separated. It uses the square root of the sum of the squares to combine the modal responses:

$$(R^i)^{max}_k = \sqrt{\sum_\alpha \left((R^i_\alpha)^{max}_k\right)^2}.$$

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=SRSS

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Square root of the sum of squares**

The Naval Research Laboratory method

The absolute value and square root of the sum of the squares methods can be combined to yield the Naval Research Laboratory method. It distinguishes the mode, β, in which the physical variable has its maximum response and adds the square root of the sum of squares of the peak responses in all other modes to the absolute value of the peak response of that mode. This method gives the estimate:

$$(R^i)_k^{max} = |(R_\beta^i)_k^{max}| + \sqrt{\sum_{\alpha \neq \beta} \left((R_\alpha^i)_k^{max}\right)^2}.$$

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=NRL

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Naval Research Laboratory**

The ten-percent method

The ten-percent method recommended by Regulatory Guide 1.92 (1976) is no longer recommended according to the "Reevaluation of Regulatory Guidance on Modal Response Combination Methods for Seismic Response Spectrum Analysis" document issued in 1999. It is retained here because of its extensive prior use. The ten-percent method modifies the square root of the sum of the squares method by adding a contribution from all pairs of modes α and β whose frequencies are within 10% of each other, giving the estimate:

$$(R^i)_k^{max} = \sqrt{\sum_\alpha \left((R_\alpha^i)_k^{max}\right)^2 + 2\sum_{\alpha<\beta} |\left(R_\alpha^i\right)_k^{max}\left(R_\beta^i\right)_k^{max}|}.$$

The frequencies of modes α and β are considered to be within 10% of each other whenever

$$\frac{\omega_\beta - \omega_\alpha}{\omega_\beta} \leq 0.1, \quad \alpha < \beta.$$

The ten-percent method reduces to the square root of the sum of the squares method if the modes are well separated with no coupling between them.

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=TENP

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Ten percent**

The complete quadratic combination method

Like the ten-percent method, the complete quadratic combination method improves the estimation for structures with closely spaced eigenvalues. The complete quadratic combination method combines the modal response with the formula

$$(R^i)_k^{max} = \sqrt{\sum_\alpha \sum_\beta (R_\alpha^i)_k^{max} \rho_{\alpha\beta} (R_\beta^i)_k^{max}},$$

where $\rho_{\alpha\beta}$ are cross-correlation coefficients between modes α and β, which depend on the ratio of frequencies and modal damping between the two modes:

$$\rho_{\alpha\beta} = \frac{8\sqrt{\xi_\alpha \xi_\beta}(\xi_\alpha + r_{\beta\alpha}\xi_\beta) r_{\beta\alpha}^{3/2}}{(1 - r_{\beta\alpha}^2)^2 + 4\xi_\alpha \xi_\beta r_{\beta\alpha}(1 + r_{\beta\alpha}^2) + 4(\xi_\alpha^2 + \xi_\beta^2) r_{\beta\alpha}^2},$$

where $r_{\beta\alpha} = \omega_\beta / \omega_\alpha$.

If the modes are well spaced, their cross-correlation coefficient will be small ($\rho_{\alpha\beta} << 1$) and the method will give the same results as the square root of the sum of the squares method.

This method is usually recommended for asymmetrical building systems since, in such cases, other methods can underestimate the response in the direction of motion and overestimate the response in the transverse direction.

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=CQC

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Complete quadratic combination**

The grouping method

This method, also known as the NRC grouping method, improves the response estimation for structures with closely spaced eigenvalues. The modal responses are grouped such that the lowest and highest frequency modes in a group are within 10% and no mode is in more than one group. The modal responses are summed absolutely within groups before performing a SRSS combination of the groups. Within the group responses are summed as

$$(R_{gr}^i)_k^{max} = |(R_1^i)_k^{max}| + |(R_2^i)_k^{max}| ... + |(R_n^i)_k^{max}|,$$

for "n" frequencies within any "gr" group and then performing

$$(R^i)_k^{max} = \sqrt{\sum_{gr=1}^{gr=n} \left((R_{gr}^i)_k^{\max} \right)^2}.$$

The above expression includes all the groups; in addition, the group can consist of just one frequency response if this frequency does not have another member that is within the 10% limit.

The ten-percent method will always produce results higher in value than the grouping method.

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=GRP

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Grouping method**

Double sum combination

This method, also known as Rosenblueth's double sum combination (Rosenblueth and Elorduy, 1969), is the first attempt to evaluate modal correlation based on random vibration theory. It utilizes the time duration t_D of strong earthquake motion. The mode correlation coefficients $c_{\alpha\beta}$, which depend also on the frequencies and damping coefficient ξ, lead to the following mode combination:

$$(R^i)_k^{max} = \sqrt{\sum_\alpha \sum_\beta (R^i_\alpha)_k^{max} c_{\alpha\beta} (R^i_\beta)_k^{max}},$$

where

$$c_{\alpha\beta} = \frac{1}{1 + \left(\frac{\omega_{\alpha'} - \omega_{\beta'}}{\xi_{\alpha'}\omega_\alpha + \xi_{\beta'}\omega_\beta}\right)^2},$$

where

$$\omega_{\alpha'} = \omega_\alpha \sqrt{1 - \xi_\alpha^2},$$

$$\xi_{\alpha'} = \xi_\alpha + \frac{2}{t_D \omega_\alpha}.$$

Input File Usage: *RESPONSE SPECTRUM, COMP=*comp*, SUM=DSC

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Summations: Double sum combination**

Selecting the modes and specifying damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS
*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying damping

Damping is almost always specified for a mode-based procedure; see "Material damping," Section 26.1.1. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for an mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

*MODAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

*MODAL DAMPING, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Damping**: **Specify damping over ranges of**: **Modes**

Use the following input to define damping by specifying a frequency range:

Step module: **Create Step**: **Linear perturbation**: **Response spectrum**: **Damping**: **Specify damping over ranges of**: **Frequencies**

Example of specifying damping

Figure 6.3.10–1 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
```
f_1, d_1
f_2, d_2
f_2, d_3
f_3, d_3
f_4, d_4

Rules for selecting modes and specifying damping coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.

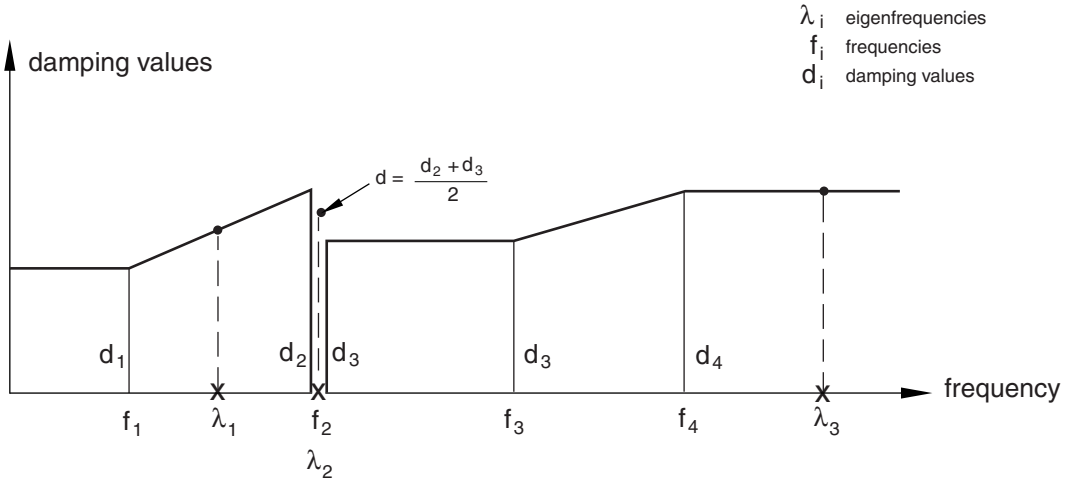


Figure 6.3.10–1 Damping values specified by frequency range.

- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Initial conditions

It is not appropriate to specify initial conditions in a response spectrum analysis.

Boundary conditions

All points constrained by boundary conditions and the ground nodes of connector elements are assumed to move in phase in any one direction. This base motion can use a different input spectrum in each of three orthogonal directions (two directions in a two-dimensional model). You define the input spectra, $S(\omega, \xi)$, as functions of frequency, ω, for different values of critical damping, ξ, as described earlier in "Specifying a spectrum." Secondary bases cannot be used in a response spectrum analysis.

Loads

The only "loading" that can be defined in a response spectrum analysis is that defined by the input spectra, as described earlier. No other loads can be prescribed in a response spectrum analysis.

Predefined fields

Predefined fields, including temperature, cannot be used in response spectrum analysis.

Material options

The density of the material must be defined (“Density,” Section 21.2.1). The following material properties are not active during a response spectrum analysis: plasticity and other inelastic effects, rate-dependent material properties, thermal properties, mass diffusion properties, electrical properties, and pore fluid flow properties—see “General and linear perturbation procedures,” Section 6.1.3.

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard can be used in a response spectrum analysis—see “Choosing the appropriate element for an analysis type,” Section 27.1.3.

Output

All the output variables in Abaqus/Standard are listed in “Abaqus/Standard output variable identifiers,” Section 4.2.1. The value of an output variable such as strain, E; stress, S; or displacement, U, is its peak magnitude.

In addition to the usual output variables available, the following modal variables are available for response spectrum analysis and can be output to the data and/or results files (see “Output to the data and results files,” Section 4.1.2):

GU	Generalized displacements for all modes.
GV	Generalized velocities for all modes.
GA	Generalized accelerations for all modes.
SNE	Elastic strain energy for the entire model per each mode.
KE	Kinetic energy for the entire model per each mode.
T	External work for the entire model per each mode.

Neither element energy densities (such as the elastic strain energy density, SENER) nor whole element energies (such as the total kinetic energy of an element, ELKE) are available for output in response spectrum analysis. However, whole model variables such as ALLIE (total strain energy) are available for modal-based procedures as output to the data and/or results files (see “Output to the data and results files,” Section 4.1.2).

Reaction force output is not supported for response spectrum analysis using eigenmodes extracted using a SIM-based frequency extraction procedure with either the AMS or Lanczos eigensolver. Reaction force output in response spectrum analysis using eigenmodes extracted with the default Lanczos eigensolver provides directional combinations of so-called, modal reaction forces weighted with maximal absolute values of corresponding generalized displacements. Directional and modal combination rules used for the reaction force calculation are the same as for other nodal output variables. Modal reaction forces are calculated in the frequency extraction procedure. They represent static reaction forces calculated for the normal mode shapes. Generally, they cannot adequately represent reaction force in dynamic analysis. For models with diagonal mass and diagonal damping matrices the

superposition of the modal reaction forces can provide a reasonable approximation of a nodal reaction force in mode-based analyses other than response spectrum analysis. In response spectrum analysis the model response can be better represented by requesting section stresses and section forces in structural elements containing supported nodes.

Input file template

```
*HEADING
…
*BOUNDARY
Data lines to define points to be excited by the base motion controlled by the input spectra
*SPECTRUM, NAME=name1, TYPE=type
Data lines to define spectrum "name1" as a function of frequency, ω, and
fraction of critical damping, ξ
*SPECTRUM, NAME=name2, TYPE=type
Data lines to define spectrum "name2" as a function of frequency, ω, and
fraction of critical damping, ξ
**
*STEP
*FREQUENCY
Data line to specify number of modes to be extracted
*END STEP
**
*STEP
*RESPONSE SPECTRUM, COMP=comp, SUM=sum
Data lines referring to response spectra and defining direction cosines
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data lines to define modal damping
*END STEP
```

Additional reference

- Rosenblueth, E., and J. Elorduy, "Response of Linear Systems to Certain Transient Disturbances," Proceedings of the Fourth World Conference on Earthquake Engineering, Santiago, Chile, 1969.

6.3.11 RANDOM RESPONSE ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "General and linear perturbation procedures," Section 6.1.3
- "Dynamic analysis procedures: overview," Section 6.3.1
- *RANDOM RESPONSE
- *PSD-DEFINITION
- *CORRELATION
- "Configuring a random response procedure" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A random response analysis:

- is a linear perturbation procedure that gives the linearized dynamic response of a model to user-defined random excitation; and
- uses the set of modes extracted in a previous eigenfrequency extraction step to calculate the power spectral densities of response variables (stresses, strains, displacements, etc.) and the corresponding root mean square (RMS) values of these same variables.

Random response analysis

Random response analysis predicts the response of a system that is subjected to a nondeterministic continuous excitation that is expressed in a statistical sense by a cross-spectral density matrix. Since the loading is nondeterministic, it can be characterized only in a statistical sense; Abaqus/Standard assumes that the excitation is stationary and ergodic. These statistical measures are explained in detail in "Random response analysis," Section 2.5.8 of the Abaqus Theory Guide. The random response procedure can, for example, be used to determine the response of an airplane to turbulence, the response of a car to road surface imperfections, the response of a structure to jet noise, or the response of a building to an earthquake.

In the most general case the excitation is defined as a frequency-dependent cross-spectral density (CSD) matrix. Except in cases involving moving noise or user subroutine **UCORR**, it is assumed that for a given load case the CSD matrix can be separated into a product of a frequency-dependent, complex-valued scalar function and a frequency-independent, complex-valued, spatial correlation matrix. This assumption helps reduce both the computational time and the amount of required user input but implies that each element of the CSD matrix in a given load case has the same frequency dependence. You can define a different frequency dependence for each load case, but the loads in one load case will not be

correlated with loads in another. Consequently, the system CSD matrix is assembled by simply summing (superimposing) the CSD matrices of the individual load cases.

The frequency-dependent scalar function can be composed of a weighted sum of user-defined, complex-valued, frequency functions. These user-defined frequency functions are specified in units of power spectral density. You assign weights to each frequency function as well as specify properties of the spatial correlation matrix that defines the correlation between excitations at different locations and in different directions for a particular load case. Frequency functions and correlations are discussed below; see "Defining the frequency functions," and "Defining the correlation."

The loads can be defined as concentrated point loads, as distributed loads, as connector element loads, or as base motion excitations, as described below in "Boundary conditions," and "Loads." Multiple, uncorrelated load cases can be defined for concentrated point loads, connector loads, and base motions. Load case 1 is reserved for all distributed loads defined in a particular step. In these steps load case 1 cannot be used for any concentrated point load, connector load, or base motion. Thus, there cannot be any correlation between distributed loads and any other load. Moreover, base motion excitations are assumed to be statistically independent (no correlation) with any other load type even when the same load case number is used. The concentrated point and connector element loads are assumed to be correlated if the same load case number is used.

The random response procedure is based on using a subset of the modes of the system, which must first be extracted by using the eigenfrequency extraction procedure. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part. The model can be preloaded prior to the eigenfrequency extraction. Initial stress effects are included in the stiffness used in the eigenfrequency extraction if geometric nonlinearities are included in the general analysis procedure used to apply the preloads ("General and linear perturbation procedures," Section 6.1.3).

The random response of the model is expressed as power spectral density values of nodal and element variables, as well as their root mean square values.

Defining the frequency range

You specify the frequency range of interest for the random response procedure. The response is calculated at multiple points between the lowest frequency of interest and the first eigenfrequency in the range, between each eigenfrequency in the range, and between the last eigenfrequency in the range and the highest frequency in the range as illustrated in Figure 6.3.11–1. The default number of calculation points in each interval is 20; you can change this number when you define the step. Accurate RMS values can be obtained only if enough points are used so that Abaqus/Standard can integrate accurately over the frequency range. The bias function allows the points on the frequency scale to be spaced closer together at the eigenfrequencies, thus allowing detailed definition of the response close to resonant frequencies and more accurate integration.

Input File Usage: *RANDOM RESPONSE
lower_freq_limit, upper_freq_limit, num_calc_pts, bias_parameter, freq_scale

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Random response**

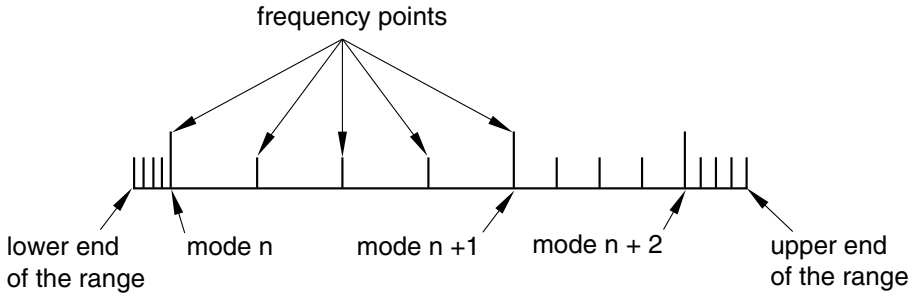


Figure 6.3.11–1 Division of range using modes and 5 calculation points.

The bias parameter

The bias parameter can be used to provide closer spacing of the result points either toward the middle or toward the ends of each frequency interval. Figure 6.3.11–2 shows a few examples of the effect of the bias parameter on the frequency spacing.

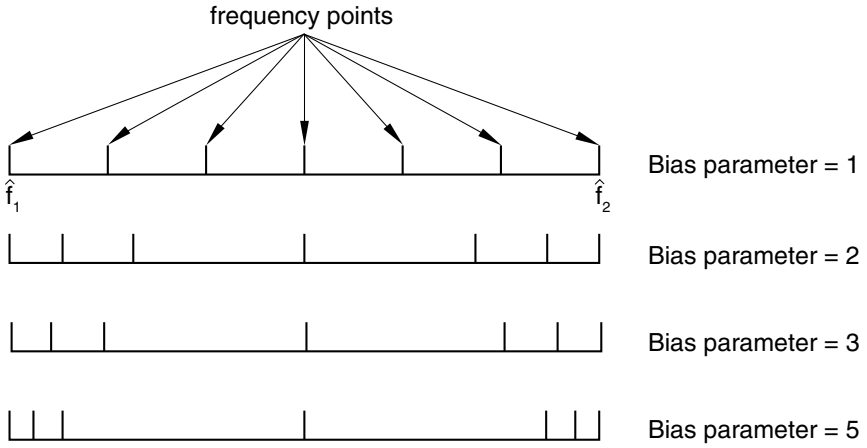


Figure 6.3.11–2 Effect of the bias parameter on the frequency spacing for a number of points $n = 7$.

The bias formula used to calculate the frequency at which results are presented is as follows:

$$\hat{f}_k = \frac{1}{2}(\hat{f}_1 + \hat{f}_2) + \frac{1}{2}(\hat{f}_2 - \hat{f}_1)|y|^{1/p}\mathrm{sign}(y),$$

where

$y \quad = -1 + 2(k-1)/(n-1);$

n is the number of frequency points at which results are to be given;

k is one such frequency point ($k = 1, 2, \ldots, n$);

$\hat{f}_1$ is the lower limit of the frequency interval;

$\hat{f}_2$ is the upper limit of the interval;

$\hat{f}_k$ is the frequency at which the kth results are given;

p is the bias parameter value; and

$\hat{f}$ is the frequency or the logarithm of the frequency, depending on the chosen frequency scale.

A bias parameter, p, that is greater than 1.0 provides closer spacing of the results points toward the ends of each frequency interval (as shown in the examples above), while values of p that are less than 1.0 provide closer spacing toward the middle of each frequency interval. The default value of the bias parameter for random response analysis is 3.0.

Defining the frequency functions

To define the random loading, you specify a frequency function and a cross-correlation definition that refers to the frequency function. The frequency functions are defined as model data (i.e., they are step independent) and must be named. A log-log scale is used in interpolating between the given values.

The type of units in the CSD matrix of the excitation are specified as part of the frequency function definition. The default type is power units. If the CSD matrix of the excitation is due to base motion, the units must be in g units and you should define the acceleration of gravity. Alternatively, decibel units can be specified; this type of units is explained below.

Input File Usage: Use one of the following options to define the frequency function:

*PSD-DEFINITION, NAME=*name*, TYPE=FORCE (default; power units)
*PSD-DEFINITION, NAME=*name*, TYPE=BASE, G=g
*PSD-DEFINITION, NAME=*name*, TYPE=DB, DB REFERENCE=P_{ref}

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power**, **Decibel**, or **Gravity**

Defining the cross-spectral density matrix in decibel units

In Abaqus/Standard the decibel value $db(f)$ is related to the frequency function $P(f)$ by the following full octave band conversion formula:

$$db(f) = 10 \log_{10} \frac{P(f)}{\sqrt{2} P_{\text{ref}} / f_{\text{c}}},$$

where P_{ref} is the user-specified reference power and f_c is the midband frequency (see Table 6.3.11–1). Hence, the frequency function follows from the function defined in decibel units as

$$P(f) = \frac{\sqrt{2}}{f_{\text{c}}} P_{\text{ref}} 10^{db(f)/10}.$$

Table 6.3.11–1 Standard octave bands.

Band number	Band center (frequency, Hz)
1	1.0
2	2.0
3	4.0
4	8.0
5	16.0
6	31.5
7	63.0
8	125.0
9	250.0
10	500.0
11	1000.0
12	2000.0
13	4000.0
14	8000.0
15	16000.0

If you have data in terms of an alternative frequency scale (e.g., one-third octave band), an equivalent full octave band power reference value can be obtained as described in "Random response analysis," Section 2.5.8 of the Abaqus Theory Guide.

$P(f)$ in decibels must be specified as a function of the frequency band; the associated midband frequencies are given in Table 6.3.11–1.

Alternate methods for defining frequency functions

You can define a frequency function in an external file or in a user subroutine.

Defining the frequency function in an external file

The data to define a frequency function can be contained in an external file.

Input File Usage: *PSD-DEFINITION, NAME=*name*, TYPE=*type*, INPUT=*file name*

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power**, **Decibel**, or **Gravity**; **Real**, **Imaginary**, **Frequency**

Defining the frequency function in a user subroutine

Complicated frequency functions can be more easily defined by user subroutine **UPSD** than by entering data directly.

Input File Usage: *PSD-DEFINITION, NAME=*name*, TYPE=*type*, USER

Any data lines given will be ignored if the USER parameter is specified.

Abaqus/CAE Usage: Load module: **Create Amplitude**; **Type: PSD Definition**; **Specification units: Power** or **Gravity**; toggle on **Specify data in an external user subroutine**

Defining the correlation

You define the cross-correlation between the applied nodal loads or base motions. You can also assign scaling (weight) factors to the frequency functions through the cross-correlation definition. Distributed loads are converted to equivalent nodal loads, which are treated as individual point loads with respect to the cross-correlation. The cross-correlation is defined in the random response step and references a particular load case number and frequency function.

Three types of correlation can be defined: correlated, uncorrelated, and moving noise. As many correlations as needed to define the random loading can be specified unless the moving noise type is chosen, in which case only one correlation can appear in the step definition.

- For the correlated type all terms in the cross-spectral density matrix are considered, which implies that the loads on all degrees of freedom within the load case are fully correlated (statistically dependent on each other).
- For the uncorrelated type only diagonal terms in the cross-spectral density matrix are considered, which implies that no correlation exists between the load on one degree of freedom and the load on another. You should exercise caution when choosing the uncorrelated type with distributed loads since the equivalent nodal forces would be uncorrelated with each other (statistically independent).
- For the moving noise type the terms in the correlation matrix depend on the relative position of the points where the loads are applied. This type can be used only in conjunction with concentrated point loads and distributed loads. In addition, the moving noise formulation assumes that the frequency function referenced by the cross-correlation defines a reference power spectral density function of the noise source. (It is a reference power spectral density because it can later be scaled by the magnitude of the loadings specified as distributed, concentrated point, or connector element loads.) Since the power spectral density is real-valued for real-valued variables, the frequency function must not contain imaginary terms when used with the moving noise type of cross-correlation.

Input File Usage: Use one of the following options to define the correlation:

*CORRELATION, TYPE=CORRELATED, PSD=*name*
*CORRELATION, TYPE=UNCORRELATED, PSD=*name*
*CORRELATION, TYPE=MOVING NOISE

For the moving noise type the reference to the power spectral density function must be given on each data line.

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **Correlated** or **Uncorrelated**; **PSD**: *psd_amplitude_name*

Specifying whether the correlation matrix is complex

For correlated or uncorrelated cross-correlations you can specify whether or not both real and imaginary terms will be included in the spatial correlation matrix. This specification does not affect the imaginary terms given for the power spectral density frequency function.

Input File Usage: Use one of the following options:

∗CORRELATION, TYPE=CORRELATED, COMPLEX=YES or NO, PSD=*name*
∗CORRELATION, TYPE=UNCORRELATED, COMPLEX=YES or NO, PSD=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **Correlated** or **Uncorrelated**; **PSD**: *psd_amplitude_name*; **Real**; **Imaginary**

Alternate methods for defining a correlation

You can define a correlation in an external input file or in a user subroutine.

Defining the correlation in an external input file

The data to define a correlation can be contained in an external input file.

Input File Usage: ∗CORRELATION, TYPE=*type*, PSD=*name*, INPUT=*file_name*

Abaqus/CAE Usage: You cannot define a correlation in an external file in Abaqus/CAE.

Defining the correlation in a user subroutine

Simple excitations, such as uncorrelated white noise, are easily defined. Excitations involving more complicated correlations, including cases where the elements of the CSD matrix have different frequency dependencies, can be defined through user subroutine **UCORR**. If the user subroutine is specified, only the load case number must be entered as part of the correlation definition. A user subroutine cannot be used to define a moving noise correlation.

For uncorrelated cross-correlations only the diagonal terms of the correlation matrix specified in **UCORR** will be used. The combination of the cross-correlation with the various kinds of applied loads is discussed in more detail below.

Input File Usage: Use one of the following options:

*CORRELATION, TYPE=CORRELATED, USER, COMPLEX=YES or NO, PSD=*name*
*CORRELATION, TYPE=UNCORRELATED, USER, PSD=*name*

Abaqus/CAE Usage: Load module; **Create Boundary Condition**; **Step:** *random_response_step*; **Category: Mechanical**; **Types for Selected Step: Displacement base motion** or **Velocity base motion** or **Acceleration base motion**; **Correlation** tabbed page: toggle on **Specify correlation**; **Approach**: **User**

Selecting the modes and specifying damping

You can select the modes to be used in modal superposition and specify damping values for all selected modes.

Selecting the modes

You can select modes by specifying the mode numbers individually, by requesting that Abaqus/Standard generate the mode numbers automatically, or by requesting the modes that belong to specified frequency ranges. If you do not select the modes, all modes extracted in the prior eigenfrequency extraction step, including residual modes if they were activated, are used in the modal superposition.

Input File Usage: Use one of the following options to select the modes by specifying mode numbers:

*SELECT EIGENMODES, DEFINITION=MODE NUMBERS
*SELECT EIGENMODES, GENERATE, DEFINITION=MODE NUMBERS

Use the following option to select the modes by specifying a frequency range:

*SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: You cannot select the modes in Abaqus/CAE; all modes extracted are used in the modal superposition.

Specifying damping

Damping is almost always specified for a random response analysis (see "Material damping," Section 26.1.1). If damping is absent, the response of a structure will be unbounded if the forcing frequency is equal to an eigenfrequency of the structure. To get quantitatively accurate results, especially near natural frequencies, accurate specification of damping properties is essential. The various damping options available are discussed in "Material damping," Section 26.1.1. You can define a damping coefficient for all or some of the modes used in the response calculation. The damping coefficient can be given for a specified mode number or for a specified frequency range. When damping is defined by specifying a frequency range, the damping coefficient for a mode is interpolated linearly between the specified frequencies. The frequency range can be discontinuous; the average damping value will be applied for an eigenfrequency at a discontinuity. The damping coefficients are assumed to be constant outside the range of specified frequencies.

Input File Usage: Use the following option to define damping by specifying mode numbers:

*MODAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define damping by specifying a frequency range:

*MODAL DAMPING, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Use the following input to define damping by specifying mode numbers:

Step module: **Create Step**: **Linear perturbation**: **Random response**: **Damping**

Defining damping by specifying frequency ranges is not supported in Abaqus/CAE.

Example of specifying damping

Figure 6.3.11–3 illustrates how the damping coefficients at different eigenfrequencies are determined for the following input:

```
*MODAL DAMPING, DEFINITION=FREQUENCY RANGE
```
f_1, d_1
f_2, d_2
f_2, d_3
f_3, d_3
f_4, d_4

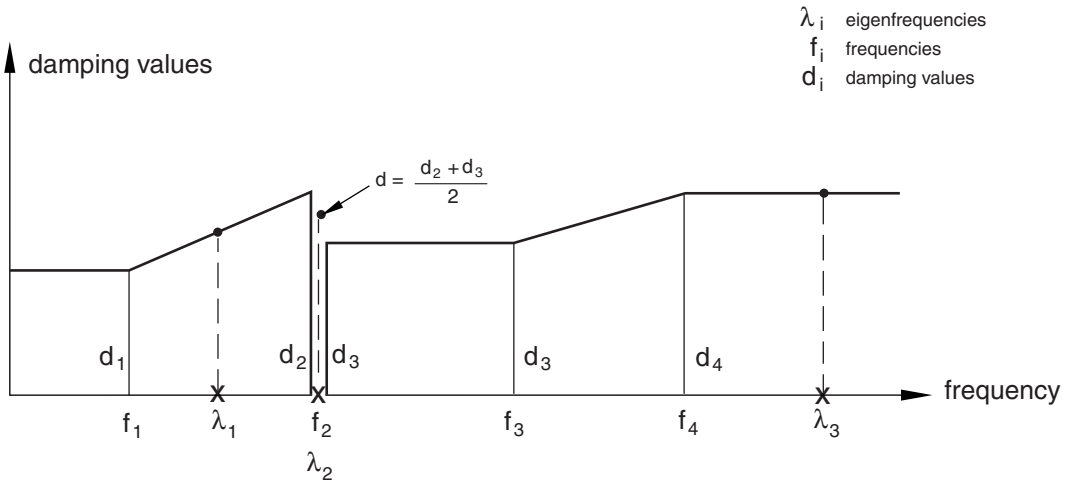


Figure 6.3.11–3 Damping values specified by frequency range.

Rules for selecting modes and specifying damping coefficients

The following rules apply for selecting modes and specifying modal damping coefficients:

- No modal damping is included by default.
- Mode selection and modal damping must be specified in the same way, using either mode numbers or a frequency range.
- If you do not select any modes, all modes extracted in the prior frequency analysis, including residual modes if they were activated, will be used in the superposition.
- If you do not specify damping coefficients for modes that you have selected, zero damping values will be used for these modes.
- Damping is applied only to the modes that are selected.
- Damping coefficients for selected modes that are beyond the specified frequency range are constant and equal to the damping coefficient specified for the first or the last frequency (depending which one is closer). This is consistent with the way Abaqus interprets amplitude definitions.

Initial conditions

It is not appropriate to specify initial conditions in a random response analysis.

Boundary conditions

It is not possible to prescribe nonzero displacements and rotations directly as boundary conditions ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) in mode-based dynamic response procedures. Therefore, in a random response analysis the motion of nodes can be specified only as base motion; nonzero displacement, velocity, or acceleration history definitions given as boundary conditions are ignored, and any changes in the support conditions from the eigenfrequency extraction step are flagged as errors. In addition, any amplitude definitions are ignored in a random response analysis.

The method for prescribing motion in modal superposition procedures is described in "Transient modal dynamic analysis," Section 6.3.7. In random response analysis only a single (primary) base can be defined.

Defining multiple load cases

The excitation defined by the base motion is assigned to numbered load cases. These load cases are then referenced in the cross-correlation definition. The load cases are associated with frequency functions through the reference in the cross-correlation definition. Any number of load cases can be defined, but load case number 1 cannot be used if distributed loads are defined in the same step.

Input File Usage: *BASE MOTION, LOAD CASE=*n*

Abaqus/CAE Usage: Base motions with load cases are not supported in Abaqus/CAE.

Converting base motion excitation to a cross-spectral density matrix

When the excitation is provided by a base motion, it is converted directly into a cross-spectral density matrix projected onto the eigenspace through the modal participation factors (see "Natural frequency extraction," Section 6.3.5), giving

$$S_{\alpha\beta}(f) = \begin{cases} \sum_J g_J^2 P^J(f) \sum_i \sum_j \Gamma_\alpha^i \Gamma_\beta^j \sum_I \Psi_{ij}^{IJ} (2\pi f)^{2\lambda_I} & \text{for } \alpha < \beta, \\ [S_{\beta\alpha}(f)]^* & \text{for } \alpha > \beta, \\ \text{Re}\left(\sum_J g_J^2 P^J(f) \sum_i \sum_j \Gamma_\alpha^i \Gamma_\beta^j \sum_I \Psi_{ij}^{IJ} (2\pi f)^{2\lambda_I} \right) & \text{for } \alpha = \beta, \end{cases}$$

where the superscript * denotes complex conjugate and where

Γ_α^i is the modal participation factor for mode α in excitation direction i (i=1–6);

$P^J(f)$ is the frequency function referenced by the Jth cross-correlation and defined as a function of the frequency f in g units;

Ψ_{ij}^{IJ} is a matrix of weight factors indicating the fraction of P^J to be associated with the correlation between base motion in directions i and j for load case I, as described below;

λ_I $= 0$, 1, or 2, depending on whether the base motion corresponding to load case I is defined in terms of an acceleration spectrum, a velocity spectrum, or a displacement spectrum (see "Transient modal dynamic analysis," Section 6.3.7); and

g_J is the user-specified acceleration of gravity for the same power spectral density frequency function that defines P^J.

If the cross-correlation is defined in user subroutine **UCORR**, Ψ_{ij}^{IJ} is defined in the user subroutine. Otherwise,

Ψ_{ij}^{IJ} $= a^{IJ}$ for all (i, j) if the excitation is correlated or

$= a^{IJ}\delta_{ij}$ if the excitation is uncorrelated,

where a^{IJ} is the (complex) value of the weight factor by which to scale the frequency function P^J used in load case I.

Loads

The loading for random response analysis is defined in general terms by the cross-spectral density matrix $S_{(N,i)(M,j)}(f)$, where f is frequency in cycles per time and the subscripts (N, i) and (M, j) refer to degree of freedom i at node N and degree of freedom j at node M, respectively. Distributed loads are converted to equivalent nodal loads, which—for the formulation of the correlation matrix—are treated in the same way as concentrated point loads. The units of $S_{(N,i)(M,j)}(f)$ are (force)^2 or (moment)^2 per frequency. In addition, any amplitude references on the concentrated point, connector element, or distributed load definitions are ignored in a random response analysis.

Defining multiple load cases

Distributed loads will be assigned automatically to load case number 1. You assign a concentrated point load or connector element load to a numbered load case. Any number of concentrated point and connector element load cases can be specified, but load case number 1 cannot be used for a concentrated point or connector element load if a distributed load is present in the same step. The concentrated point, connector element, and distributed load cases are associated with frequency functions through the cross-correlation definition.

Input File Usage: Use one or more of the following options:

*CLOAD, LOAD CASE=*n*
*CONNECTOR LOAD, LOAD CASE=*m*
*DLOAD

Correlated and uncorrelated loading

For correlated or uncorrelated cross-correlations, the cross-spectral density matrix is defined as

$$S_{(N,i)(M,j)}(f) = \begin{cases} \sum_J P^J(f) \sum_I C^{IJ}_{(N,i)(M,j)} F^I_{(N,i)} F^I_{(M,j)} & \text{for } (N,i) < (M,j), \\ \left[S_{(M,i)(N,j)}(f)\right]^* & \text{for } (N,i) > (M,j), \\ \text{Re}\left(\sum_J P^J(f) \sum_I C^{IJ}_{(N,i)(M,j)} F^I_{(N,i)} F^I_{(M,j)}\right) & \text{for } (N,i) = (M,j), \end{cases}$$

where the superscript * denotes complex conjugate and where

$F^I_{(N,i)}$ is the load magnitude applied to degree of freedom i at node N for load case I;

$P^J(f)$ is the frequency function referenced by the Jth cross-correlation and defined as a function of the frequency f in power (force) or decibel units; and

$C^{IJ}_{(N,i)(M,j)}$ is a matrix of weight factors indicating the fraction of P^J to be associated with the $(N,i)(M,j)$ cross-correlation term for load case I, as described below.

If the cross-correlation is defined in user subroutine **UCORR**, $C^{IJ}_{(N,i)(M,j)}$ is defined in the user subroutine. Otherwise,

$C^{IJ}_{(N,i)(M,j)}$ $= a^{IJ}$ for all (N,M,i,j) if the excitation is correlated or

$= a^{IJ}\delta_{NM}\delta_{ij}$ if the excitation is uncorrelated,

where a^{IJ} is the (complex) value of the weight factor by which to scale the frequency function P^J used in load case I.

Moving noise loading

For moving noise cross-correlations, the cross-spectral density matrix is defined as

$$S_{(N,i)(M,j)}(f) = \sum_I P^I(f) \exp\left(i2\pi f(\mathbf{x}_N - \mathbf{x}_M) \cdot \mathbf{v}^I \frac{1}{\mathbf{v}^I \cdot \mathbf{v}^I}\right) F^I_{(N,i)} F^I_{(M,j)},$$

where

$F^I_{(N,i)}$ is the load magnitude applied to degree of freedom i at node N for load case I;

$P^I(f)$ is the reference power spectral density function associated with load case I and defined as a function of the frequency f in power (force) or decibel units;

$\mathbf{v}^I$ is the velocity vector of noise propagation given for load case I; and

$\mathbf{x}_N$ are the coordinates of node N.

This definition of moving noise implies that the different noise sources have no cross-correlation. Therefore, it is most generally used with only one noise source ($I = 1$ only). In addition, since $P^I(f)$ is the actual power spectral density of the moving noise source, it must be defined as a real-valued function.

Predefined fields

Predefined fields, including temperature, cannot be used in random response analysis.

Material options

As in any dynamic analysis procedure, mass or density ("Density," Section 21.2.1) must be assigned to some regions of any separate parts of the model where dynamic response is required. The following material properties are not active during a random response analysis: plasticity and other inelastic effects, rate-dependent properties, thermal properties, mass diffusion properties, electrical properties, and pore fluid flow properties (see "General and linear perturbation procedures," Section 6.1.3).

Elements

Other than generalized axisymmetric elements with twist, any of the stress/displacement elements in Abaqus/Standard can be used in a random response analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3).

Output

In random response analysis the value of a variable is its power spectral density; all of the output variables in Abaqus/Standard are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1. Power spectral density values are not available for concentrated and distributed loads and for SINV.

Options are also provided in random response analysis to obtain root mean square values for certain variables, as listed below. Total values include base motion, while relative values are measured relative to the base motion.

Element integration point variables:

RS	Root mean square of all stress components.
RE	Root mean square of all strain components.

Element nodal point variables:

MISES	Mises equivalent stress..
RMISES	Root mean square of Mises equivalent stress.

For connector elements, the following element output variables are available:

RCTF	Root mean square of connector total forces.
RCEF	Root mean square of connector elastic forces.
RCVF	Root mean square of connector viscous forces.
RCRF	Root mean square of connector reaction forces.
RCSF	Root mean square of connector friction forces.
RCU	Root mean square of connector relative displacements.
RCCU	Root mean square of connector constitutive displacements.

Nodal variables:

RU	Root mean square values of all components of the relative displacement/rotation at a node.
RTU	Root mean square values of all components of the total displacement/rotation at a node.
RV	Root mean square values of all components of the relative velocity at a node.
RTV	Root mean square values of all components of the total velocity at a node.
RA	Root mean square values of all components of the relative acceleration at a node.
RTA	Root mean square values of all components of the total acceleration at a node.
RRF	Root mean square values of all components of reaction forces and reaction moments at a node.

No energy values are available for a random response analysis.

To reduce the computational cost of random response analysis, you should request output only for selected element and node sets. Abaqus/Standard will calculate the response for only the element and nodal variables requested.

When MISES or RMISES output is requested, Abaqus/Standard stores the needed data in the output database (**.odb**) file and Abaqus/Viewer does the actual computation of the responses. These computations require element stress output in the frequency step preceding the random response step. Note that specifying the name of the element set in the output request in the random response step has no effect on these two output variables. If MISES or RMISES output for a selected set of elements is desired, the name of that element set needs to be specified for the element stress output request in the preceding frequency step. Unlike in other procedures, MISES and RMISES output for random response analysis is computed at the element nodal points and not at the element integration points.

Input file template

```
*HEADING
…
*PSD-DEFINITION, NAME=name, TYPE=type
Data lines to define a frequency function (or PSD function for moving noise)
**
*STEP
*FREQUENCY
Data line to control eigenvalue extraction
*BOUNDARY
Data lines to assign degrees of freedom to the primary base
*END STEP
*STEP
*RANDOM RESPONSE
Data line to specify frequency range of interest
*SELECT EIGENMODES
Data lines to define the applicable mode ranges
*MODAL DAMPING
Data line to define modal damping
*CORRELATION, PSD=name, TYPE=type
Data lines to specify correlation for various excitation load cases (n, p)
*DLOAD
Data lines to define distributed loads
*CLOAD, LOAD CASE=n
Data lines to define concentrated loads in load case n
*CONNECTOR LOAD, LOAD CASE=m
Data lines to define connector loads in load case m
*BASE MOTION, DOF=dof, LOAD CASE=p
Data lines to define base motion p
*END STEP
```

6.4 Steady-state transport analysis

- “Steady-state transport analysis,” Section 6.4.1

6.4.1 STEADY-STATE TRANSPORT ANALYSIS

Product: Abaqus/Standard

References

- "Defining an analysis," Section 6.1.2
- "Symmetric model generation," Section 10.4.1
- *STEADY STATE TRANSPORT
- *SYMMETRIC MODEL GENERATION
- *MOTION
- *TRANSPORT VELOCITY
- *ACOUSTIC FLOW VELOCITY

Overview

A steady-state transport analysis:

- allows for steady-state rolling and sliding solutions including frictional effects and inertia effects;
- allows for steady-state solutions to be obtained directly or by using a quasi-steady-state (pass-by-pass) technique;
- is used to model the interaction between a deformable rolling object and one or more flat, convex, or concave surfaces;
- is based on a specialized analysis capability where the rigid body motion is described in a spatial or Eulerian manner and the deformation in a material or Lagrangian manner;
- allows for one element set in a model to be described in an Eulerian manner while the rest of the elements in the model are treated in a classical Lagrangian manner;
- can be preceded by a static stress analysis or followed by a natural frequency extraction or a complex eigenvalue extraction step;
- uses regular stress/displacement elements and special steady-state rolling and sliding contact pairs;
- is currently available only for three-dimensional analysis with an axisymmetric geometry or a periodic geometry; and
- allows rate-independent, rate-dependent, or history-dependent material behavior.

Steady-state transport analysis

It is cumbersome to model rolling and sliding contact, such as a tire rolling along a rigid surface or a disc rotating relative to a brake assembly, using a traditional Lagrangian formulation since the frame of reference in which motion is described is attached to the material. An observer in this reference frame views even steady-state rolling as a time-dependent process since each point undergoes a repeated

history of deformation. Such an analysis is computationally expensive since a transient analysis must be performed and fine meshing is required along the entire surface of the cylinder.

The steady-state transport analysis capability in Abaqus/Standard uses a reference frame that is attached to the axle of the rotating cylinder. An observer in this frame sees the cylinder as points that are not moving, although the material of which the cylinder is made is moving through those points. This removes the explicit time dependence from the problem—the observer sees a fixed point anywhere, with material moving through it. Thus, the finite element mesh describing the cylinder in this frame of reference does not undergo the large rigid body spinning motion. This means that a fine mesh is required only near the contact zone.

This description can be viewed as a mixed Lagrangian/Eulerian method, where rigid body rotation is described in a spatial or Eulerian manner, and deformation, which is now measured relative to the rotating rigid body, is described in a material or Lagrangian manner. It is this kinematic description that converts the steady-state moving contact problem into a purely spatially dependent simulation.

The steady-state rolling and sliding analysis capability provides solutions that include frictional effects, inertia effects, and material convection for most rate-independent, rate-dependent, and history-dependent material models.

The theory is described in detail in "Steady-state transport analysis," Section 2.7.1 of the Abaqus Theory Guide.

Input File Usage: *STEADY STATE TRANSPORT

Pass-by-pass analysis technique

By default, the steady-state transport analysis procedure in Abaqus/Standard solves for a steady-state rolling and sliding solution directly as a series of increments, with iterations to obtain equilibrium within each increment. The solution in each increment is a steady-state solution corresponding to the loads acting on the structure at that instant. The steady-state transport analysis procedure also provides an alternative technique to obtain a quasi-steady-state rolling and sliding solution as a series of increments, with iterations to obtain equilibrium within each increment. However, the solution in each increment is usually not a steady-state solution corresponding to the loads acting on the structure at that instant. A steady-state solution is generally obtained in several increments, with each increment corresponding to a loading pass through the structure. Each loading pass through the structure can have a different magnitude.

The pass-by-pass analysis technique is relevant only when used with plasticity/creep models. It has no effect on a viscoelastic material model.

Input File Usage: *STEADY STATE TRANSPORT, PASS BY PASS

Unstable problems

Local instabilities (e.g., surface wrinkling, material instability, or local buckling), can occur in a steady-state transport analysis. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while

the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Defining the model

A steady-state transport analysis requires the definition of streamlines. The streamlines are the trajectories that the material follows during transport through the mesh. To meet this requirement, the mesh must be generated using the symmetric model generation capability, which is described in detail in "Symmetric model generation," Section 10.4.1. The three-dimensional model can be created either by revolving an axisymmetric model about its axis of revolution or by revolving a single three-dimensional repetitive sector about its axis of symmetry.

Revolving an axisymmetric cross-section to create a three-dimensional model

You can generate a three-dimensional mesh by revolving a two-dimensional cross-section about a symmetry axis, so that the streamlines follow the mesh lines. In this case the symmetric model generation capability requires a two-dimensional cross-section of the body as a starting point. The cross-section, which must be discretized with axisymmetric finite elements, is defined in a separate input file. A data check analysis must be performed to write the model information to a restart file. The restart file is read in a subsequent run, and a three-dimensional model is generated by Abaqus/Standard by revolving the cross-section about the symmetry axis, starting at a reference plane. Both the symmetry axis and reference plane of the new three-dimensional model can be oriented in any direction in the global coordinate system. The symmetry axis also defines the axis of the spinning body. A nonuniform discretization in the circumferential direction can be specified to allow a finer mesh in the contact region than elsewhere in the model.

Input File Usage: *SYMMETRIC MODEL GENERATION, REVOLVE

Revolving a single three-dimensional sector to create a periodic model

Alternatively, you can generate a periodic three-dimensional mesh by revolving a single three-dimensional sector about its axis of symmetry. To accurately account for the material convection when the streamline integration is performed, the segment angle for the repetitive three-dimensional sector must be chosen small enough.

In this case the symmetric model generation capability requires a single three-dimensional sector as a starting point. The original three-dimensional sector is defined in a separate input file. A data check analysis must be performed to write the model information to a restart file. The restart file is read in a subsequent run, and a three-dimensional periodic model is generated by Abaqus/Standard by revolving the original three-dimensional sector about the symmetry axis. Both the symmetry axis and the original three-dimensional repetitive sector can be oriented in any direction in the global coordinate system. The symmetry axis also defines the axis of the spinning body. There is no restriction that the meshes on the two symmetry surfaces of the repetitive sector match in any way. If the surface meshes on either side of the original sector are not matched completely, constraints will be generated automatically to couple the opposing neighboring surfaces when revolving the original sector to create a periodic model.

Input File Usage: *SYMMETRIC MODEL GENERATION, PERIODIC

Identifying the elements being treated in an Eulerian manner

By default, the rigid body motion in the whole model will be described in a spatial or Eulerian manner. In some cases you may want only part of the model to be treated with the Eulerian method while the rest should be treated with the classical Lagrangian method. One typical example is a disc brake where the disc itself can be treated with the Eulerian method while the brake assembly (brake pads and caliper) is treated with the Lagrangian method. In this case you can specify the name of an element set for which the rigid body motion will be described in an Eulerian manner. The elements that are not included in the element set will be treated with the classical Lagrangian method. Only one Eulerian element set can be specified in the whole model. In a new steady-state transport step or upon restart (see "Restarting an analysis," Section 9.1.1) you can respecify a set of elements to be treated with the Eulerian method even after it has previously been treated with the Lagrangian method and vice versa. Elements treated with the Eulerian method and elements treated with the Lagrangian method cannot be mixed along a streamline.

Input File Usage: *STEADY STATE TRANSPORT, ELSET=*name*

Defining reference frame motions

The deformable and rigid bodies can each be defined in their own moving reference frame in a steady-state rolling and sliding analysis. The motion of these reference frames can be defined quite generally and provides modeling of a spinning deformable body traveling along a straight line, or "cornering" or "precessing" around an axis such as shown in Figure 6.4.1–1. It is also possible to define reference frame motions for rigid bodies, including translations and rotations. The rigid body can be flat, convex, or concave, which allows for modeling of a deformable body in contact with a rotating drum, such as a tire rolling on a drum, or for modeling a tire mounted on a rigid rim.

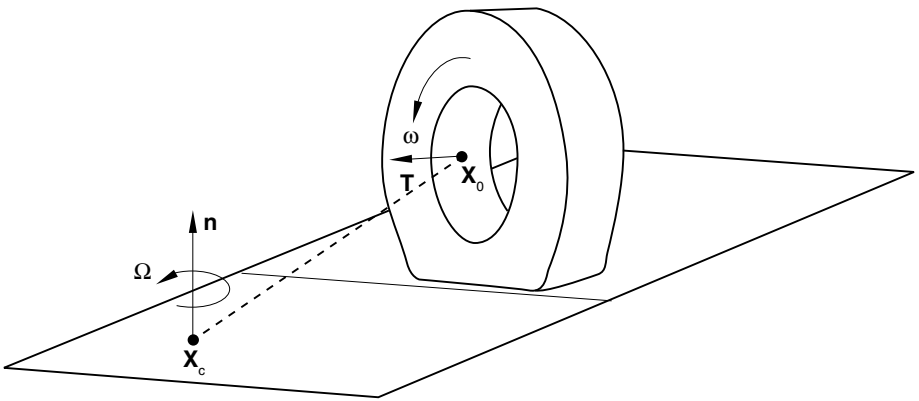


Figure 6.4.1–1 Constant cornering example showing conventions for defining reference frame motions.

When defining different reference frame motions for bodies that interact, you must make sure that the interactions are indeed steady. For example, for a planar rigid surface the relative reference frame

motion must be tangential to the rigid surface, and for a body of revolution the relative reference frame motion must be rotation around its axis. Convergence difficulties will persist if the interactions are not steady.

Spinning motion

The spinning motion of the deformable body around its own axis is described by a user-specified angular velocity, $\boldsymbol{\omega}$ (see Figure 6.4.1–1). This angular velocity defines the transport of material through the mesh; you define the magnitude of the spinning rotation, ω. The axis of revolution is the symmetry axis used for generating the mesh as described in "Defining the model." The transport velocity must be defined for all nodes on the spinning body. The magnitude of the angular velocity can also be defined with user subroutine **UMOTION**.

The transport velocity can also be applied to a rigid body based on a three-dimensional surface of revolution. In that case the velocity is applied to the rigid body reference node to describe the transport of the (rigid) material relative to the reference node. Abaqus/Standard assumes that the rigid body spins around the axis of revolution of the rigid body. This option can, for example, be applied to the rigid body representing the rim on which a tire is mounted.

Abaqus/Standard will automatically update the position and orientation of the rotation axis to the current configuration in a large-displacement analysis, such as in the case where a prescribed load applied to the reference node of a rotating rigid drum maintains the contact pressure between the tire and drum or the case where a camber angle is applied to the axle of the deformable body.

Input File Usage: Use either of the following options:

```
*TRANSPORT VELOCITY
*TRANSPORT VELOCITY, USER
```

Defining a reference frame for translational or rotational motion

The rotating deformable body is also associated with a reference frame. This reference frame can either translate or rotate with respect to the fixed global reference frame. Similarly, each rigid body must be defined in a reference frame that is either fixed, translates, or rotates. For example, to associate straight line travel at ground velocity, $\mathbf{c}$, with a spinning deformable body, the deformable body can be defined in a reference frame translating at velocity $\mathbf{c}$ and the rigid surface can be defined in a fixed reference frame. Alternatively, the deformable body can be defined in a reference frame that does not translate and the rigid body can be defined in a frame translating at velocity $-\mathbf{c}$. Another example is a deformable body precessing along a circular path such as shown in Figure 6.4.1–1. In such a case a rotating frame is associated with the deformable body that defines the precession axis and angular velocity, while the rigid body is defined in a fixed reference frame. All components of the reference frame motion are zero unless otherwise specified; components of the reference frame motion cannot be treated as unknowns to be determined by the simulation.

You can apply a specified motion of the reference frame to all nodes of the deformable body or to the reference node of a rigid body. A translating reference frame is defined by specifying the components of the velocity vector, $\mathbf{c}$. A rotating reference frame is defined by specifying the magnitude of an angular rotation velocity, Ω, and the position and orientation of the axis of rotation in the current configuration.

The position and orientation of the axis are applied at the beginning of the step and remain fixed during the step.

Input File Usage: Use the following option to define the motion of a translating reference frame:
*MOTION, TRANSLATION
Use the following option to define the motion of a rotating reference frame:
*MOTION, ROTATION

Contact conditions

Abaqus/Standard provides contact between a rigid surface and deformable body moving with different velocities, such as contact between a rolling tire and the ground, as well as contact between surfaces moving with the same velocity, such as the contact between the bead and rim in a tire analysis. Abaqus/Standard also provides contact between two deformable bodies moving with the same velocity, such as the contact between the tread blocks on a tire surface, as well as contact between two deformable bodies moving with different velocities, such as the contact between a disc and brake assembly.

Contact between a rigid surface and a deformable body moving with different velocities

The rigid surface can be either an analytical surface or made from rigid elements. When the master and slave surfaces move with different velocities, you will normally select to use a Coulomb friction law that assumes that slip occurs if the frictional stress

$$\tau_{eq} = \sqrt{\tau_1^2 + \tau_2^2}$$

is equal to the critical stress $\tau_{crit} = \mu p$, where τ_1 and τ_2 are the shear stresses on the contact plane, μ is the friction coefficient, and $\boldsymbol{p}$ is the contact pressure. No slip occurs when $\tau_{eq} < \tau_{crit}$. For steady-state transport the condition of no slip is approximated in Abaqus/Standard by stiff "viscous" behavior

$$\tau_\alpha = \kappa_s \dot{\gamma}_\alpha,$$

where $\dot{\gamma}_\alpha$ are the tangential slip velocities that depend on deformation along a streamline and

$$\kappa_s = \frac{\mu p}{2 F_f \omega R}$$

is the "stick viscosity," $\boldsymbol{R}$ is the radius of the cylinder, and F_f is a user-defined slip tolerance for which the default is 0.005. Using a larger slip tolerance makes convergence of the solution more rapid at the expense of solution accuracy. Using a smaller slip tolerance imposes the "no relative motion" constraint more accurately but may slow convergence. The default value provides a conservative balance between efficiency and accuracy for rolling contact problems.

Since this frictional model used for steady-state rolling is different from the frictional models used with other analysis procedures in Abaqus/Standard, discontinuities may arise in the solutions between a steady-state transport analysis and any other analysis procedure, such as a static footprint analysis. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to a steady-

state rolling analysis use a zero coefficient of friction. You can then modify the friction properties in the steady-state transport analysis step to use the desired friction coefficient (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).

This frictional model is more relevant in a tire analysis since the velocity of the rotating tire strongly depends on the deformation gradients along a streamline on the contact surface. The solution state at a material point depends on the solution of neighboring points, and convective effects must be considered. However, since the deformation gradients along a streamline on the contact surface are small in a disc brake analysis, a simplified frictional model, which ignores the convective effect on the contact surface, can be used. Such a frictional model is discussed in the following section.

Contact between two deformable bodies moving with different velocities

When the slave and master surfaces rotate with different velocities, such as contact between a disc and brake assembly, slip will develop between the two deformable surfaces. The transport velocity ("Spinning motion") and the motion of a reference frame ("Defining a reference frame for translational or rotational motion") can be defined in a steady-state transport analysis procedure to model the steady-state frictional sliding between two deformable bodies that are moving with different velocities. In this case it is assumed that the slip rate simply follows from the difference in velocities specified by the transport velocity and the motion of the reference frame and is independent of the deformation gradient along a streamline or the nodal displacements on the contact surface. No convective effects are considered between the contact surfaces, and the frictional stress does not depend on any history effects. Hence, the frictional stress is given by

$$\tau_\alpha = \mu p \frac{\dot{\gamma}_\alpha}{|\dot{\gamma}_\alpha|} = \mu p t_\alpha,$$

where μ is the friction coefficient, p is the contact pressure, t_α are the local tangent directions, and $\dot{\gamma}_\alpha$ are the slip velocities that are defined by the transport velocity and the motion of the reference frame. If no velocity or the same velocity are defined at contact nodes with friction, sticking conditions are applied automatically. The friction model is described in detail in "Coulomb friction," Section 5.2.3 of the Abaqus Theory Guide.

Such a simplified frictional model is relevant only in a disc brake analysis. It should be used with care in a rolling tire analysis where deformation gradients on the contact surface are significant.

Since this frictional behavior is different from the frictional models used with other analysis procedures in Abaqus/Standard, discontinuities may arise in the solutions between a steady-state transport analysis and any other analysis procedure. An example is the discontinuity that occurs between the initial preloading of the disc pads in a disc brake system and the subsequent braking analysis where the disc spins with a prescribed rotation. To ensure a smooth transition in the solution, it is recommended that all analysis steps prior to a steady-state analysis use a zero coefficient of friction (see "Including friction properties in a contact property definition" in "Frictional behavior," Section 37.1.5). You can then increase the friction coefficient to the desired value in the steady-state transport analysis (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).

Contact between surfaces spinning with the same angular velocity

When the slave and master surfaces rotate with the same angular velocity, such as the surface between the bead and rim in a tire analysis, no relative velocity develops between the surfaces. In such a case, frictional stresses develop as a reaction between the bodies. Abaqus/Standard will automatically determine that the slave and master surface rotate with the same speed and apply the standard Coulomb friction model, which is described in detail in "Frictional behavior," Section 37.1.5.

When the standard Coulomb friction model is used in a reference frame that implies flow of material through the mesh, convective effects must be considered. However, Abaqus/Standard assumes that no convective effects are present between surfaces during steady-state transport analysis. In other words, Abaqus/Standard assumes that the frictional stress at a point depends on the history of deformation in the Lagrangian reference frame and ignores any history effects that may occur as a result of the deformation that the point experiences during the spinning motion. The assumption that the frictional stress does not depend on history effects during rolling is valid for modeling contact between a tire bead and rim where relative slip occurs only during rim mounting in a static analysis prior to the steady-state transport analysis. When slip occurs during the steady-state transport analysis, the solution obtained is no longer the correct steady-state solution because convective effects are ignored. To ensure that no slip takes place between the surfaces during steady-state rolling, it is recommended that you modify the friction properties in the steady-state transport analysis step to activate rough friction (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).

Incrementation

Abaqus/Standard uses Newton's method to solve the nonlinear equilibrium equations. The nonlinearities in a steady-state transport analysis arise from large-displacement effects, material nonlinearity, and boundary nonlinearities such as contact and friction. If geometrically nonlinear behavior is expected other than the large rigid body rotation associated with the steady-state motion, the step definition should include nonlinear geometric effects.

The steady-state rolling and sliding solution must often be obtained as a series of increments, with iterations to obtain equilibrium within each increment. If the direct steady-state solution technique is used, the solution in each increment is a steady-state solution corresponding to the loads acting on the structure at that instant. If the pass-by-pass steady-state solution technique is used, the solution in each increment is usually not a steady-state solution corresponding to the loads acting on the structure at that instant. In this case a steady-state solution is generally obtained in several increments, with each increment corresponding to a loading pass through the structure.

Since Newton's method has a finite radius of convergence, too large an increment in the applied load can prevent any solution from being obtained because the current steady-state solution is too far away from the new steady-state equilibrium solution that is being sought: it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic incrementation

In most cases the default automatic incrementation scheme is preferred because it will select increment sizes based on computational efficiency.

Input File Usage: *STEADY STATE TRANSPORT

Direct incrementation

Direct user control of the increment size is also provided because if you have considerable experience with a particular problem, you may be able to select a more economical approach.

Input File Usage: *STEADY STATE TRANSPORT, DIRECT

Using the maximum number of iterations to determine the increment size

The solution to an increment can be accepted after the maximum number of iterations allowed has been completed (as defined in "Commonly used control parameters," Section 7.2.2), even if the equilibrium tolerances are not satisfied. This approach is not recommended; it should be used only in special cases when you have a thorough understanding of how to interpret results obtained in this way. Very small increments and a minimum of two iterations are usually necessary in this case.

Input File Usage: *STEADY STATE TRANSPORT, DIRECT=NO STOP

Convergence in a steady-state transport analysis

The steady-state transport procedure may experience convergence difficulties in certain situations that are described below.

Convergence issues with friction

The frictional forces that develop on the contact surface as a result of steady-state rolling are functions of the spinning angular velocity, $\boldsymbol{\omega}$, and the traveling straight line velocity, $\mathbf{c}$, or cornering velocity, $\boldsymbol{\Omega}$. When these frictional forces are large, convergence of Newton's method becomes difficult. Convergence problems in Abaqus/Standard are usually resolved by taking a smaller load increment. However, contact forces due to steady-state rolling usually do not reduce when the magnitudes of the velocities are reduced. For example, if a spinning object is prevented from moving ($\mathbf{c} = 0.0$), full slipping conditions will develop over the entire contact zone for all values of spinning angular velocity $\boldsymbol{\omega} > 0.0$. Consequently, the frictional force remains constant for all $\boldsymbol{\omega} > 0.0$ (provided that the normal force remains constant), so that smaller increments in the velocities ($\boldsymbol{\omega}, \boldsymbol{\Omega}, \mathbf{c}$) do not reduce the magnitude of the frictional forces and, hence, do not overcome convergence difficulties.

To provide for convergence through the use of smaller increments in such cases, the friction coefficient can be increased from zero to the desired value over the analysis step. This is accomplished by setting the initial friction coefficient for the model to zero (see "Including friction properties in a contact property definition" in "Frictional behavior," Section 37.1.5), then increasing the friction coefficient to its final value in the steady-state transport analysis step (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).

Convergence issues with the Mullins effect material model

If the Mullins effect material model is included in the material definition (see "Mullins effect," Section 22.6.1), there could be a strong discontinuity in the response of a structure in transitioning from

a static (non-rolling) state to a steady-state rolling state. This discontinuity is due to the damage that occurs during the transient response (such as the damage that occurs as the structure undergoes its first revolution after static preloading). Since the transient response is not modeled during a steady-state transport analysis, the resulting discontinuity in the response can lead to convergence problems. The damage associated with the Mullins effect is independent of the angular speed of rotation: as a result, time increment cutbacks do not resolve the convergence problems. The Mullins effect can be ramped up over the time period of the step in these situations to obtain a converged solution. In such a case the change in response due to damage is applied gradually over the step. The solution at the end of the step corresponds to the fully damaged material; solutions during the step correspond to a partially damaged material and are, therefore, physically meaningless. Thus, it is recommended that in going from a static to a steady-state rolling solution, a do-nothing step at a low angular speed of rotation be first carried out with the Mullins effect ramped on. This facilitates resolution of the discontinuity in a gradual manner. The do-nothing step can then be followed by the regular steady-state transport step with the Mullins effect applied instantaneously at the beginning of the step. This approach is illustrated in "Analysis of a solid disc with Mullins effect and permanent set," Section 3.1.7 of the Abaqus Example Problems Guide.

Input File Usage: ∗STEADY STATE TRANSPORT, MULLINS=RAMP *or* STEP (default)

Convergence issues with streamline integration in plasticity/creep models

Although in principle any material point along a streamline can be used as a starting point for the streamline integration when material convective calculations are performed, Abaqus/Standard always uses the material points in the original sector or the material points in the original cross-section as starting points for the streamline integration in a model with periodic geometry or axisymmetric geometry, respectively.

If the pass-by-pass solution technique is used, after an increment has been performed for all the streamlines, Abaqus/Standard will automatically use the state obtained at the end of the streamline as the starting state for the streamline integration in the subsequent increment. This iterative process is repeated for each increment until a steady-state solution is reached.

If the direct steady-state solution technique is used, several local iterations are usually required for each streamline, with a local iteration corresponding to an integration over a closed loop streamline. After a local iteration has been performed for a streamline, Abaqus/Standard will check to see if the steady-state condition is satisfied for the streamline. This is best measured by ensuring the differences between the stresses/strains at the starting point of the streamline obtained before and after the iteration are sufficiently small. If the steady-state condition is not satisfied for the streamline, Abaqus/Standard will automatically use the state obtained at the end of the previous local iteration as the starting state for the streamline integration in the subsequent local iteration. This iterative process is repeated until a steady-state solution is reached for all the streamlines.

To improve the rate of convergence, it is recommended that you apply loads on elements or nodes away from the starting points of the streamlines.

Convergence issues with unconstrained mesh motion

Unconstrained rigid body modes of the mesh motion will cause convergence problems for a steady-state transport analysis, similar to convergence problems for unconstrained rigid body modes in a static analysis. Friction cannot be relied on to restrict rigid body modes in a steady-state transport analysis, because frictional stresses depend on relative material velocities rather than relative nodal displacements for steady-state transport. Restricting the (steady-state) material velocity does not restrict nodal displacements for steady-state transport analyses. The material velocity includes effects of material flowing through the mesh and is governed by the spinning motion (see "Spinning motion"), reference frame motions (see "Defining a reference frame for translational or rotational motion"), and nodal positions relative to the spinning axis.

Consider the examples shown in Figure 6.4.1–2. End-on views are shown in Figure 6.4.1–2, so the axial direction (spinning axis) is horizontal in this figure. The axial component of the reference frame motion is zero for all three of these cases (either by explicit specification or implicitly by default). Since the material velocity in the axial direction at steady state is zero for both bodies (according to the reference frame motion), the frictional force in the axial direction will remain zero for all three of these cases, which may not be intuitive. The first case shown in Figure 6.4.1–2 (a) involves a planar interface and a boundary condition in the axial direction. At the time the rolling body reaches steady state, the axial motion has stopped, the axial friction force is zero, and the reaction force associated with the boundary condition is zero (which also may not be intuitive).

The second case in Figure 6.4.1–2 (b) has a planar interface and an applied force in the axial direction. The axial frictional force for the steady-state solution remains zero, as already discussed, so no axial force arises to counter the applied force. Therefore, Abaqus/Standard will not provide a converged solution in this case, which is an example of unconstrained rigid body mesh motion. The third case shown in Figure 6.4.1–2 (c) is like the second case, except a "curb" has been added to the rigid surface. In this case, a contact force in the normal direction occurs at the location of the "curb," which counters the applied force, so the analysis is able to converge.

As a real world example, consider a car traveling along a straight planar road with a truck moving parallel to the car, applying a constant concentrated force that pushes the car sideways. With zero toe angle on the car's front wheels (that is, the wheels are exactly aligned with the longitudinal axis of the car), steady-state motion is impossible, and the car will eventually slip off the road. To resist the push in steady-state motion, the car wheels need to be aligned with the proper toe angle.

Initial conditions

Initial values of stresses, temperatures, field variables, solution-dependent state variables, etc. can be specified. "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the available initial conditions.

Boundary conditions

Boundary conditions can be applied to any of the displacement or rotation degrees of freedom (1–6). (See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1, for details of applying

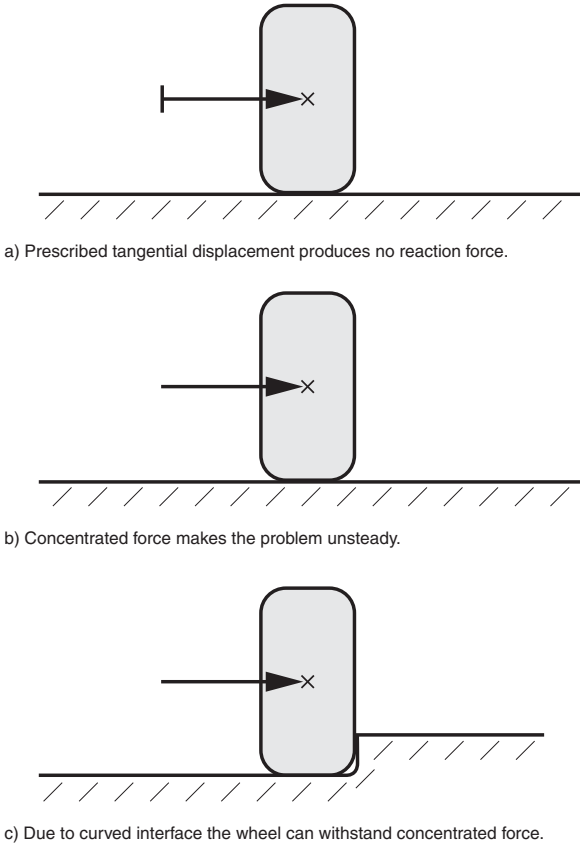


Figure 6.4.1–2 Displacement boundary condition versus concentrated force.

boundary conditions to rotation degrees of freedom when large rotation will occur.) During the analysis prescribed boundary conditions can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2). Boundary conditions restrict the mesh motion but do not restrict the transport of material through the mesh (due to the spinning motion discussed in "Spinning motion").

Loads

Loading in a steady-state transport analysis includes the motion of the structure, inertia (d'Alembert) forces due to motion, concentrated loads, distributed pressures, and body forces.

Inertia effects

The motion of the deformable body gives rise to inertia (d'Alembert) forces that can be included. These forces include centrifugal and Coriolis effects.

The density of the material must be defined in the material description. At higher rotational velocities, inertia forces can give rise to instabilities in the form of standing waves, which are likely to prevent convergence of the Newton algorithm.

Input File Usage: Use the following option to include inertia forces:

*STEADY STATE TRANSPORT, INERTIA=YES

Inertia loads for tetrahedral elements

Inertia loads for tetrahedral elements C3D4, C3D10, C3D10HS, and C3D10M are not taken into account in a steady-state transport analysis. Tetrahedral elements will appear only in a periodic model created by revolving a three-dimensional sector that contains tetrahedral elements. Tetrahedral elements will not appear in an axisymmetric model created by revolving a two-dimensional cross-section about a symmetry axis. See "Symmetric model generation," Section 10.4.1, for details.

Other prescribed loads

The following loads can be prescribed in a steady-state transport analysis, as described in "Concentrated loads," Section 34.4.2:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6).
- Distributed pressure forces or body forces can be applied; the distributed load types available with particular elements are described in Part VI, "Elements."

In most cases such loads should be applied around the whole circumference of the body; a load on a single point or element corresponds to a spatially fixed load, which in most cases is not realistic.

Predefined fields

The following predefined fields can be specified in a steady-state transport analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in a steady-state transport analysis, nodal temperatures can be specified as a predefined field. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values only affect field-variable-dependent material properties, if any.

Material options

Since the steady-state transport capability uses a kinematic description that implies flow of material through the mesh, convective effects must be considered for the material response. Most material models that describe mechanical behavior (including user-defined materials) are available for use in a steady-state transport analysis. In particular, history-dependent viscoelasticity ("Time domain viscoelasticity," Section 22.7.1), history-dependent Mullins effect ("Mullins effect," Section 22.6.1),

classical metal plasticity (“Classical metal plasticity,” Section 23.2.1), rate-dependent yield (“Rate-dependent yield,” Section 23.2.3), rate-dependent creep (“Rate-dependent plasticity: creep and swelling,” Section 23.2.4), two-layer viscoplasticity (“Two-layer viscoplasticity,” Section 23.2.11), and models defined using the parallel rheological framework (“Parallel rheological framework,” Section 22.8.2) can all be used during a steady-state transport analysis. However, models defined within the parallel rheological framework cannot include plasticity.

The following material properties are not active during a steady-state transport analysis: thermal properties (except for thermal expansion), mass diffusion properties, electrical properties, and pore fluid flow properties.

Abaqus/Standard also provides the ability to obtain the fully relaxed long-term elastic or elastic-plastic solution during a steady-state transport analysis if the material description includes viscoelastic or viscoplastic material properties. If the material description includes viscoelastic material properties, the long-term solution will ignore the material convection calculations. If the two-layer viscoplastic material model is used, the long-term solution will include only the material convection calculations based on the long-term response of the elastic-plastic network.

Input File Usage: *STEADY STATE TRANSPORT, LONG TERM

Choosing an appropriate material model

Since material points in a spinning and sliding body undergo repeated loading/unloading cycles, an appropriate material model must be chosen to characterize the response correctly under such loading conditions. The use of plasticity material models with isotropic type hardening is generally not recommended since they will continue to harden during cyclic loading, which may lead to a large number of iterations until the steady-state solution is reached. Kinematic hardening plasticity models should be used to model the inelastic behavior of materials that are subjected to repeated loading.

For rate-dependent creep, the two-layer viscoplasticity model is recommended (“Two-layer viscoplasticity,” Section 23.2.11) for modeling the response of materials with significant time-dependent behavior as well as plasticity at elevated temperatures.

For history-dependent viscoelasticity, it is more appropriate to use cyclic (frequency domain) test data to calibrate the time-domain viscoelastic material model for steady-state transport analysis. The cyclic experiments should be performed in the frequency range anticipated in the rolling simulation. Abaqus/Standard internally converts the frequency domain storage and loss modulus data into a time-domain (Prony series) representation. This data conversion capability is described in detail in “Time domain viscoelasticity,” Section 22.7.1.

Analysis steps prior to a steady-state transport analysis

It is recommended that the solutions in any analysis step prior to a steady-state transport analysis, such as a static footprint or preloading solution, be based on the long-term elastic moduli or the long-term elastic-plastic response if viscoelastic or viscoplastic material properties are used (for example, see “Static stress analysis,” Section 6.2.2). The long-term solution provides a smooth transition between a static analysis and a slow rolling or sliding steady-state transport analysis.

Material convection in nonlinear analysis

When material convection is included in the steady-state transport solution, Abaqus/Standard uses an approximate Jacobian matrix in the Newton solution of the nonlinear equilibrium equations. The rate of convergence in such a case is no longer quadratic but depends strongly on the severity of the nonlinearities. It is often necessary to adjust the default solution controls ("Commonly used control parameters," Section 7.2.2) to obtain a steady-state transport solution when material convection is considered.

Elements

Most of the three-dimensional stress/displacement elements in Abaqus/Standard can be used in a steady-state transport analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). When the three-dimensional model is generated from an axisymmetric cross-section, the element type used in the two-dimensional model determines the element type in the three-dimensional model. The correspondence between the two-dimensional and three-dimensional element types is described in "Symmetric model generation," Section 10.4.1. If the three-dimensional periodic model is generated from a single three-dimensional sector, any of the stress/displacement elements in Abaqus/Standard can be used.

Output

The element output available for a steady-state transport analysis includes stress, strain, energies, and the values of state, field, and user-defined variables. The nodal output available includes displacements, velocities, reaction forces, and coordinates. The contact output variable CSLIP contains steady-state slip rates for the steady-state transport procedure, unlike the usual definition of this variable. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Limitations

The steady-state transport analysis capability has several limitations.

- The deformable structure must be a full 360° cylindrical body of revolution. Convective boundary conditions are not available to model segments of a cylinder.
- The capability is not available in two dimensions.
- Only one deformable spinning body is permitted. The symmetric model generation capability must be used to generate the deformable body ("Symmetric model generation," Section 10.4.1).
- Material model definitions for models defined using the parallel rheological framework ("Parallel rheological framework," Section 22.8.2) cannot include plasticity.

Input file template

```
*HEADING
…
*SYMMETRIC MODEL GENERATION, REVOLVE
```

Data lines to define model generation
***SURFACE INTERACTION**
***FRICTION**
Specify zero friction coefficient
******
***STEP**
***STATIC**
Data lines to define analysis steps prior to transport analysis
***END STEP**
…
***STEP**
***STEADY STATE TRANSPORT**
Data line to define incrementation
***CHANGE FRICTION**
***FRICTION**
Data lines to redefine friction coefficient
***BOUNDARY**
Data lines to define boundary conditions
***TRANSPORT VELOCITY**
Data lines to define spinning angular velocity
***MOTION, TRANSLATION or ROTATION**
Data lines to define traveling velocity or cornering rotational velocity
***EL PRINT and/or *NODE PRINT**
Data lines to request output variables
***END STEP**

6.5 Heat transfer and thermal-stress analysis

- “Heat transfer analysis procedures: overview,” Section 6.5.1
- “Uncoupled heat transfer analysis,” Section 6.5.2
- “Fully coupled thermal-stress analysis,” Section 6.5.3
- “Adiabatic analysis,” Section 6.5.4

6.5.1 HEAT TRANSFER ANALYSIS PROCEDURES: OVERVIEW

Abaqus can solve the following types of heat transfer problems:

- **Uncoupled heat transfer analysis:** Heat transfer problems involving conduction, forced convection, and boundary radiation can be analyzed in Abaqus/Standard or Abaqus/CFD. See "Uncoupled heat transfer analysis," Section 6.5.2. In these analyses the temperature field is calculated without knowledge of the stress/deformation state or the electrical field in the bodies being studied. Pure heat transfer problems can be transient or steady-state and linear or nonlinear.
- **Sequentially coupled thermal-stress analysis:** If the stress/displacement solution is dependent on a temperature field but there is no inverse dependency, a sequentially coupled thermal-stress analysis can be conducted in Abaqus/Standard. Sequentially coupled thermal-stress analysis is performed by first solving the pure heat transfer problem, then reading the temperature solution into a stress analysis as a predefined field. See "Sequentially coupled thermal-stress analysis," Section 16.1.2. In the stress analysis the temperature can vary with time and position but is not changed by the stress analysis solution. Abaqus allows for dissimilar meshes between the heat transfer analysis model and the thermal-stress analysis model. Temperature values will be interpolated based on element interpolators evaluated at nodes of the thermal-stress model.
- **Fully coupled thermal-stress analysis:** A coupled temperature-displacement procedure is used to solve simultaneously for the stress/displacement and the temperature fields. A coupled analysis is used when the thermal and mechanical solutions affect each other strongly. For example, in rapid metalworking problems the inelastic deformation of the material causes heating, and in contact problems the heat conducted across gaps may depend strongly on the gap clearance or pressure.

  Both Abaqus/Standard and Abaqus/Explicit provide coupled temperature-displacement analysis procedures, but the algorithms used by each program differ considerably. In Abaqus/Standard the heat transfer equations are integrated using a backward-difference scheme, and the coupled system is solved using Newton's method. These problems can be transient or steady-state and linear or nonlinear. In Abaqus/Explicit the heat transfer equations are integrated using an explicit forward-difference time integration rule, and the mechanical solution response is obtained using an explicit central-difference integration rule. Fully coupled thermal-stress analysis in Abaqus/Explicit is always transient. Cavity radiation effects cannot be included in a fully coupled thermal-stress analysis. See "Fully coupled thermal-stress analysis," Section 6.5.3, for more details.
- **Fully coupled thermal-electrical-structural analysis:** A coupled thermal-electrical-structural procedure is used to solve simultaneously for the stress/displacement, the electrical potential, and the temperature fields. A coupled analysis is used when the thermal, electrical, and mechanical solutions affect each other strongly. An example of such a process is resistance spot welding, where two or more metal parts are joined by fusion at discrete points at the material interface. The fusion is caused by heat generated due to the current flow at the contact points, which depends on the pressure applied at these points.

  These problems can be transient or steady-state and linear or nonlinear. Cavity radiation effects cannot be included in a fully coupled thermal-electrical-structural analysis. This procedure is available

only in Abaqus/Standard. See “Fully coupled thermal-electrical-structural analysis,” Section 6.7.4, for more details.

- **Adiabatic analysis:** An adiabatic mechanical analysis can be used in cases where mechanical deformation causes heating, but the event is so rapid that this heat has no time to diffuse through the material. Adiabatic analysis can be performed in Abaqus/Standard or Abaqus/Explicit; see “Adiabatic analysis,” Section 6.5.4. An adiabatic analysis can be static or dynamic and linear or nonlinear.
- **Coupled thermal-electrical analysis:** A fully coupled thermal-electrical analysis capability is provided in Abaqus/Standard for problems where heat is generated due to the flow of electrical current through a conductor. See “Coupled thermal-electrical analysis,” Section 6.7.3.
- **Cavity radiation:** In Abaqus/Standard cavity radiation effects can be included (in addition to prescribed boundary radiation) in uncoupled heat transfer problems. See “Cavity radiation,” Section 41.1.1. The cavities can be open or closed. Symmetries and blocking within cavities can be modeled. view factors are calculated automatically, and motion of objects bounding a cavity can be prescribed during the analysis. Cavity radiation problems are nonlinear and can be transient or steady-state.

6.5.2 UNCOUPLED HEAT TRANSFER ANALYSIS

Products: Abaqus/Standard Abaqus/CFD Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Heat transfer analysis procedures: overview," Section 6.5.1
- *HEAT TRANSFER
- "Including volumetric heat generation in heat transfer analyses," Section 12.10.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring a heat transfer procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Uncoupled heat transfer problems:

- are those in which the temperature field is calculated without consideration of the stress/deformation or the electrical field in the bodies being studied;
- can include conduction, boundary convection, and boundary radiation;
- can be transient or steady-state; and
- can be linear or nonlinear.

In Abaqus/Standard uncoupled heat transfer problems:

- involve heat transfer in solid bodies;
- can include cavity radiation effects—see "Cavity radiation," Section 41.1.1;
- can include forced convection through the mesh if forced convection/diffusion heat transfer elements are used;
- can include thermal interactions such as gap radiation, conductance, and heat generation between contact surfaces—see "Thermal contact properties," Section 37.2.1;
- can include thermal material behavior defined in user subroutine **UMATHT**—see "User-defined thermal material behavior," Section 26.7.2; and
- require the use of heat transfer elements.

In Abaqus/CFD uncoupled heat transfer problems:

- involve heat transfer in solid bodies;
- must not involve fluid flow; and
- require the use of a fluid element type with a solid section—see "Fluid element library," Section 28.2.2.

Heat transfer analysis in Abaqus/Standard

Uncoupled heat transfer analysis is used to model solid body heat conduction with general, temperature-dependent conductivity, internal energy (including latent heat effects), and quite general convection and radiation boundary conditions, including cavity radiation. Forced convection of a fluid through the mesh can be modeled by using forced convection/diffusion elements.

Sources of nonlinearity in a heat transfer analysis

Heat transfer problems can be nonlinear because the material properties are temperature dependent or because the boundary conditions are nonlinear. Usually the nonlinearity associated with temperature-dependent material properties is mild because the properties do not change rapidly with temperature. However, when latent heat effects are included, the analysis may be severely nonlinear (see "Latent heat," Section 26.2.4).

Boundary conditions are very often nonlinear; for example, film coefficients can be functions of surface temperature. Again, the nonlinearities are often mild and cause little difficulty. An exception is the "boiling" film condition, in which the film coefficient can change very rapidly because the fluid adjacent to the surface boils. A rapidly changing film condition (within a step or from one step to another) can be modeled easily using temperature-dependent and field-variable-dependent film coefficients. Radiation effects always make heat transfer problems nonlinear. Nonlinearities in radiation grow as temperatures increase.

Abaqus/Standard uses an iterative scheme to solve nonlinear heat transfer problems. The scheme uses the Newton method with some modifications to improve stability of the iteration process in the presence of highly nonlinear latent heat effects.

Steady-state cases involving severe nonlinearities are sometimes more effectively solved as transient cases because of the stabilizing influence of the heat capacity terms. The required steady-state solution can be obtained as the very long transient time response; the transient will simply stabilize the solution for that long time response.

Matrix storage and solution scheme

In heat transfer analyses involving cavity radiation or forced convection/diffusion elements, the system of equations is unsymmetric. The nonsymmetric matrix storage and solution scheme is invoked automatically in these cases (see "Defining an analysis," Section 6.1.2).

Steady-state analysis

Steady-state analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. The problem then has no intrinsic physically meaningful time scale. Nevertheless, you can assign an initial time increment, a total time period, and maximum and minimum allowed time increments to the analysis step, which is often convenient for output identification and for specifying prescribed temperatures and fluxes with varying magnitudes.

Any fluxes or boundary condition changes to be applied during a steady-state heat transfer step should be given within the step, using appropriate amplitude references to specify their "time" variations

("Amplitude curves," Section 34.1.2). If fluxes and boundary conditions are specified for the step without amplitude references, they are assumed to change linearly with "time" during the step, from their magnitudes at the end of the previous step (or zero, if this is the beginning of the analysis) to their newly specified magnitudes at the end of the heat transfer step.

Input File Usage: *HEAT TRANSFER, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Steady state**

Automatic incrementation

When steady-state analysis is chosen, you suggest an initial "time" increment and define a "time" period for the step; Abaqus/Standard then increments through the step accordingly. By default, Abaqus/Standard automatically determines a suitable increment size for each increment of the step.

Fixed incrementation

You can also use a fixed incrementation scheme, in which Abaqus/Standard uses the same increment size for the duration of the step. The suggested initial "time" increment, Δt_0, defines the increment size.

Input File Usage: Set the initial increment, minimum increment size, and maximum increment size to the same value:

*HEAT TRANSFER, STEADY STATE
Δt_0, , Δt_0, Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Steady-state**: **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Transient analysis

Time integration in transient problems is done with the backward Euler method (sometimes also referred to as the modified Crank-Nicholson operator) in the pure conduction elements. This method is unconditionally stable for linear problems.

The forced convection/diffusion elements use the trapezoidal rule for time integration. They include numerical diffusion control (the "upwinding" Petrov-Galerkin method) and, optionally, numerical dispersion control. The elements with dispersion control offer improved solution accuracy in cases where the transient response of the fluid is important. Artificial dispersion control introduces a stability limit on the size of the time increment such that the local Courant number

$$C = |\mathbf{v}|\frac{\Delta t}{\Delta \ell}$$

must be less than 1, where Δt is the time increment, $|\mathbf{v}|$ is the magnitude of the velocity vector, and $\Delta \ell$ is a characteristic element length in the direction of flow; that is, heat cannot be convected across more than one element length, $\Delta \ell$, in a single increment of time. In a uniform velocity field the smallest element will dictate the stable time increment. Approximate calculation of the Courant number, C, is helpful during the mesh design stages so that excessively small stable time increments can be avoided. The

elements without dispersion control have no such stability limit; therefore, it may be more economical to use the elements without this feature in transient cases where transient effects in the fluid itself are not a critical part of the solution (for example, when the important solution is the temperature field in the solid bodies that are included in the model, and when characteristic transient times in the fluid are very much shorter than characteristic transient times in the solids).

Time incrementation in a transient heat transfer analysis can be controlled directly by you or automatically by Abaqus/Standard. Automatic time incrementation is generally preferred.

Automatic incrementation

The time increments can be selected automatically based on the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see "Time integration accuracy in transient problems," Section 7.2.4).

Input File Usage: *HEAT TRANSFER, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed incrementation

If you select direct incrementation and do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will then be used throughout the analysis.

Input File Usage: *HEAT TRANSFER
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious oscillations due to small time increments

In transient heat transfer analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta\ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta\ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. Abaqus/Standard provides no check on the user-defined initial time increment; you must ensure that the given value does not violate the above criterion.

In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions especially in terms of the heat flux for small

time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes occur.

Unless you specify a maximum allowable time increment size as part of the heat transfer step definition, there is no upper limit on the time increment size (the integration procedure is unconditionally stable, at least for linear problems). However, if forced convection/diffusion elements including numerical dispersion control (element types DCC*xx*D) are included in the model, there is a numerical stability limit on the allowable time increment. The requirement is that $\Delta t \leq \Delta\ell/|\mathbf{v}|$, where $\mathbf{v}|$ is the magnitude of the fluid velocity and $\Delta\ell$ is a characteristic element length in the direction of flow. Abaqus/Standard will adjust the time increment automatically to satisfy this stability limit.

Ending a transient analysis

A transient analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or after the given time period, whichever comes first. Steady state is defined by the temperature change rate: when the temperature at every temperature degree of freedom changes at a rate that is less than the user-specified rate (given as part of the step definition), the analysis terminates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

*HEAT TRANSFER, END=PERIOD (default)

Use the following option to end the analysis based on the temperature change rate:

*HEAT TRANSFER, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Heat transfer**: **Response: Transient**: **Incrementation**: **End step when temperature change is less than**

Internal heat generation

Volumetric heat generation within a material can be defined either in user subroutine **HETVAL** or user subroutine **UMATHT**. These user subroutines are mutually exclusive.

Defining internal heat generation in user subroutine HETVAL

If user subroutine **HETVAL** is used to define internal heat generation, heat generation must be included in the material definition with the other thermal property definitions.

Heat generation might be associated with (relatively low) energy phase changes occurring during the solution. Such heat generation usually depends on state variables (such as the fraction transformed), which themselves evolve with the solution and are stored as solution-dependent state variables (see "User subroutines: overview," Section 18.1.1). The heat generation is computed in user subroutine **HETVAL**, where any associated state variables can also be updated. The subroutine will be called at all material calculation points for which the material definition includes heat generation.

Input File Usage: *HEAT GENERATION

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Heat Generation**

Defining internal heat generation in user subroutine **UMATHT**

If user subroutine **UMATHT** is used to define internal heat generation, all other thermal properties must also be defined within the subroutine.

Input File Usage: *USER MATERIAL

Abaqus/CAE Usage: Property module: material editor: **General**: **User Material**: **User material type: Thermal**

Forced convection through the mesh

The velocity of a fluid moving through the mesh can be prescribed if forced convection/diffusion heat transfer elements are used. Conduction between the fluid and adjacent forced convection/diffusion heat transfer elements will be affected by the mass flow rate of the fluid. For example, if a pipe is filled with a fluid with an initial temperature profile that contains a temperature pulse, the initial temperature pulse will not only diffuse (because of conduction in the fluid and the pipe), but it will also be transported (or convected) down the pipe. Since the fluid velocity is prescribed, it is called forced convection.

Natural convection occurs when differences in fluid density created by thermal gradients cause motion of the fluid (bouyancy-driven flow). The forced convection/diffusion elements are not designed to handle this phenomenon; the flow must be prescribed.

You can specify the mass flow rates per unit area (or through the entire section for one-dimensional elements) at the nodes. Abaqus/Standard interpolates the mass flow rates to the material points. The numerical solution of the transient heat transfer equation including convection becomes increasingly difficult as convection dominates diffusion. The Peclet number, γ, is a dimensionless parameter that indicates the degree of convection dominance over diffusion:

$$\gamma = |\mathbf{v}|\Delta\ell\frac{\rho c}{k},$$

where $|\mathbf{v}|$ is the magnitude of the velocity vector, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta\ell$ is a characteristic element length in the direction of flow. Large values of γ indicate that convection dominates over diffusion on the spatial scale defined by the element size, $\Delta\ell$. In general, Peclet numbers greater than about 1000 should not be used.

Petrov-Galerkin finite elements are used in Abaqus/Standard to model systems with high Peclet numbers accurately; these elements use nonsymmetric, upwinded weighting functions to control numerical diffusion and dispersion and, thus, stabilize results. The upwinding term is partly a function of the element Peclet number, as described in "Convection/diffusion," Section 2.11.3 of the Abaqus Theory Guide.

If the fluid flows along a boundary along which a rapid change of temperature is prescribed, it is, in fact, subjected to a thermal transient, even for steady-state analysis. This transient can give rise to the same kind of spurious temperature oscillations that are observed in transient heat transfer analysis, as discussed earlier in this section. Since Abaqus/Standard uses first-order elements for convective heat transfer, the oscillation can be eliminated by lumping the heat capacity terms. However, the upwinded weighting functions prevent lumping in the direction of the flow. Hence, spurious oscillations may still

occur, in particular if the flow is not precisely tangential to the boundary along which the temperature change occurs.

Input File Usage: Use the following option within the heat transfer step definition to prescribe the fluid velocity:

*MASS FLOW RATE

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Modifying or removing mass flow rates

By default, the mass flow rates given are modifications of existing flow rates or are to be applied in addition to any mass flow rates defined previously. You can remove all previously defined mass flow rates and, optionally, specify new mass flow rates.

Input File Usage: Use the following option to modify an existing flow rate or to specify an additional flow rate:

*MASS FLOW RATE, OP=MOD (default)

Use the following option to release all previously applied flow rates and to specify new flow rates:

*MASS FLOW RATE, OP=NEW

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Specifying time-dependent mass flow rates

Mass flow rates can be given in combination with an amplitude definition, if required, to control the magnitude of the flow rate as a function of time (“Amplitude curves,” Section 34.1.2).

Input File Usage: Use both of the following options to define a time-dependent mass flow rate:

*AMPLITUDE, NAME=*name*
*MASS FLOW RATE, AMPLITUDE=*name*

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Defining mass flow rates in a user subroutine

Mass flow rates can be defined by user subroutine **UMASFL**. **UMASFL** will be called for each specified node. Any mass flow rate values given directly will be ignored.

Input File Usage: *MASS FLOW RATE, USER

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Reading the mass flow rate data from an alternate file

The data for the mass flow rate can be contained in an alternate file. See “Input syntax rules,” Section 1.2.1, for the syntax of the file name.

Input File Usage: *MASS FLOW RATE, INPUT=*file_name*

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Cavity radiation

Cavity radiation can be activated in a heat transfer step. This feature involves interacting heat transfer between all of the facets of the cavity surface, dependent on the facet temperatures, facet emissivities, and the geometric view factors between each facet pair. When the thermal emissivity is a function of temperature or field variables, you can specify the maximum allowable emissivity change during an increment in addition to the maximum temperature change to control the time incrementation. See "Cavity radiation," Section 41.1.1, for more information.

Input File Usage: Use the following option in the step definition to activate cavity radiation:

*RADIATION VIEW FACTOR

Use the following option to specify the maximum allowable emissivity change:

*HEAT TRANSFER, MXDEM=*max_delta_emissivity*

Abaqus/CAE Usage: You can specify the maximum allowable emissivity change for a heat transfer step.

Step module: **Create Step**: **General**: **Heat transfer**: **Incrementation**: **Max. allowable emissivity change per increment**

Heat transfer analysis in Abaqus/CFD

Uncoupled heat transfer analysis can be used in Abaqus/CFD to model heat conduction in solids provided there are no fluids in the model. This capability is distinct from fluid analysis with heat transfer, which is covered in "Fluid dynamic analysis," Section 6.6. Solids should be modeled using fluid element types with a solid section type (see "Fluid element library," Section 28.2.2). General temperature-dependent conductivity, convection, and radiation boundary conditions are supported. Nonlinearity can be introduced through temperature-dependent material properties and radiation boundary conditions. The solid heat transfer analysis is implemented using a finite volume formulation.

Steady-state analysis

Steady-state analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. The problem then has no intrinsic physically meaningful time scale. In this case the time incrementation input data are ignored, and Abaqus/CFD automatically iterates until steady-state conditions are reached. For output purposes the time increment size is set to unity, and simulation "time" can be interpreted as iteration count. Any fluxes or boundary conditions used in a steady-state heat transfer step should use values that are constant over time.

Input File Usage: *HEAT TRANSFER, CENTERING=ELEMENT,
TYPE=THERMAL FLOW, STEADY STATE

Transient analysis

Transient analysis is conducted using the trapezoidal rule and a fixed time increment size that you specify. Material response is linearized over each time increment. Iterations are performed automatically within each increment when radiation boundary conditions are present.

Input File Usage: *HEAT TRANSFER, CENTERING=ELEMENT, TYPE=THERMAL FLOW
Δt_0

Linear equation solvers

The solution methods for heat conduction equations in Abaqus/CFD rely on scalable parallel preconditioned Krylov solvers. A set of preselected default convergence criteria and iteration limits are prescribed for all linear equation solvers. The default solver settings should provide computationally efficient and robust solutions across a spectrum of heat transfer problems. However, Abaqus/CFD provides full access to diagnostic information, convergence criteria, and optional solvers.

Input File Usage: *ENERGY EQUATION SOLVER

Initial conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Forced convection through the mesh

In an Abaqus/Standard heat transfer analysis involving forced convection through the mesh, you can define nonzero initial mass flow rates at the nodes of the forced convection/diffusion heat transfer elements in the model, as described in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

For element types DCC1D2 and DCC1D2D the mass flow rate is positive from the first to the second node of the element. For two- and three-dimensional elements the direction of the mass flow rate is defined by giving the components in the x-, y-, and z-directions.

Input File Usage: *INITIAL CONDITIONS, TYPE=MASS FLOW RATE

Abaqus/CAE Usage: Mass flow rate is not supported in Abaqus/CAE.

Boundary conditions

In Abaqus/Standard, boundary conditions can be used to prescribe temperatures (degree of freedom 11) at nodes in a heat transfer analysis (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Shell elements have additional temperature degrees of freedom 12, 13, etc. through the thickness (see "Conventions," Section 1.2.2). Boundary conditions can be specified as functions of time by referring to amplitude curves (see "Amplitude curves," Section 34.1.2).

In the finite volume formulation in Abaqus/CFD, temperature is an element-based degree of freedom. You can use boundary conditions to prescribe surface temperatures, and Abaqus/CFD computes element values that satisfy the heat conduction equations. See "Boundary conditions in Abaqus/CFD," Section 34.3.2.

For purely diffusive heat transfer elements a boundary without any prescribed boundary conditions (natural boundary condition) corresponds to an insulated surface. For forced convection/diffusion elements only the flux associated with conduction is zero; energy is free to convect across an unconstrained surface. This natural boundary condition correctly models areas where fluid is crossing

a surface (as, for example, at the upstream and downstream boundaries of the mesh) and prevents spurious reflections of energy back into the mesh.

Loads

The following types of loading can be prescribed in a heat transfer analysis, as described in "Thermal loads," Section 34.4.4:

- Concentrated heat fluxes (Abaqus/Standard only).
- Body fluxes and distributed surface fluxes.
- Average-temperature radiation conditions.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Cavity radiation effects can also be included in Abaqus/Standard, as described in "Cavity radiation," Section 41.1.1.

Predefined fields

Predefined temperature fields are not allowed in heat transfer analyses. Boundary conditions should be used instead to specify temperatures, as described earlier.

Other predefined field variables can be specified in a heat transfer analysis. These values will affect field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

The thermal conductivity of the materials in a heat transfer analysis must be defined. The specific heat and density of the materials must also be defined for transient heat transfer problems. Latent heat can be defined for diffusive heat transfer elements in Abaqus/Standard if changes in internal energy due to phase changes are important. Latent heat cannot be defined directly for forced convection/diffusion elements. See "Thermal properties: overview," Section 26.2.1, for details on defining thermal properties in Abaqus.

Alternatively, user subroutine **UMATHT** can be used in Abaqus/Standard to define the thermal constitutive behavior of the material, including internal heat generation. For example, if a material modeled can go through a complex phase change, the specific heat can be defined in user subroutine **UMATHT** in sufficient detail to capture the phase change.

Thermal expansion coefficients are not meaningful in an uncoupled heat transfer analysis problem since deformation of the structure is not considered.

Elements

The heat transfer element library in Abaqus/Standard includes diffusive heat transfer elements, which allow for heat storage (specific heat and latent heat effects) and heat conduction.

Forced convection/diffusion heat transfer elements are also available in Abaqus/Standard: in addition to heat storage and heat conduction these elements allow for forced convection caused by fluid flowing through the mesh. These elements cannot be used with latent heat—see "Solid (continuum)

elements," Section 28.1.1, for additional details. Forced convection/diffusion elements with dispersion control are available for problems where the temperature transient in the fluid must be calculated accurately. See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Multiple temperatures are available through the thickness of shell heat transfer elements in Abaqus/Standard. See "Choosing a shell element," Section 29.6.2.

The first-order heat transfer elements in Abaqus/Standard (such as the 2-node link, 4-node quadrilateral, and 8-node brick) use a numerical integration rule with the integration stations located at the corners of the element for the heat capacitance terms and for the calculations of the distributed surface fluxes. First-order diffusive elements are preferred in cases involving latent heat effects since they use such a special integration technique to provide accurate solutions with large latent heats. The forced convection/diffusion elements cannot use this special integration technique and, therefore, are unsuitable for problems with latent heat effects. The second-order heat transfer elements use conventional Gaussian integration. Thus, the second-order elements are to be preferred for problems when the solution will be smooth (without latent heat effects), and usually give more accurate results for the same number of nodes in the mesh.

Thermal interactions between adjacent surfaces and thermal gap elements are also provided in Abaqus/Standard to model heat transfer across the boundary layer between a solid and a fluid or between two closely adjacent solids. See "Thermal contact properties," Section 37.2.1.

In Abaqus/CFD fluid elements should be used with solid sections to build uncoupled solid heat transfer models (see "Fluid element library," Section 28.2.2). Fluid materials are not allowed in these models. This capability is distinct from fluid analysis with heat transfer, where fluid elements are used with fluid sections (see "Fluid dynamic analysis," Section 6.6).

Output

Different types of heat transfer output are available depending on whether you are performing an Abaqus/Standard or an Abaqus/CFD analysis.

Output in Abaqus/Standard

The following heat transfer output variables are available:

Element integration point variables:

HFL	Magnitude and components of the heat flux vector.
HFL*n*	Component *n* of the heat flux vector (*n*=1, 2, 3).
HFLM	Magnitude of the heat flux vector.
TEMP	Integration point temperatures.
MFR	User-specified mass flow rates.
MFR*n*	Component *n* of the mass flow rate (*n*=1, 2, 3).

Whole element variables:

FLUXS	Current values of uniform distributed heat fluxes.
NFLUX	Fluxes at the nodes caused by heat conduction (internal fluxes).

FILM	Current values of film conditions.
RAD	Current values of radiation conditions.

Nodal variables:

NT	Nodal point temperatures.
NT*n*	Temperature degree of freedom *n* at a node (*n*=11, 12, …).
RFL	Reaction flux values due to prescribed temperature.
RFL*n*	Reaction flux value *n* at a node (*n*=11, 12, …).
CFL	Concentrated flux values.
CFL*n*	Concentrated flux value *n* at a node (*n*=11, 12, …).
RFLE	Total flux at a node, including flux convected through the node in forced convection/diffusion elements but excluding external fluxes due to user-defined concentrated fluxes, distributed fluxes, film conditions, radiation conditions, and cavity radiation. Since RFLE is a scalar nodal output variable, care should be taken when summing it over on two surfaces with shared nodes. If node sets on both surfaces include the shared nodes, the output of RFLE on the common nodes will contribute to the sums of this output quantity on both surfaces.
RFLE*n*	Total flux value *n* at a node (*n*=11, 12, …).

All of the output variables available in Abaqus/Standard are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Output in Abaqus/CFD

The following heat transfer output variables are available:

Element variables:

TEMP	Current values temperature.

Nodal variables:

TEMP	Current values temperature.

Surface variables:

AVGTEMP	Area-averaged surface temperature.
HEATFLOW	Integrated normal heat flux on a surface. Heat flow is positive when heat is added to the system. This output request does not include the convective heat flow.
HFL	Heat flux vector on a surface. This output request does not include the convective heat flow.
HFLN	Normal heat flux on a surface. This output request does not include the convective heat flow.

All of the output variables available in Abaqus/CFD are listed in "Abaqus/CFD output variable identifiers," Section 4.2.3.

Input file template

```
*HEADING
…
*PHYSICAL CONSTANTS, ABSOLUTE ZERO=θ^Z
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data lines to prescribe initial temperatures at the nodes
*AMPLITUDE, NAME=trefamp
Data lines to define amplitude curve to be used for radiation reference temperature, θ^0
*FILM PROPERTY, NAME=film
Data lines to define the convection film coefficient, h, as a function of temperature
**
*STEP
Transient analysis including forced convection through the mesh
*HEAT TRANSFER, END=SS, DELTMX=Δθ_max
Data line to define incrementation and steady state
**
*CFLUX and/or *DFLUX
Data lines to define concentrated and/or distributed fluxes
*FILM
Data lines referring to film property table  film
*RADIATE, AMPLITUDE=trefamp
Data lines to define boundary radiation
**
*EL PRINT
 TEMP, HFL
 NFLUX, FILM, RAD
*NODE PRINT
 NT11, RFL
*END STEP
```

6.5.3 FULLY COUPLED THERMAL-STRESS ANALYSIS

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Heat transfer analysis procedures: overview," Section 6.5.1
- *COUPLED TEMPERATURE-DISPLACEMENT
- *DYNAMIC TEMPERATURE-DISPLACEMENT
- "Specifying an inelastic heat fraction," Section 12.10.3 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring a fully coupled, simultaneous heat transfer and stress procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring a dynamic fully coupled thermal-stress procedure using explicit integration" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A fully coupled thermal-stress analysis:

- is performed when the mechanical and thermal solutions affect each other strongly and, therefore, must be obtained simultaneously;
- requires the existence of elements with both temperature and displacement degrees of freedom in the model;
- can be used to analyze time-dependent material response;
- cannot include cavity radiation effects but may include average-temperature radiation conditions (see "Thermal loads," Section 34.4.4); and
- takes into account temperature dependence of material properties only for the properties that are assigned to elements with temperature degrees of freedom.

In Abaqus/Standard a fully coupled thermal-stress analysis:

- neglects inertia effects; and
- can be transient or steady-state.

In Abaqus/Explicit a fully coupled thermal-stress analysis:

- includes inertia effects; and
- models transient thermal response.

Fully coupled thermal-stress analysis

Fully coupled thermal-stress analysis is needed when the stress analysis is dependent on the temperature distribution and the temperature distribution depends on the stress solution. For example, metalworking problems may include significant heating due to inelastic deformation of the material which, in turn, changes the material properties. In addition, contact conditions exist in some problems where the heat conducted between surfaces may depend strongly on the separation of the surfaces or the pressure transmitted across the surfaces (see "Thermal contact properties," Section 37.2.1). For such cases the thermal and mechanical solutions must be obtained simultaneously rather than sequentially. Coupled temperature-displacement elements are provided for this purpose in both Abaqus/Standard and Abaqus/Explicit; however, each program uses different algorithms to solve coupled thermal-stress problems.

Fully coupled thermal-stress analysis in Abaqus/Standard

In Abaqus/Standard the temperatures are integrated using a backward-difference scheme, and the nonlinear coupled system is solved using Newton's method. Abaqus/Standard offers an exact as well as an approximate implementation of Newton's method for fully coupled temperature-displacement analysis.

Exact implementation

An exact implementation of Newton's method involves a nonsymmetric Jacobian matrix as is illustrated in the following matrix representation of the coupled equations:

$$\begin{bmatrix} \boldsymbol{K_{uu}} & \boldsymbol{K_{u\theta}} \\ \boldsymbol{K_{\theta u}} & \boldsymbol{K_{\theta\theta}} \end{bmatrix} \begin{Bmatrix} \Delta \boldsymbol{u} \\ \Delta \boldsymbol{\theta} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{R_u} \\ \boldsymbol{R_\theta} \end{Bmatrix},$$

where $\Delta \boldsymbol{u}$ and $\Delta \boldsymbol{\theta}$ are the respective corrections to the incremental displacement and temperature, $\boldsymbol{K_{ij}}$ are submatrices of the fully coupled Jacobian matrix, and $\boldsymbol{R_u}$and $\boldsymbol{R_\theta}$ are the mechanical and thermal residual vectors, respectively.

Solving this system of equations requires the use of the unsymmetric matrix storage and solution scheme. Furthermore, the mechanical and thermal equations must be solved simultaneously. The method provides quadratic convergence when the solution estimate is within the radius of convergence of the algorithm. The exact implementation is used by default.

Approximate implementation

Some problems require a fully coupled analysis in the sense that the mechanical and thermal solutions evolve simultaneously, but with a weak coupling between the two solutions. In other words, the components in the off-diagonal submatrices $\boldsymbol{K_{u\theta}}$, $\boldsymbol{K_{\theta u}}$ are small compared to the components in the diagonal submatrices $\boldsymbol{K_{uu}}$, $\boldsymbol{K_{\theta\theta}}$. An example of such a situation is the disc brake problem ("Thermal-stress analysis of a disc brake," Section 5.1.1 of the Abaqus Example Problems Guide). For these problems a less costly solution may be obtained by setting the off-diagonal submatrices to zero so that we obtain an approximate set of equations:

$$\begin{bmatrix} \boldsymbol{K_{uu}} & 0 \\ 0 & \boldsymbol{K_{\theta\theta}} \end{bmatrix} \begin{Bmatrix} \Delta \boldsymbol{u} \\ \Delta \boldsymbol{\theta} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{R_u} \\ \boldsymbol{R_\theta} \end{Bmatrix}.$$

As a result of this approximation the thermal and mechanical equations can be solved separately, with fewer equations to consider in each subproblem. The savings due to this approximation, measured as solver time per iteration, will be of the order of a factor of two, with similar significant savings in solver storage of the factored stiffness matrix. Further, in many situations the subproblems may be fully symmetric or approximated as symmetric, so that the less costly symmetric storage and solution scheme can be used. The solver time savings for a symmetric solution is an additional factor of two. Unless you explicitly choose the unsymmetric matrix storage and solution scheme, selection of the scheme will depend on other details of the problem (see "Defining an analysis," Section 6.1.2).

This modified form of Newton's method does not affect solution accuracy since the fully coupled effect is considered through the residual vector $\boldsymbol{R_j}$ at each increment in time. However, the rate of convergence is no longer quadratic and depends strongly on the magnitude of the coupling effect, so more iterations are generally needed to achieve equilibrium than with the exact implementation of Newton's method. When the coupling is significant, the convergence rate becomes very slow and may prohibit obtaining a solution. In such cases the exact implementation of Newton's method is required. In cases where it is possible to use this approximation, the convergence in an increment will depend strongly on the quality of the first guess to the incremental solution, which you can control by selecting the extrapolation method used for the step (see "Defining an analysis," Section 6.1.2).

Input File Usage: Use the following option to specify a separated solution scheme:

*SOLUTION TECHNIQUE, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Other**: **Solution technique: Separated**

Steady-state analysis

A steady-state coupled temperature-displacement analysis can be performed in Abaqus/Standard. In steady-state cases you should assign an arbitrary "time" scale to the step: you specify a "time" period and "time" incrementation parameters. This time scale is convenient for changing loads and boundary conditions through the step and for obtaining solutions to highly nonlinear (but steady-state) cases; however, for the latter purpose, transient analysis often provides a natural way of coping with the nonlinearity.

Frictional slip heat generation is normally neglected in for the steady-state case. However, it can still be accounted for if motions are used to specify translational or rotational nodal velocities in disk brake-type problems or if user subroutine **FRIC** provides the incremental frictional dissipation through the variable **SFD**. If frictional heat generation is present, the heat flux into the two contact surfaces depends on the slip rate of the surfaces. The "time" scale in this case cannot be described as arbitrary, and a transient analysis should be performed.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Steady state**

Transient analysis

Alternatively, you can perform a transient coupled temperature-displacement analysis. You can control the time incrementation in a transient analysis directly, or Abaqus/Standard can control it automatically. Automatic time incrementation is generally preferred.

Automatic incrementation controlled by a maximum allowable temperature change

The time increments can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see "Time integration accuracy in transient problems," Section 7.2.4).

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**, **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed incrementation

If you do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will be used throughout the analysis.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious oscillations due to small time increments

In transient analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta\ell^2,$$

where Δt is the time increment, ρ is the density, $\boldsymbol{c}$ is the specific heat, $\boldsymbol{k}$ is the thermal conductivity, and $\Delta\ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Automatic incrementation controlled by the creep response

The accuracy of the integration of time-dependent (creep) material behavior is governed by the user-specified accuracy tolerance parameter, $tolerance \geq (\dot{\bar{\varepsilon}}^{cr}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}_{t})\Delta t$. This parameter is used to prescribe the maximum strain rate change allowed at any point during an increment, as described in "Rate-dependent plasticity: creep and swelling," Section 23.2.4. The accuracy tolerance parameter can be specified together with the maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$ (described above); however, specifying the accuracy tolerance parameter activates automatic incrementation even if $\Delta\theta_{max}$ is not specified.

Input File Usage: ∗COUPLED TEMPERATURE-DISPLACEMENT, DELTMX=$\Delta\theta_{max}$, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**, **Max. allowable temperature change per increment:** $\Delta\theta_{max}$, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Selecting explicit creep integration

Nonlinear creep problems ("Rate-dependent plasticity: creep and swelling," Section 23.2.4) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required as long as no other nonlinearities are present. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For most coupled thermal-stress analyses, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases the implicit integration scheme may be invoked automatically by Abaqus/Standard.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine **CREEP**; you can restrict Abaqus/Standard to using this method for creep problems (with or without geometric nonlinearity included). See "Rate-dependent plasticity: creep and swelling," Section 23.2.4, for further details.

Input File Usage: ∗COUPLED TEMPERATURE-DISPLACEMENT, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*, **Creep/swelling/viscoelastic integration: Explicit**

Excluding creep and viscoelastic response

You can specify that no creep or viscoelastic response will occur during a step even if creep or viscoelastic material properties have been defined.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, DELTMX=$\Delta\theta_{max}$, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**: **Basic**: **Response: Transient**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers a method of stabilizing this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Units

In coupled problems where two different fields are active, take care when choosing the units of the problem. If the choice of units is such that the terms generated by the equations for each field are different by many orders of magnitude, the precision on some computers may be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid ill-conditioned matrices. For example, consider using units of Mpascal instead of pascal for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations and the heat flux continuity equations.

Fully coupled thermal-stress analysis in Abaqus/Explicit

In Abaqus/Explicit the heat transfer equations are integrated using the explicit forward-difference time integration rule

$$\theta^N_{(i+1)} = \theta^N_{(i)} + \Delta t_{(i+1)} \dot{\theta}^N_{(i)},$$

where θ^N is the temperature at node N and the subscript i refers to the increment number in an explicit dynamic step. The forward-difference integration is explicit in the sense that no equations need to be solved when a lumped capacitance matrix is used. The current temperatures are obtained using known values of $\dot{\theta}^N_{(i)}$ from the previous increment. The values of $\dot{\theta}^N_{(i)}$ are computed at the beginning of the increment by

$$\dot{\theta}^N_{(i)} = (C^{NJ})^{-1}(P^J_{(i)} - F^J_{(i)}),$$

where C^{NJ} is the lumped capacitance matrix, P^J is the applied nodal source vector, and F^J is the internal flux vector.

The mechanical solution response is obtained using the explicit central-difference integration rule with a lumped mass matrix as described in "Explicit dynamic analysis," Section 6.3.3. Since both the forward-difference and central-difference integrations are explicit, the heat transfer and mechanical solutions are obtained simultaneously by an explicit coupling. Therefore, no iterations or tangent stiffness matrices are required.

Explicit integration can be less expensive computationally and simplifies the treatment of contact. For a comparison of explicit and implicit direct-integration procedures, see "Dynamic analysis procedures: overview," Section 6.3.1.

Stability

The explicit procedure integrates through time by using many small time increments. The central-difference and forward-difference operators are conditionally stable. The stability limit for both operators (with no damping in the mechanical solution response) is obtained by choosing

$$\Delta t \leq min(\frac{2}{\omega_{max}}, \frac{2}{\lambda_{max}}),$$

where ω_{max} is the highest frequency in the system of equations of the mechanical solution response and λ_{max} is the largest eigenvalue in the system of equations of the thermal solution response.

Estimating the time increment size

An approximation to the stability limit for the forward-difference operator in the thermal solution response is given by

$$\Delta t \approx \frac{L^2_{min}}{2\alpha},$$

where L_{min} is the smallest element dimension in the mesh and $\alpha = \frac{k}{\rho c}$ is the thermal diffusivity of the material. The parameters $\boldsymbol{k}$, ρ, and $\boldsymbol{c}$ represent the material's thermal conductivity, density, and specific heat, respectively.

In most applications of explicit analysis the mechanical response will govern the stability limit. The thermal response may govern the stability limit when material parameter values are non-physical or a very large amount of mass scaling is used. The calculation of the time increment size for the mechanical solution response is discussed in "Explicit dynamic analysis," Section 6.3.3.

Stable time increment report

Abaqus/Explicit writes a report to the status (`.sta`) file during the data check phase of the analysis that contains an estimate of the minimum stable time increment and a listing of the elements with the smallest stable time increments and their values. The initial minimum stable time increment accounts

for the stability requirements of both the thermal and mechanical solution responses. The initial stable time increments listed do not include damping (bulk viscosity), mass scaling, or penalty contact effects in the mechanical solution response.

This listing is provided because often a few elements have much smaller stability limits than the rest of the elements in the mesh. The stable time increment can be increased by modifying the mesh to increase the size of the controlling element or by using appropriate mass scaling.

Time incrementation

The time increment used in an analysis must be smaller than the stability limits of the central- and forward-difference operators. Failure to use such a time increment will result in an unstable solution. When the solution becomes unstable, the time history response of solution variables, such as displacements, will usually oscillate with increasing amplitudes. The total energy balance will also change significantly.

Abaqus/Explicit has two strategies for time incrementation control: fully automatic time incrementation (where the code accounts for changes in the stability limit) and fixed time incrementation.

Scaling the time increment

To reduce the chance of a solution going unstable, the stable time increment computed by Abaqus/Explicit can be adjusted by a constant scaling factor. This factor can be used to scale the default global time estimate, the element-by-element estimate, or the fixed time increment based on the initial element-by-element estimate; it cannot be used to scale a fixed time increment that you specified directly.

Input File Usage: Use any of the following options:

*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT, SCALE FACTOR=*f*
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT, ELEMENT BY ELEMENT, SCALE FACTOR=*f*
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT, FIXED TIME INCREMENTATION, SCALE FACTOR=*f*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Time scaling factor:** *f*

Automatic time incrementation

The default time incrementation scheme in Abaqus/Explicit is fully automatic and requires no user intervention. Two types of estimates are used to determine the stability limit: element-by-element for both the thermal and mechanical solution responses and global for the mechanical solution response. An analysis always starts by using the element-by-element estimation method and may switch to the global estimation method under certain circumstances, as explained in "Explicit dynamic analysis," Section 6.3.3.

In an analysis Abaqus/Explicit initially uses a stability limit based on the thermal and mechanical solution responses in the whole model. This element-by-element estimate is determined using the smallest time increment size due to the thermal and mechanical solution responses in each element.

The element-by-element estimate is conservative; it will give a smaller stable time increment than the true stability limit, which is based upon the maximum frequency of the entire model. In general, constraints such as boundary conditions and kinematic contact have the effect of compressing the eigenvalue spectrum, and the element-by-element estimates do not take this into account (see "Explicit dynamic analysis," Section 6.3.3)

The stable time increment size due to the mechanical solution response will be determined by the global estimator as the step proceeds unless the element-by-element estimator is chosen, fixed time incrementation is specified, or one of the conditions explained in "Explicit dynamic analysis," Section 6.3.3, prevents the use of global estimation. The stable time increment size due to the thermal solution response will always be determined by using an element-by-element estimation method. The switch to the global estimation method in mechanical solution response occurs once the algorithm determines that the accuracy of the global estimation method is acceptable. For details, see "Explicit dynamic analysis," Section 6.3.3

For three-dimensional continuum elements and elements with plane stress formulations (shell, membrane, and two-dimensional plane stress elements) an "improved" estimate of the element characteristic length is used by default. This "improved" method usually results in a larger element stable time increment than a more traditional method. For analyses using variable mass scaling, the total mass added to achieve a given stable time increment will be less with the improved estimate.

Input File Usage: Use the following option to specify the element-by-element estimation method:

```
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT,
ELEMENT BY ELEMENT
```

Use the following option to activate the "improved" element time estimation method:

```
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT,
IMPROVED DT METHOD=YES
```

Use the following option to deactivate the "improved" element time estimation method:

```
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT,
IMPROVED DT METHOD=NO
```

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Automatic**, **Stable increment estimator: Element-by-element**

The ability to deactivate the "improved" element time estimation method is not supported in Abaqus/CAE.

Fixed time incrementation

A fixed time incrementation scheme is also available in Abaqus/Explicit. The fixed time increment size is determined either by the initial element-by-element stability estimate for the step or by a user-specified time increment.

Fixed time incrementation may be useful when a more accurate representation of the higher mode response of a problem is required. In this case a time increment size smaller than the element-by-element estimates may be used. The element-by-element estimate can be obtained simply by running a data check analysis (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

When fixed time incrementation is used, Abaqus/Explicit will not check that the computed response is stable during the step. You should ensure that a valid response has been obtained by carefully checking the energy history and other response variables.

If you choose to use time increments the size of the initial element-by-element stability limit throughout a step, the dilatational wave speed and the thermal diffusivity in each element at the beginning of the step are used to compute the fixed time increment size. To reduce the chance of a solution going unstable, the initial stable time increment that Abaqus/Explicit computes can be adjusted by a constant scaling factor, as described above in "Scaling the time increment." Alternatively, you can specify a time increment size directly.

Input File Usage: Use the following option to request time increments the size of the element-by-element stability limit:

```
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT,
FIXED TIME INCREMENTATION
```

Use the following option to specify the time increment size directly:

```
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT,
DIRECT USER CONTROL
Δt
```

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Dynamic, Temp-disp, Explicit**: **Incrementation**: **Type: Fixed**, **Use element-by-element time increment estimator** or **User-defined time increment**: Δt

Reducing the computational cost by using selective subcycling

The selective subcycling method can be used in a coupled thermal-stress analysis exactly as in a pure mechanical analysis, as described in "Explicit dynamic analysis," Section 6.3.3 and "Selective subcycling," Section 11.7.1.

Monitoring output variables for extreme values

The extreme values defined as the element and nodal variables in a coupled thermal-stress analysis can be monitored exactly as described in "Explicit dynamic analysis," Section 6.3.3, for a pure mechanical analysis.

Initial conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures. Initial stresses, field variables, etc. can also be defined; "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the initial conditions that are available for a fully coupled thermal-stress analysis.

Boundary conditions

Boundary conditions can be used to prescribe both temperatures (degree of freedom 11) and displacements/rotations (degrees of freedom 1–6) at nodes in fully coupled thermal-stress analysis (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Shell elements in Abaqus/Standard have additional temperature degrees of freedom 12, 13, etc. through the thickness (see "Conventions," Section 1.2.2).

Boundary conditions can be specified as functions of time by referring to amplitude curves ("Amplitude curves," Section 34.1.2).

Boundary conditions applied during a dynamic coupled temperature-displacement response step should use appropriate amplitude references ("Amplitude curves," Section 34.1.2). If boundary conditions are specified for the step without amplitude references, they are applied instantaneously at the beginning of the step. Since Abaqus/Explicit does not admit jumps in displacement, the value of a nonzero displacement boundary condition that is specified without an amplitude reference will be ignored, and a zero velocity boundary condition will be enforced.

Loads

The following types of thermal loads can be prescribed in a fully coupled thermal-stress analysis, as described in "Thermal loads," Section 34.4.4:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Node-based film and radiation conditions.
- Average-temperature radiation conditions.
- Element and surface-based film and radiation conditions.

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

Predefined temperature fields are not allowed in a fully coupled thermal-stress analysis. Boundary conditions should be used instead to prescribe temperature degree of freedom 11 (and 12, 13, etc. in Abaqus/Standard shell elements), as described earlier.

Other predefined field variables can be specified in a fully coupled thermal-stress analysis. These values will affect only field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

The materials in a fully coupled thermal-stress analysis must have both thermal properties, such as conductivity, and mechanical properties, such as elasticity, defined. See Part V, "Materials," for details on the material models available in Abaqus.

In Abaqus/Standard internal heat generation can be specified; see "Uncoupled heat transfer analysis," Section 6.5.2.

Thermal strain will arise if thermal expansion ("Thermal expansion," Section 26.1.2) is included in the material property definition.

In Abaqus/Standard a fully coupled temperature-displacement analysis can be used to analyze static creep and swelling problems, which generally occur over fairly long time periods ("Rate-dependent plasticity: creep and swelling," Section 23.2.4); viscoelastic materials ("Time domain viscoelasticity," Section 22.7.1); or viscoplastic materials ("Rate-dependent yield," Section 23.2.3).

Inelastic energy dissipation as a heat source

You can specify an inelastic heat fraction in a fully coupled thermal-stress analysis to provide for inelastic energy dissipation as a heat source. The heat flux per unit volume, r^{pl}, that is added into the thermal energy balance is computed using the equation

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

or, in the case when the nonlinear isotropic/kinematic hardening model is used, from the following equation:

$$r^{pl} = \eta(\boldsymbol{\sigma} - \boldsymbol{\alpha}) : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where η is a user-defined factor (assumed constant), $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\alpha}$ is the backstress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining.

Inelastic heat fractions are typically used in the simulation of high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation significantly influences temperature-dependent material properties. The generated heat is treated as a volumetric heat flux source term in the heat balance equation.

An inelastic heat fraction can be specified for materials with plastic behavior that use the Mises or Hill yield surface ("Inelastic behavior," Section 23.1.1). It cannot be used with the combined isotropic/kinematic hardening model. The inelastic heat fraction can be specified for user-defined material behavior in Abaqus/Explicit and will be multiplied by the inelastic energy dissipation coded in the user subroutine to obtain the heat flux. In Abaqus/Standard the inelastic heat fraction cannot be used with user-defined material behavior; in this case the heat flux that must be added to the thermal energy balance is computed directly in the user subroutine.

An inelastic heat fraction can also be specified for material definitions that include time-domain linear viscoelasticity ("Time domain viscoelasticity," Section 22.7.1) and time-domain nonlinear viscoelasticity defined within the parallel rheological framework ("Parallel rheological framework,"

Section 22.8.2), except in Abaqus/Explicit for large-strain linear viscoelasticity. For large-strain linear viscoelasticity in Abaqus/Standard ("Time domain viscoelasticity," Section 22.7.1), the energy dissipation is computed only approximately. Hence, the fraction of the dissipated energy converted into heat can be computed only approximately.

The default value of the inelastic heat fraction is 0.9. If you do not include the inelastic heat fraction behavior in the material definition, the heat generated by inelastic deformation is not included in the analysis.

Input File Usage: *INELASTIC HEAT FRACTION
η

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Inelastic Heat Fraction**: **Fraction:** η

Elements

Coupled temperature-displacement elements that have both displacements and temperatures as nodal variables are available in both Abaqus/Standard and Abaqus/Explicit (see "Choosing the appropriate element for an analysis type," Section 27.1.3). In Abaqus/Standard simultaneous temperature/displacement solution requires the use of such elements; pure displacement elements can be used in part of the model in the fully coupled thermal-stress procedure, but pure heat transfer elements cannot be used. In Abaqus/Explicit any of the available elements can be used in the fully coupled thermal-stress procedure; however, the thermal solution will be obtained only at nodes where the temperature degree of freedom has been activated (i.e., at nodes attached to coupled temperature-displacement elements).

The first-order coupled temperature-displacement elements in Abaqus use a constant temperature over the element to calculate thermal expansion. The second-order coupled temperature-displacement elements in Abaqus/Standard use a lower-order interpolation for temperature than for displacement (parabolic variation of displacements and linear variation of temperature) to obtain a compatible variation of thermal and mechanical strain.

Output

See "Abaqus/Standard output variable identifiers," Section 4.2.1, and "Abaqus/Explicit output variable identifiers," Section 4.2.2, for a complete list of output variables. The types of output available are described in "Output," Section 4.1.1.

Input file template

```
*HEADING
…
** Specify the coupled temperature-displacement element type
*ELEMENT, TYPE=CPS4T
…
**
```

***STEP**
***COUPLED TEMPERATURE-DISPLACEMENT or**
***DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT**
Data line to define incrementation
***BOUNDARY**
Data lines to define nonzero boundary conditions on displacement or temperature degrees of freedom
***CFLUX and/or *CFILM and/or**
***CRADIATE and/or *DFLUX and/or**
***DSFLUX and/or *FILM and/or**
***SFILM and/or *RADIATE and/or**
***SRADIATE**
Data lines to define thermal loads
***CLOAD and/or *DLOAD and/or *DSLOAD**
Data lines to define mechanical loads
***FIELD**
Data lines to define field variable values
***END STEP**

6.5.4 ADIABATIC ANALYSIS

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Heat transfer analysis procedures: overview," Section 6.5.1
- *DYNAMIC
- *STATIC
- *DENSITY
- *INELASTIC HEAT FRACTION
- *SPECIFIC HEAT
- "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining thermal material models," Section 12.10 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

An adiabatic stress analysis:

- is used in cases where mechanical deformation causes heating but the event is so rapid that this heat has no time to diffuse through the material—for example, a very high-speed forming process;
- can be conducted as part of a dynamic analysis ("Implicit dynamic analysis using direct integration," Section 6.3.2, or "Explicit dynamic analysis," Section 6.3.3) or as part of a static analysis ("Static stress analysis," Section 6.2.2);
- in Abaqus/Standard is available only for the isotropic hardening metal plasticity models with a Mises yield surface ("Classical metal plasticity," Section 23.2.1);
- in Abaqus/Explicit is relevant only for the metal plasticity models (including both Mises and Hill yield surfaces);
- can be conducted if parts of the model are elastic only—no change in temperature occurs in the elastic regions; and
- requires that a material's density, specific heat, and inelastic heat fraction (fraction of inelastic dissipation rate that appears as heat flux) be specified.

Adiabatic analysis

Adiabatic thermal-stress analysis is typically used to simulate high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation is an important effect because of temperature-dependent material properties. The temperature increase

is calculated directly at the material integration points according to the adiabatic thermal energy increases caused by inelastic deformation; temperature is not a degree of freedom in the problem. No allowance is made for conduction of heat in an adiabatic analysis. For problems where both inelastic heating and conduction of the heat are important, a fully coupled temperature-displacement analysis must be performed ("Fully coupled thermal-stress analysis," Section 6.5.3).

In an adiabatic analysis plastic straining gives rise to a heat flux per unit volume of

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where r^{pl} is the heat flux that is added into the thermal energy balance, η is the user-specified inelastic heat fraction (assumed constant; discussed below), $\boldsymbol{\sigma}$ is the stress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining. The heat equation solved at each integration point is

$$\rho c(\theta)\dot{\theta} = r^{pl},$$

where ρ is the material density and $c(\theta)$ is the specific heat (see "Density," Section 21.2.1, and "Specific heat," Section 26.2.3).

Input File Usage: Use any of the following procedures to perform an adiabatic analysis:

*DYNAMIC, ADIABATIC
*DYNAMIC, EXPLICIT, ADIABATIC
*STATIC, ADIABATIC

Abaqus/CAE Usage: Use any of the following procedures to perform an adiabatic analysis:

Step module:
Create Step: **Dynamic, Implicit**: **Basic**: **Include adiabatic heating effects**
Create Step: **Dynamic, Explicit**: **Basic**: **Include adiabatic heating effects**
Create Step: **Static, General**: **Basic**: **Include adiabatic heating effects**

Subsequent thermal diffusion analysis in Abaqus/Standard

In Abaqus/Standard thermal diffusion analysis can be performed after the adiabatic calculation (for example, to study the cool-down of a component after sudden deformation). In this case the temperatures at the end of the adiabatic analysis must be written to the Abaqus/Standard results file as element variables averaged at the nodes. Since temperature values in an adiabatic analysis can be written to the results file as element quantities only by using the TEMP output variable identifier, they cannot be read directly into a subsequent thermal diffusion analysis as initial conditions. However, if you postprocess the results file to produce a second results file in which the temperature data are provided as nodal quantities, a subsequent heat transfer analysis can be performed with these temperatures as initial conditions. See "Predefined fields," Section 34.6.1, and "Accessing the results file information," Section 5.1.3, for details. Alternatively, you could postprocess the results file to produce a data list containing data pairs consisting of nodes and temperatures.

The temperatures, NT, obtained from the heat transfer analysis can then be used to drive a continuation of the previous stress analysis. This stress analysis should be restarted from the end of the adiabatic analysis and will provide the response to the change of the temperature field obtained during

the heat transfer analysis. In this case Abaqus/Standard will automatically read the temperatures from the results file that was obtained from the heat transfer analysis and apply them in the restarted analysis.

Example

The following input options could be used to perform a heat transfer analysis using the temperatures from an adiabatic analysis and then continue the stress analysis:

```
**Static adiabatic analysis
 …
*STEP
*STATIC, ADIABATIC
 …
**Write the temperatures to the results file as element
**variables averaged at the nodes
*EL FILE, POSITION=AVERAGED AT NODES
 TEMP
*END STEP
**Heat transfer analysis using the temperatures from the
**static analysis as initial conditions
 …
*INITIAL CONDITIONS, TYPE=TEMPERATURE, FILE=new results file,
 STEP=step, INC=increment
*STEP
*HEAT TRANSFER
 …
*NODE FILE
 NT
*END STEP
**Restart from the adiabatic analysis using temperatures
**obtained from the heat transfer analysis
*RESTART, WRITE, READ, STEP=k, INC=i, END STEP
 …
*STEP
*STATIC
 …
*TEMPERATURE, FILE=heat_transfer_results_file
 …
*END STEP
```

Fully coupled temperature-displacement analysis

If the continuation of the analysis into thermal diffusion requires a fully coupled temperature-displacement analysis (see “Fully coupled thermal-stress analysis,” Section 6.5.3), the simplest (but

more expensive) approach is to use coupled temperature-displacement elements throughout the adiabatic analysis. At the end of the static or the dynamic adiabatic calculations, the temperatures must be written to the results file as element variables averaged at the nodes. In addition, you must constrain all temperature degrees of freedom since they are not used in the adiabatic analysis. The adiabatic analysis can then be restarted to apply the correct temperature distribution obtained from the adiabatic analysis to the temperature degree of freedom of each node in the model. To create the input for the boundary conditions, you must postprocess the results file obtained from the adiabatic analysis and extract the value of TEMP at each node in the model (see "Accessing the results file information," Section 5.1.3). The temperature boundary conditions can be released as needed in subsequent coupled temperature-displacement analysis steps.

Example

The following input options could be used to perform a coupled temperature-displacement analysis using the temperatures from an adiabatic analysis:

```
**Static adiabatic analysis, coupled temperature-displacement
**plane stress elements
 …
*ELEMENT, TYPE=CPS4T, ELSET=EALL
 …
*BOUNDARY
nodes, 11, 11, 0.0
*STEP
*STATIC, ADIABATIC
 …
**Write the temperatures to the results file as element
**variables averaged at the nodes
*EL FILE, POSITION=AVERAGED AT NODES
 TEMP
*END STEP
**Restart from the adiabatic analysis
*RESTART, WRITE, READ, STEP=k, INC=i, END STEP
 …
*STEP
*STATIC
**Dummy step to associate the temperature variable TEMP with
**the temperature degree of freedom at each node
 1.0, 1.0
 …
*BOUNDARY, OP=NEW
 node, 11, 11, temperature
 …
```

```
*END STEP
**Coupled temperature displacement run for cool down of
**structure: continuation of the restart analysis
 …
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT
 0.1, 1.0
 …
*BOUNDARY, OP=NEW
**no temperature boundary condition specified
*END STEP
```

Initial conditions

Initial temperatures can be prescribed at nodes as initial conditions. Initial values of stresses, field variables, solution-dependent state variables, etc. can also be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Boundary conditions

Boundary conditions can be applied to displacement degrees of freedom in an adiabatic analysis in the same way that they are applied in nonadiabatic dynamic, explicit dynamic, or static analysis steps (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Temperature is not a degree of freedom in an adiabatic analysis.

Loads

The loading options available for an adiabatic analysis are the same as those available for nonadiabatic dynamic, explicit dynamic, or static analysis steps (see "Applying loads: overview," Section 34.4.1).

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

Predefined temperature fields cannot be used during an adiabatic analysis step.

The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

In Abaqus/Standard only Mises plasticity with isotropic elasticity and isotropic hardening ("Inelastic behavior," Section 23.1.1) is allowed in adiabatic stress analysis. Kinematic or combined hardening is not available, but rate effects can be included. However, portions of the model can include only elastic material; no change in temperature occurs in the elastic regions, since there is no source of heat generation. In Abaqus/Explicit both Mises and Hill plasticity are allowed in adiabatic stress analysis.

You must specify the density, the inelastic heat fraction, and the specific heat as part of the material definition for the material in which heat will be generated by plastic dissipation. You can also specify latent heat if necessary ("Latent heat," Section 26.2.4).

The inelastic heat fraction is the amount of inelastic dissipation used to calculate the increase in temperature. The default value of the inelastic heat fraction is 0.9. If the inelastic heat fraction is not included in the material definition, the heat generated by inelastic deformation is not included in the analysis.

In Abaqus/Standard adiabatic analyses can also be carried out with user subroutine **UMAT**. In this case the temperature must be defined as a solution-dependent state variable, and all coupling terms must be included in the user subroutine. If conductivity ("Conductivity," Section 26.2.2) is defined for the material, it will be ignored during adiabatic analysis steps.

Input File Usage: All of the following options must be included in the material definition:

*DENSITY
*INELASTIC HEAT FRACTION
*SPECIFIC HEAT

The following option can be included if latent heat effects are important:

*LATENT HEAT

Abaqus/CAE Usage: All of the following must be included in the material definition:

Property module:
Material editor: **General→Density**
Material editor: **Thermal→Inelastic Heat Fraction**
Material editor: **Thermal→Specific Heat**

The following can be included if latent heat effects are important:

Property module: material editor: **Thermal→Latent Heat**

Temperature-dependent material properties

Material properties can be temperature dependent. Since the only source of temperature change in adiabatic analysis is inelastic deformation, the temperature can only rise. This temperature rise may cause thermal expansion (usually a small effect) and localization of the deformation if the flow stress is reduced by the temperature rise. Since the adiabatic assumption applies only in rapid events and inelastic deformation usually causes significant temperature rises only if the deformation is substantial, the strain rates are often large in adiabatic analysis. The softening of the material caused by the temperature rise

may, thus, be offset somewhat by strengthening associated with rate dependence if the material is rate sensitive.

Elements

Any of the stress/displacement or coupled temperature-displacement elements in Abaqus can be used in an adiabatic analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). Mass or spring elements will not contribute to the heating of the material since they cannot generate plastic strains.

If coupled temperature-displacement elements are used in an adiabatic analysis, the temperature degrees of freedom will be ignored.

Output

Since temperatures are updated at the material calculation points, output of temperature is available with output variable TEMP, not with output variable NT.

The element output available for an adiabatic analysis includes stress; strain; energies; the values of state, field, and user-defined variables; and composite failure measures. The nodal output available includes displacements, reaction forces, and coordinates. All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1, and "Abaqus/Explicit output variable identifiers," Section 4.2.2.

Input file template

```
*HEADING
...
*MATERIAL, NAME=name
*ELASTIC, TYPE=ISOTROPIC
Data lines to define isotropic linear elasticity
*PLASTIC
Data lines to define metal plasticity
*DENSITY
Data lines to define density
*INELASTIC HEAT FRACTION
Data line to define inelastic heat fraction
*SPECIFIC HEAT
Data lines to define specific heat
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=type
Data lines to specify initial conditions
*AMPLITUDE, NAME=name
Data lines to define amplitude variations
**
```

***STEP, NLGEOM**
The NLGEOM parameter is used in Abaqus/Standard to include geometric nonlinearity
***DYNAMIC, ADIABATIC** *or* ***DYNAMIC, EXPLICIT, ADIABATIC** *or*
***STATIC, ADIABATIC**
Data line to control time incrementation or to specify the time period of the step
BOUNDARY, AMPLITUDE=name*
Data lines to describe nonzero or zero-valued boundary conditions
***CLOAD and/or *DLOAD and/or *DSLOAD**
Data lines to specify loads
***FIELD**
Data lines to specify field variable values
***END STEP**

6.6 Fluid dynamic analysis

- "Fluid dynamic analysis procedures: overview," Section 6.6.1
- "Incompressible fluid dynamic analysis," Section 6.6.2

6.6.1 FLUID DYNAMIC ANALYSIS PROCEDURES: OVERVIEW

Overview

Abaqus/CFD provides advanced computational fluid dynamics capabilities with extensive support for preprocessing and postprocessing provided in Abaqus/CAE. These scalable parallel CFD simulation capabilities address a broad range of nonlinear coupled fluid-thermal and fluid-structural problems.

Abaqus/CFD can solve the following types of incompressible flow problems:

- **Laminar and turbulent:** Internal or external flows that are steady-state or transient, span a broad Reynolds number range, and involve complex geometry may be simulated with Abaqus/CFD. This includes flow problems induced by spatially varying distributed body forces.
- **Thermal convective:** Problems that involve heat transfer and require an energy equation and that may involve buoyancy-driven flows (i.e., natural convection) can also be solved with Abaqus/CFD. This type of problem includes turbulent heat transfer for a broad range of Prandtl numbers.
- **Deforming-mesh ALE:** Abaqus/CFD includes the ability to perform deforming-mesh analyses using an arbitrary Lagrangian-Eulerian (ALE) description of the equations of motion, heat transfer, and turbulent transport. Deforming-mesh problems may include prescribed boundary motion that induces fluid flow or FSI problems where the boundary motion is relatively independent of the fluid flow.

For more details, see "Incompressible fluid dynamic analysis," Section 6.6.2.

Activation of fields in Abaqus/CFD

In Abaqus/CFD the active fields (degrees of freedom) are determined by the analysis procedure and the options specified, such as turbulence models and auxiliary transport equations. For example, using the energy equation in conjunction with the incompressible flow procedure activates the velocity, pressure, and temperature degrees of freedom. For a complete listing of the available degrees of freedom, see "Active degrees of freedom" in "Boundary conditions in Abaqus/CFD," Section 34.3.2.

6.6.2 INCOMPRESSIBLE FLUID DYNAMIC ANALYSIS

Products: Abaqus/CFD Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Fluid dynamic analysis procedures: overview," Section 6.6.1

Overview

An incompressible fluid dynamics analysis:

- is one where the velocity field is divergence-free and the pressure does not contain a thermodynamic component;
- is one where the energy contained in acoustic waves is small relative to the energy transported by advection (i.e., when the Mach number is in the range $0 \leq M \leq 0.1 - 0.3$);
- can be either laminar or turbulent, steady or time-dependent;
- can be used to study either internal or external flows;
- can include energy transport and buoyancy forces;
- can be used with a deforming mesh for ALE calculations; and
- can be performed with conjugate heat transfer or fluid-structure interaction.

Incompressible fluid dynamic analysis

Incompressible flow is one of the most frequently encountered flow regimes encompassing a diverse set of problems that include: atmospheric dispersal, food processing, aerodynamic design of automobiles, biomedical flows, electronics cooling, and manufacturing processes such as chemical vapor deposition, mold filling, and casting.

You can perform a transient or steady-state incompressible flow analysis.

Input File Usage: Use the following option for a transient incompressible flow analysis:

*CFD, INCOMPRESSIBLE NAVIER STOKES

Use the following option for a steady-state incompressible flow analysis:

*CFD, INCOMPRESSIBLE NAVIER STOKES, STEADY STATE

Abaqus/CAE Usage: In Abaqus/CAE you can only define a transient incompressible flow analysis.

Step module: **Create Step**: **General**: **Flow**; **Flow type**: **Incompressible**

Governing equations

The unsteady momentum equations in integral form for an arbitrary control volume can be written as

$$\frac{d}{dt}\int_V \rho\mathbf{v}\,dV + \int_S \rho\mathbf{v}\otimes(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\,dS = -\int_V \nabla p\,dV + \int_S \boldsymbol{\tau}\cdot\mathbf{n}\,dS + \int_V \mathbf{f}\,dV.$$

For steady state the integral form of the momentum conservation equation becomes

$$\int_S \rho\mathbf{v}\otimes\mathbf{v}\cdot\mathbf{n}\,dS = -\int_V \nabla p\,dV + \int_S \boldsymbol{\tau}\cdot\mathbf{n}\,dS + \int_V \mathbf{f}\,dV,$$

where

V	is an arbitrary control volume with surface area S,
$\mathbf{n}$	is the outward normal to S,
ρ	is the fluid density,
p	is the pressure,
$\mathbf{v}$	is the velocity vector,
$\mathbf{v}_m$	is the velocity of the moving mesh,
$\mathbf{f}$	is the body force, and
$\boldsymbol{\tau}$	is the viscous shear stress.

The viscous shear stress, $\boldsymbol{\tau}$, is also referred to as the deviatoric stress, $\mathbf{S}$, where $\mathbf{S} = \eta\dot{\boldsymbol{\gamma}}$. For more information, see "Viscosity," Section 26.1.4.

Incompressibility requires a solenoidal velocity field expressed by

$$\nabla\cdot\mathbf{v} = \mathbf{0}.$$

Numerical implementation

The solution of the incompressible Navier-Stokes equations poses a number of algorithmic issues due to the divergence-free velocity condition and the concomitant spatial and temporal resolution required to achieve solutions in complex geometries for engineering applications. The Abaqus/CFD incompressible solver uses the integral form of the conservation equations. For time-dependent problems, an advanced second-order projection method is used for an arbitrary deforming domain. For steady state the solution approach is based on a SIMPLE algorithm on a fixed mesh. For both the projection and SIMPLE algorithms, a node-centered finite-element discretization for the pressure and a cell-centered finite volume discretization of all the other transported variables (such as velocity, temperature, turbulence, etc.) are adopted. This hybrid approach guarantees accurate solutions and eliminates the possibility of spurious pressure modes (without the need for any artificial dissipation) while retaining the local conservation properties associated with traditional finite volume methods. An edge-based implementation is used for all transport equations permitting a single implementation that spans a broad variety of element topologies ranging from simple tetrahedral and hexahedral elements to arbitrary polyhedral. In Abaqus/CFD tetrahedral, wedge, and hexahedral elements are supported.

Projection method (for transient analysis)

The basic concept for projection methods is the legitimate segregation of pressure and velocity fields for efficient solution of the incompressible Navier-Stokes equations. Over the past two decades, projection methods have found broad application for problems involving laminar and turbulent fluid dynamics, large density variations, chemical reactions, free surfaces, mold filling, and non-Newtonian behavior.

In practice, the projection is used to remove the part of the velocity field that is not divergence-free ("div-free"). The projection is achieved by splitting the velocity field into div-free and curl-free components using a Helmholtz decomposition. The projection operators are constructed so that they satisfy prescribed boundary conditions and are norm-reducing, resulting in a robust solution algorithm for incompressible flows.

SIMPLE method (for steady-state analysis)

The SIMPLE (Semi-Implicit Method for Pressure Linked Equations) method is a pressure-based method developed to efficiently simulate steady-state flows.

The primary idea behind the SIMPLE method is to create a discrete pressure correction equation by enforcing mass continuity over each cell. The divergence-free velocity field is then obtained by relating the discrete pressure correction (and, hence, the discrete pressure) field with the discrete form of the momentum equations.

Least-squares gradient estimation

The solution methods in Abaqus/CFD use a linearly complete second-order accurate least-squares gradient estimation. This permits accurate evaluation of dual-edge fluxes for both advective and diffusive processes. All transport equations in Abaqus/CFD make use of the second-order least-squares operators.

Advection methods

The advection treatment in Abaqus/CFD is edge-based, monotonicity-preserving, and preserves smooth variations to second order in space. The advection algorithm relies on a least-squares gradient estimation with unstructured-grid slope limiters that are topology independent. Sharp gradients are captured within approximately 2–3 elements; i.e., $O(3h)$, and the use of slope limiting in conjunction with a local diffusive limiter precludes over-/under-shoots in advected fields. For the transient solver, the advection terms in the momentum and transport equations can be treated either explicitly or implicitly (see the discussion in "Time incrementation" below).

Energy equation

The energy transport equation is optionally activated in Abaqus/CFD for non-isothermal flows. For small density variations, the Boussinesq approximation provides the coupling between momentum and energy equations. In turbulent flows, the energy transport includes a turbulent heat flux based on the turbulent eddy viscosity and turbulent Prandtl number. Abaqus/CFD provides a temperature-based energy equation.

The transient form of the energy equation, in temperature form, can be obtained from the first law of thermodynamics and is given by

$$\frac{d}{dt}\int_V \rho C_p \theta dV + \int_S \rho C_p \theta(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\,dS = \int_V r\,dV - \int_S \mathbf{q}\cdot\mathbf{n}dS.$$

For steady state it is given by

$$\int_S \rho C_p \theta \mathbf{v}\cdot\mathbf{n}\,dS = \int_V r\,dV - \int_S \mathbf{q}\cdot\mathbf{n}dS,$$

where C_p is the constant pressure specific heat, θ is the temperature, $\mathbf{q}$ is heat flux due to conduction defined by Fourier's law, and r is the heat supplied externally into the body per unit volume. The energy equation is solved in terms of temperature in Abaqus/CFD.

Input File Usage: Use the following option to specify an isothermal flow problem (default):

*CFD, ENERGY EQUATION=NO ENERGY

Use the following option to specify a thermal (heat) transport problem with temperature as the primary transport scalar variable:

*CFD, ENERGY EQUATION=TEMPERATURE

Abaqus/CAE Usage: Use the following option to specify an isothermal flow problem:

Step module: **Create Step**: **General**: **Flow**; **Basic** tabbed page: **Energy equation**: **None**

Use the following option to specify a thermal (heat) transport problem with temperature as the primary transport scalar variable:

Step module: **Create Step**: **General**: **Flow**; **Basic** tabbed page: **Energy equation**: **Temperature**

Turbulence models

Turbulence modeling is a pacing technology for computational fluid dynamics. There is no single universal turbulence model that can adequately handle all possible flow conditions and geometrical configurations. This is complicated by the plethora of turbulence models and modeling approaches that are currently available; e.g., Reynolds Averaged Navier-Stokes (RANS), Unsteady Reynolds Averaged Navier-Stokes (URANS), Large-Eddy Simulation (LES), Implicit Large-Eddy Simulation (ILES), and hybrid RANS/LES (HRLES). Ultimately, you must ensure that the approximations made in a given turbulence model are consistent with the physical problem being modeled.

The following turbulent flow models are available in Abaqus/CFD: Spalart-Allmaras (SA), k–ε RNG, k–ε realizable, and k–ω SST. These models span a relatively broad set of flow problems that include steady-state and time-dependent flows, fluid-structure interaction (FSI), and conjugate heat transfer (CHT).

Spalart-Allmaras (SA) turbulence model

The Spalart-Allmaras (SA) model is a one-equation turbulence model that uses an eddy-viscosity variable with a nonlinear transport equation. The model was developed based on empiricism, dimensional analysis, and a requirement for Galilean invariance. The model has found broad use and has been calibrated for two-dimensional mixing layers, wakes, and flat-plate boundary layers. The model produces reasonably accurate predictions of turbulent flows in the presence of adverse pressure gradients and can be used for flows where separation occurs. This model is spatially local and requires only moderate resolution in boundary layers. Although initially designed for external and free-shear flows, the Spalart-Allmaras model can also be used for internal flows.

The basic form of the one-equation Spalart-Allmaras model consists of one transport equation for the turbulent eddy viscosity, $\widetilde{\nu}$. The model requires the normal distance from the wall used in the damping functions needed to control the turbulent viscosity in the near-wall region. Abaqus/CFD automatically computes the normal distance function, permitting simple specification of the model boundary conditions.

The transient form of the turbulent viscosity transport equation for the Spalart-Allmaras model is given by

$$\frac{d}{dt}\int_V \rho\widetilde{\nu}\,dV + \int_S \rho\widetilde{\nu}(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\,dS = \int_V \rho c_{b1}\widetilde{S}\widetilde{\nu}\,dV - \int_V \rho c_{w1} f_w \left(\frac{\widetilde{\nu}}{d}\right)^2 dV$$

$$+\int_V \frac{\rho(1+c_{b2})}{\sigma}\nabla\cdot\{(\nu+\widetilde{\nu})\nabla\widetilde{\nu}\}\,dV - \int_V \frac{\rho c_{b2}}{\sigma}(\nu+\widetilde{\nu})\nabla\cdot\nabla\widetilde{\nu}\,dV.$$

The steady-state form of the equation is given by

$$\int_S \rho\widetilde{\nu}\mathbf{v}\cdot\mathbf{n}\,dS = \int_V \rho c_{b1}\widetilde{S}\widetilde{\nu}\,dV - \int_V \rho c_{w1} f_w \left(\frac{\widetilde{\nu}}{d}\right)^2 dV$$

$$+\int_V \frac{\rho(1+c_{b2})}{\sigma}\nabla\cdot\{(\nu+\widetilde{\nu})\nabla\widetilde{\nu}\}\,dV - \int_V \frac{\rho c_{b2}}{\sigma}(\nu+\widetilde{\nu})\nabla\cdot\nabla\widetilde{\nu}\,dV.$$

The damping functions and model coefficients used in the above two equations are defined as:

$$f_w = g\left(\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right)^{\frac{1}{6}},$$

$$f_{v1} = \frac{\chi^3}{\chi^3+c_{v1}^3},$$

$$f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}},$$

$$\chi = \frac{\widetilde{\nu}}{\nu},$$

$$g = r + c_{w2}(r^6 - r),$$

$$r = \frac{\widetilde{\nu}}{\widetilde{S}\kappa^2 d^2},$$

$$\widetilde{S} = S + \frac{\widetilde{\nu}}{\kappa^2 d^2} f_{v2},$$

$$S = \sqrt{2R_{ij}R_{ij}},$$

$$R_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right),$$

where d is the normal distance from the wall, and the effective turbulent viscosity is defined as

$$\nu_t = \widetilde{\nu} f_{v1}.$$

The Spalart-Allmaras model coefficients are shown in Table 6.6.2–1. In addition, a turbulent Prandtl number, Pr_t, can be specified.

Table 6.6.2–1 Spalart-Allmaras model coefficients.

c_{b1}	c_{b2}	c_{v1}	σ	c_{w1}	c_{w2}	c_{w3}	κ	c_{v2}
0.1355	0.622	7.1	0.6667	$\frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}$	0.3	2	0.41	5

The Spalart-Allmaras model can provide very accurate boundary layer results if the near-wall region is resolved (near-wall resolution such that the nondimensional wall distance is approximately 3). However, the implementation of boundary conditions for the Spalart-Allmaras model in Abaqus/CFD permits the use of coarser meshes as well.

Input File Usage: Use both of the following options:

*CFD
*TURBULENCE MODEL, TYPE=SPALART ALLMARAS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Turbulence**
tabbed page: **Spalart-Allmaras**

Wall functions

The Spalart-Allmaras turbulence model can be integrated throughout the inner layer of the turbulent boundary layer due to its built-in low Reynolds number damping functions. However, the model usually requires extremely fine near-wall resolutions on the order of $y^+ < 2$ to accurately predict the eddy viscosity in the entire boundary layer. In general, the $y^+ < 2$ near-wall resolution requirement is a very stringent constraint to enforce in complex high Reynolds number flow problems. Consequently, a wall-function approach is implemented to relax the near-wall resolution required by the Spalart-Allmaras model.

The conventional wall-function approach is based on the law-of-the-wall, which is a semi-empirical universal velocity profile obtained in equilibrium wall-bounded flows when the flow velocity, V, and wall-normal distance, y, are normalized with the kinematic viscosity, ν, and friction velocity, v_τ (known as viscous units or wall units): The friction velocity is computed as follows:

y^* is computed as follows:

$$y^*_{(\chi)} = \frac{\chi}{\kappa}.$$

The Law-of-the-wall is computed $V^+ = V^+_{(y^*)}$. The friction velocity can be computed from the viscous and logarithmic layer relationships as

$$v_\tau = g v_{\tau Vis} + (1 - g) v_{\tau Log}.$$

$$v_{\tau Vis} = \frac{\nu y^*_{(\chi)}}{y},$$

$$v_{\tau Log} = \frac{V}{V^+_{(y^*(\chi))}}.$$

The blending function is defined as

$$g = e^{-Rey/y^+_c},$$

$$Rey = C_\mu^{1/4} y^*(\chi).$$

The wall function is defined as

$$V^+_{(y^+)} = \begin{cases} y^+ & \text{if } y^+ \le y^+_c \\ \frac{1}{\kappa}\ln(Ey^+) & \text{if } y^+ > y^+_c, \end{cases}$$

$$V^+ = \frac{V}{v_\tau},$$

$$y^+ = \frac{y v_\tau}{\nu},$$

$$v_\tau = \sqrt{\frac{\tau_{wall}}{\rho}}.$$

In the above equations ρ is the density, τ_{wall} is the shear stress at the wall, y_c^+ is the intersection point of the linear and logarithmic velocity profile, and $\kappa = 0.41$ and $E = 8.4$ are constants.

The conventional wall-function approach has been adapted within Abaqus/CFD to a form that asymptotes to a standard wall function for coarse meshes, yet will also give results identical to a wall-function-free approach for fine meshes. It is termed a hybrid wall function. The key aspect in the implementation of the hybrid wall-function method within the Spalart-Allmaras model is to obtain the friction velocity as a function of $\widetilde{\nu}$ and the local velocity field, which was shown before. Having computed the friction velocity, the wall shear can be obtained as

$$\tau_{wall} = \rho v_\tau \frac{V}{V^+(y^+)},$$

$$y^+ = \frac{y v_\tau(\chi)}{\nu}.$$

The hybrid wall-function approach is independent of the near-wall resolution; therefore, the law-of-the-wall $V^+(y^+)$ implemented needs to accurately predict the viscous-sublayer, the logarithmic-layer, and the buffer layer (the region that connects the viscous and logarithmic zones) since the cell-center adjacent to the wall can be located anywhere within the inner layer. Therefore, a single smooth correlation that reproduces the entire law-of-the-wall proposed by Reichardt (1951) is implemented.

$$V^+(y^+) = \frac{1}{\kappa}\ln(1+\kappa y^+) + C\Big[1 - e^{-\frac{y^+}{y_c^+}} - \frac{y^+}{y_c^+}e^{-by^+}\Big],$$

where

$$C = \frac{1}{\kappa}\ln\left(\frac{E}{\kappa}\right),$$

$$b = \frac{1}{2}\left(\frac{y_c^+\kappa}{C} + \frac{1}{D}\right).$$

Implementation in the momentum equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-shear stress in coarse meshes. The wall shear is obtained from the wall-function approach through an effective edge viscosity:

$$\mu_{\text{eff}} = \frac{\tau_{wall} y}{V} = \left(\frac{\rho v_\tau V}{V^+}\right)\frac{y}{V}.$$

Energy wall functions

The wall-function approach can be extended to the energy equation by using the temperature law-of-the-wall, which is a semi-empirical universal temperature profile obtained in equilibrium wall-bounded flows when the temperature, T, and wall-normal distance, y, are normalized with wall units. The standard temperature wall function is defined as

$$T^+(y^+) = \frac{(T_w - T)\rho c_p u_\tau}{\dot{q}} = \begin{cases} P_r y^+ & \text{if } y^+ \le y^+_{Tc} \\ P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right) & \text{if } y^+ > y^+_{Tc}, \end{cases}$$

where $y^+_{T_c}$ is the intersection point of the viscous-sublayer and the logarithmic layer in the temperature wall function, T_w is the wall temperature, P_r is the Prandtl number, P_{rT} is the turbulent Prandtl number, $\dot{q}$ is the wall-heat flux, c_p is the specific heat coefficient at constant pressure, and P is computed using the Jayatilleke (1969) expression:

$$P(P_r, P_{rT}) = 9.24\left[\left(\frac{P_r}{P_{rT}}\right)^{3/4} - 1\right]\left(1 + 0.28e^{-0.007P_r/P_{rT}}\right).$$

For the hybrid wall-function approach a continuous temperature wall function proposed by Kader (1981) is implemented:

$$T^+(y^+) = e^{-\Gamma}P_r y^+ + e^{-1/\Gamma}P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right)$$

with the blending function defined as

$$\Gamma = \frac{0.01(P_r y^+)^4}{1 + 5P_r^3 y^+}.$$

Finally, the heat flux is obtained from the precomputed v_τ flow properties temperature field and the continuous temperature wall function:

$$\dot{q} = \frac{(T_w - T)\rho c_p v_\tau}{T^+}.$$

Implementation in the energy equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-heat flux in coarse meshes. The wall-heat flux is obtained from the wall-function approach through an effective edge heat conductivity.

$$\kappa_{\text{eff}} = \dot{q}\frac{y}{(T_w - T)}.$$

k–epsilon RNG turbulence model

The ***k***–ε RNG model is a two-equation turbulence model that evolves an equation for the turbulent kinetic energy, ***k***, and the energy dissipation rate, ε. The model equations are developed from fundamental physical principles and dimensional analysis; the equation for ***k*** is derived using first principles, and the equation for ε is postulated using physical insight. The main advantage of the RNG version is that the model coefficients are obtained using a mathematical approach called Renormalization Group theory, commonly used in physics. This calibration method removes most of the uncertainty introduced when the model coefficients are calibrated using a finite set of canonical flows as is done in most turbulence RANS models. In addition, it changes the epsilon equation (Yakhot et al., 1992). The model equations are as follows:

$$\frac{d}{dt}\int_V \rho k\, dV + \int_S \rho k(\mathbf{v} - \mathbf{v}_m)\cdot \mathbf{n}\, dS$$

$$= \int_S \frac{(\mu + \mu_T)}{\sigma_k}\nabla k \cdot \mathbf{n}\, dS + \int_V (\tau_{ij}S_{ij} + G_b)\, dV - \int_V \rho\varepsilon\, dV,$$

$$\frac{d}{dt}\int_V \rho\varepsilon\, dV + \int_S \rho\varepsilon(\mathbf{v} - \mathbf{v}_m)\cdot \mathbf{n}\, dS$$

$$= \int_S \frac{(\mu + \mu_T)}{\sigma_\varepsilon}\nabla \varepsilon \cdot \mathbf{n}\, dS + \int_V C_{\varepsilon_1}\frac{\varepsilon}{k}(\tau_{ij}S_{ij} + C_{\varepsilon 3}G_b)\, dV - \int_V C_{\varepsilon_2}\frac{\rho\varepsilon^2}{k}\, dV,$$

where τ_{ij} is the Reynolds stress tensor and is closed as follows:

$$\tau_{ij} = 2\mu_T S_{ij},$$

$$S_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}.$$

The turbulent viscosity, μ_T, is

$$\mu_T = \frac{\rho C_\mu k^2}{\varepsilon},$$

$$C_{\varepsilon_2} = \widetilde{C}_{\varepsilon_2} + \frac{C_\mu \eta^3(1 - \eta/\eta_0)}{1 + \beta\eta^3},$$

and

$$\eta = \frac{k}{\varepsilon}\sqrt{2S_{ij}S_{ij}}.$$

The second and third terms on the right-hand-side of the k–ε transport equations above represent the production and dissipation of k and ε, respectively.

The k–ε RNG model coefficients are shown in Table 6.6.2–2. In addition, a turbulent Prandtl number (Pr_t) can be specified.

Table 6.6.2–2 k–ε RNG model coefficients.

C_μ	C_{ε_1}	$\widetilde{C}_{\varepsilon_2}$	σ_k	σ_ε	β	η_0
0.085	1.42	1.68	0.72	0.72	0.012	4.38

Input File Usage: Use both of the following options:

∗CFD
∗TURBULENCE MODEL, TYPE=RNG KEPSILON

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Turbulence** tabbed page: **k-epsilon renormalization group (RNG)**

Wall functions

It is well known that the k–ε model has limitations, especially on wall-bounded flows where high values of eddy viscosity in the near-wall region are usually reproduced. For high Reynolds number flows often encountered in many industrial applications, a full resolution of the thin viscous sub-layer that occurs near a wall using a fine mesh may not be economical. Consequently, for meshes that cannot resolve the viscous sub-layer, wall functions are used to represent the effects of the viscous sub-layer on the transport processes. In Abaqus/CFD wall functions are used to avoid the need for highly resolved boundary layer meshes. This approach relies on the law of the wall to obtain the wall-shear stress.

The law of the wall is a universal velocity profile that wall-bounded flows develop in the absence of pressure gradients. The law of the wall is

$$V^+ = \begin{cases} y^+ & \text{if } y^+ \le y_c^+ \\ \frac{1}{\kappa}\ln(Ey^+) & \text{if } y^+ > y_c^+, \end{cases}$$

where

$$V^+ = \frac{V}{v_\tau},$$

$$y^+ = \frac{yv_\tau}{\nu},$$

$$v_\tau = \sqrt{\frac{\tau_{wall}}{\rho}}.$$

V is the wall tangent velocity, ν is the kinematic viscosity, ρ is the density, τ_{wall} is the shear stress at the wall, y_c^+ is the intersection point of the linear and logarithmic velocity profile, and $\kappa = 0.41$ and $E = 8.4$ are constants.

The standard law of the wall profile is limited in its usage. For example, in recirculating flows the turbulent kinetic energy $\boldsymbol{k}$ becomes zero at separation and reattachment points, where, by definition, v_τ is zero. This singular behavior causes the predicted results to be erroneous. To overcome this, the standard law of the wall is modified based on a new scale for the friction velocity following the method proposed by Launder and Spalding (1974). The modified friction velocity is given by

$$v^* = C_\mu^{1/4} k^{1/2},$$

which does not suffer from a singular behavior at flow reattachment, separation, and at points of flow impingement. Correspondingly, the wall distances are re-scaled as follows:

$$y^* = \frac{y v^*}{\nu} = \frac{y C_\mu^{1/4} k^{1/2}}{\nu}.$$

The modified law of the wall reduces to the standard law of the wall under the conditions of uniform wall-shear stress and when the generation and dissipation of turbulent kinetic energy are in balance (i.e., when the turbulence structure is in equilibrium). Under such conditions, $v^* \approx v^+$ and, thus, $y^* \approx y^+$.

The wall-shear stress for the modified law of the wall can be evaluated as (Albets-Chico, et al., 2008)

$$\tau_{wall} = \begin{cases} \frac{\mu V_p}{y_p} & \text{if } y^* \le y_c^+ \\ \frac{\kappa \rho V_p v^*}{\ln\left[E y^*\right]} & \text{if } y^* > y_c^+, \end{cases}$$

where the subscript $\boldsymbol{p}$ denotes the wall element center at which all the quantities of interest are evaluated. The use of the wall function requires the modification of the transport equations for $\boldsymbol{k}$ and ϵ for the wall layer of elements. Specifically, the production and dissipation terms in the governing transport equation for the turbulent kinetic energy $\boldsymbol{k}$ are modified to account for the presence of the wall.

Following the procedure outlined in Craft et al. (2002), an average value of the production of $\boldsymbol{k}$ as given below is used in the transport equation. Such an average is obtained based on a two-layer model of the wall element (i.e., the wall element is divided into a partly viscous sub-layer region and a partly turbulent log-layer or inertial layer region).

$$\text{Average Production}(k) = \begin{cases} 0 & \text{if } y^* \le y_c^+ \\ \frac{\tau_{wall}^2}{\rho \kappa C_\mu^{1/4} k^{1/2} y_n} \ln\left(\frac{y_n}{y_v}\right) & \text{if } y^* > y_c^+, \end{cases}$$

where y_n is the maximum of the wall normal distances of all the vertices of a given wall element, and y_v is the wall normal distance of the edge of the viscous sub-layer, where

$$y_v^* = \frac{C_\mu^{1/4} k^{1/2}}{\nu} y_v = y_c^+.$$

Similarly, an average value of the dissipation rate for $\boldsymbol{k}$ is also prescribed for the wall elements based on a two-layer integration and is given by

$$\text{Average Dissipation}(k) = \begin{cases} \frac{2\mu k}{y_v^2} & \text{if } y^* \le y_c^+ \\ \frac{2\mu k}{y_n y_v} + \frac{\rho C_\mu^{3/4} k^{3/2}}{\kappa y_n} \ln\left(\frac{y_n}{y_v}\right) & \text{if } y^* > y_c^+. \end{cases}$$

The transport equation for ε is not solved for the wall layer elements. Instead, the value of ε is directly prescribed at the point $\boldsymbol{p}$ as follows:

$$\varepsilon = \begin{cases} \frac{2\nu k}{y_v^2} & \text{if } y^* \le y_c^+ \\ \frac{C_\mu^{3/4} k^{3/2}}{\kappa y_p} & \text{if } y^* > y_c^+. \end{cases}$$

Therefore, integration of the $\boldsymbol{k}$ and ε transport equations is performed with a zero flux (i.e., homogeneous Neumann boundary conditions) at the walls.

Guidelines on wall functions

The main advantage of wall functions is the relaxed requirement on mesh resolution at walls. However, the main disadvantage of using wall functions is the dependence on the near-wall mesh resolution. Wall functions based on the law of the wall approach usually work best for wall elements whose centers lie in the fully turbulent layer (inertial or log layer) for which such functions are designed. This effectively imposes a lower limit on the value of the scaled wall coordinate, y^*. For complex geometries, ensuring that all the near wall cells are outside the viscous sublayer is difficult. The precise location of the logarithmic region is solution dependent and may vary with time. To accommodate a more flexible mesh, a resolution-insensitive wall function (Durbin, 2009) has been implemented. Briefly, this wall function is based on limiting the minimum value of y^* such that the value of the velocity gradient at the first wall-attached element is the same as if it was located on the edge of the viscous sub-layer. A best practice for wall-bounded flows is to have at least 8–10 points in the boundary layer region where $y^* \le 300$ (see Casey and Wintergerste, 2000).

Modification of the momentum equation

Using wall functions, the calculation of the wall-shear stress or the viscous flux of momentum based on the molecular viscosity μ and the numerically estimated velocity gradient can produce large errors. Hence, proper modifications to the momentum equations need to be introduced to account for the poorly resolved excess wall friction. The necessary modification is done through a modified viscosity for the wall elements that corrects for the erroneous estimate of the velocity gradient (Bredberg, 2000). This is implemented for the wall elements as:

$$\mu_{\text{eff}} = \begin{cases} \mu & \text{if } y^* \le y_c^+ \\ \frac{\rho \kappa C_\mu^{1/4} {k_p}^{1/2} y_p}{\ln\left(E {y_p}^*\right)} & \text{if } y^* > y_c^+. \end{cases}$$

.

Energy wall functions

The wall-function approach can be extended to the energy equation by using the temperature law-of-the-wall, which is a semi-empirical universal temperature profile obtained in equilibrium wall-bounded flows when the temperature, T, and wall-normal distance, y, are normalized with wall units. The standard temperature wall function is defined as

$$T^+(y^+) = \frac{(T_w - T)\rho c_p u_\tau}{\dot{q}} = \begin{cases} P_r y^+ & \text{if } y^+ \le y_{Tc}^+ \\ P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right) & \text{if } y^+ > y_{Tc}^+, \end{cases}$$

where $y_{T_c}^+$ is the intersection point of the viscous-sublayer and logarithmic layer in the temperature wall function, T_w is the wall temperature, P_r is the Prandtl number, P_{rT} is the turbulent Prandtl number, $\dot{q}$ is the wall-heat flux, c_p is the specific heat coefficient at constant pressure, and P is computed using the Jayatilleke (1969) expression:

$$P(P_r, P_{rT}) = 9.24\left[\left(\frac{P_r}{P_{rT}}\right)^{3/4} - 1\right]\left(1 + 0.28e^{-0.007P_r/P_{rT}}\right).$$

Finally, the heat flux is obtained from the precomputed v_τ flow properties temperature field and the continuous temperature wall function:

$$\dot{q} = \frac{(T_w - T)\rho c_p v_\tau}{T^+}.$$

Implementation in the energy equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-heat flux in coarse meshes. The wall-heat flux is obtained from the wall-function approach through an effective edge heat conductivity:

$$\kappa_{\text{eff}} = \dot{q}\frac{y}{(T_w - T)}.$$

k–epsilon realizable turbulence model

The k–ε realizable model is a two-equation turbulence model that evolves an equation for the turbulent kinetic energy, k, and the energy dissipation rate, ε. The model equations are developed from fundamental physical principles and dimensional analysis; the equation for k is derived using first principles, and the equation for ε is postulated using physical insight. This particular version uses realizability constraints,

which imposes mathematical consistency in the Reynolds stresses (such as enforcing the positivity of the normal stresses and the Cauchy-Schwarz inequality) to modify the model coefficients and the epsilon equation. These modifications guarantee the physical consistency in the predicted Reynolds stresses, thus improving the accuracy of the predictions (Shih et al., 1995). The model equations are as follows:

$$\frac{d}{dt}\int_V \rho k\, dV + \int_S \rho k(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\, dS$$

$$= \int_S \Big(\mu + \frac{\mu_T}{\sigma_k}\Big)\nabla k\cdot\mathbf{n}\, dS + \int_V (\tau_{ij}S_{ij} + G_b)\, dV - \int_V \rho\varepsilon\, dV,$$

$$\frac{d}{dt}\int_V \rho\varepsilon\, dV + \int_S \rho\varepsilon(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\, dS$$

$$= \int_S \Big(\mu + \frac{\mu_T}{\sigma_\varepsilon}\Big)\nabla\varepsilon\cdot\mathbf{n}\, dS + \int_V C_{\varepsilon 1}(\rho S\varepsilon + \frac{\varepsilon}{k}C_{\varepsilon 3}G_b)\, dV - \int_V C_{\varepsilon 2}\frac{\rho\varepsilon^2}{(k+\sqrt{\nu\varepsilon})}\, dV,$$

where the Reynolds stress tensor is defined as

$$\tau_{ij} = 2\mu_T S_{ij},$$

with the eddy viscosity, μ_T, defined as

$$\mu_T = C_\mu \frac{\rho k^2}{\varepsilon}.$$

The turbulence viscosity coefficient is computed using realizability constraints:

$$C_\mu = \frac{1}{A_0 + A_s U^{(*)}\frac{k}{\varepsilon}},$$

where

$$U^{(*)} = \sqrt{S_{ij}S_{ij} + R_{ij}R_{ij}},$$

$$A_s = \sqrt{6}\cos(\phi), \quad \phi = \frac{1}{3}\arccos(\sqrt{6}W),$$

$$W = \frac{S_{ij}S_{jk}S_{ki}}{\tilde{S}^3}, \quad S = \sqrt{2S_{ij}S_{ij}}, \quad \tilde{S} = \frac{S}{\sqrt{2}},$$

$$S_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i},$$

$$R_{ij} = \frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}.$$

Realizability conditions are imposed in the the production coefficient of the ε equation, which yields

$$C_{\varepsilon 1} = \max\left(C_{\varepsilon 1 max}, \frac{\eta}{5+\eta}\right),$$

$$\eta = \frac{Sk}{\varepsilon}.$$

The k–ε realizable model coefficients are shown in Table 6.6.2–2. In addition, a turbulent Prandtl number, Pr_t, can be specified.

Table 6.6.2–3 k–ε realizable model coefficients.

A_0	$C_{\varepsilon 2}$	σ_k	σ_ε	$C_{\varepsilon 1 max}$
4.0	1.9	1.0	1.2	0.43

Input File Usage: Use both of the following options:

*CFD
*TURBULENCE MODEL, TYPE=KEPSILON REALIZABLE

Abaqus/CAE Usage: The k–ε realizable turbulence model is not supported in Abaqus/CAE.

Two-layer model

To increase the accuracy of the k–ε realizable model in cases where the mesh is not fine enough to resolve the inner layer in wall-bounded flows, a near-wall model, known as the two-layer model, is implemented. Following the work of Chen and Patel (1988), algebraic equations for ε and μ_T are used in the inner layer to replace their turbulence model equations:

$$\varepsilon_{2\text{layer}} = \frac{k^{3/2}}{l_\varepsilon},$$

$$l_\varepsilon = C_l y\left(1 - e^{(-Re_y/A_\varepsilon)}\right),$$

where

$$A_\varepsilon = 2C_l, \quad C_l = \kappa C_\mu^{-3/4}.$$

The algebraic equation μ_T is

$$\mu_{\text{T2layer}} = \mu Re_y C_\mu^{1/4} \kappa \left[1 - e^{(-Re_y/A_\mu)}\right],$$

$$Re_y = \frac{\rho k^{1/2} y}{\mu},$$

where y is the wall-normal distance. The two-layer approach combines the algebraic equation with the ε transport equation. To provide a smooth transition, the work of Jongen (1998) is followed by implementing a smooth blending function to bridge both formulations:

$$\lambda_\varepsilon = \frac{1}{2}\left[1 + \tanh\left(\frac{\mathrm{Re_y} - \mathrm{Re_y^*}}{\mathrm{A}}\right)\right],$$

where A defines the width of the blending function. We define a width so λ_ε will be within 1% of its far-field value. Given a variation of ΔRe_y, one obtains

$$A = \frac{|\Delta Re_y|}{\tanh 0.98}.$$

The final blending of ε is done through combining the transport and algebraic equations at the discretized level:

$$a_p \varepsilon_p + \lambda_\varepsilon \Sigma_{nb} a_{nb} \varepsilon_{nb} = \lambda_\varepsilon S_\varepsilon + (1 - \lambda_\varepsilon) a_p \varepsilon_{\mathrm{2layer}},$$

where a_p and a_{nb} are the diagonal and off-diagonal terms, respectively, and S_ε is the right-hand side of the discrete transport equation. Finally, the blending of the eddy viscosity is conducted in a straightforward way since it only involves two algebraic equations:

$$\mu_T' = \lambda_\varepsilon \mu_T + (1 - \lambda_\varepsilon)\mu_{\mathrm{T2layer}}.$$

Wall functions

The two-layer approach takes care of the near-wall integration of the ε equation; however, it does not provide any modeling for the turbulent kinetic energy equation. Therefore, to model the budget of the turbulent kinetic energy at the inner layer correctly, an additional near-wall treatment needs to go into the $\boldsymbol{k}$ equation. A hybrid-wall function treatment is implemented to integrate the $\boldsymbol{k}$ equation throughout the inner layer. The hybrid wall function uses the law-of-the-wall and equilibrium assumptions to approximate the sources of $\boldsymbol{k}$ when the mesh resolution is not fine enough to resolve the inner layer. However, if the mesh is fine enough to resolve the entire inner-layer, the hybrid-wall function recovers the viscous sublayer relations for the $\boldsymbol{k}$ sources. The law-of-the-wall and viscous sublayer sources are finally blended to provide a smooth formulation that allows the model to provide accurate solutions irrespective of the mesh's near-wall resolution.

The-law-of-the-wall is a universal velocity profile that wall-bounded flows develop in the absence of pressure gradients. The velocity profile consists of two well-defined regions:

$$V^+ = \begin{cases} y^+ & \text{if } y^+ \le y_c^+ \quad \text{Viscous sublayer} \\ \frac{1}{\kappa}\ln(Ey^+) & \text{if } y^+ > y_c^+ \quad \text{Logarithmic layer,} \end{cases}$$

where

$$V^+ = \frac{V}{v_\tau},$$

$$y^+ = \frac{yv_\tau}{\nu},$$

$$v_\tau = \sqrt{\frac{\tau_{wall}}{\rho}}.$$

V is the wall tangent velocity, ν is the kinematic viscosity, ρ is the density, τ_{wall} is the shear stress at the wall, y_c^+ is the intersection point of the viscous sublayer and the logarithmic layer, $\kappa = 0.41$ is the Von Karman constant, and $E = 8.4$ is the law-of-the-wall's logarithmic constant. The viscous sublayer is the region where momentum transfer is dominated by molecular diffusion; turbulence here is virtually absent. The logarithmic layer or fully turbulent regime establishes the onset of the fully turbulent regime; here momentum transport is fully dominated by turbulent transport relegating molecular diffusion to a second-order effect.

The standard law of the wall profile is limited in its usage. For example, in recirculating flows the turbulent kinetic energy, k, becomes zero at separation and reattachment points, where, by definition, v_τ is zero. This singular behavior causes the predicted results to be erroneous. To overcome this, the standard law of the wall is modified based on a new scale for the friction velocity following the method proposed by Launder and Spalding (1974). The modified friction velocity is given by

$$v^* = C_\mu^{1/4} k^{1/2},$$

which does not suffer from a singular behavior at flow reattachment, separation, and at points of flow impingement. Correspondingly, the wall distances are re-scaled as follows:

$$y^* = \frac{yv^*}{\nu} = \frac{yC_\mu^{1/4}k^{1/2}}{\nu}.$$

The modified law of the wall reduces to the standard law of the wall under the conditions of uniform wall-shear stress and when the generation and dissipation of turbulent kinetic energy are in balance (i.e., when the turbulence structure is in equilibrium). Under such conditions, $v^* \approx v^+$ and, thus, $y^* \approx y^+$.

The wall-shear stress for the modified law of the wall can be evaluated as (Albets-Chico, et al., 2008)

$$\tau_{wall} = \rho u^* \frac{V_p}{V^+(y_p^*)},$$

where the subscript p denotes the wall element center at which all the quantities of interest are evaluated. Using the wall function the k sources are derived to correct the turbulent kinetic energy budget at the inner layer as follows:

$$\text{Production}(k) = \begin{cases} 0.0 & \text{if } y^* \le y_c^+ \\ \frac{\tau_{wall}^2}{\rho \kappa u^* y_n} & \text{if } y^* > y_c^+. \end{cases}$$

Following a similar procedure the value of the dissipation rate for k is given by

$$\text{Dissipation}(k) = \begin{cases} \frac{2\mu k}{y^2} & \text{if } y^* \le y_c^+ \\ \frac{\rho u^{*3}}{\kappa y} & \text{if } y^* > y_c^+. \end{cases}$$

Having defined the sources in the viscous sublayer and logarithmic layer, a blending approach is implemented to provide a smooth transition that allows us to operate on meshes with arbitrary near-wall resolution:

$$\text{Production}(k) = (1-f)\frac{\tau_{wall}^2}{\rho \kappa u^* y_n},$$

$$\text{Dissipation}(k) = \rho\left(f\frac{2\nu k}{y^2} + (1-f)\frac{u^{*3}}{\kappa y} + f(1-f)\left(\frac{2\nu k}{y^2} + \frac{u^{*3}}{\kappa y}\right)\right),$$

$$u^* = \sqrt{f\frac{\nu V_p}{y_p} + (1-f)C_\mu^{1/2}k},$$

$$f = e^{\frac{-R_{ey}}{y_c^+}},$$

where

$$R_{ey} = \frac{\sqrt{k}y}{\nu}.$$

The boundary conditions enforced at the wall for both the k and ε transport equations is a zero flux (i.e., homogeneous Neumann boundary conditions) at the walls.

Implementation in the momentum equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-shear stress in coarse meshes. The wall shear is obtained from the wall-function approach through an effective edge viscosity:

$$\mu_{\text{eff}} = \frac{\tau_{wall} y}{V} = \left(\frac{\rho v_\tau V}{V^+}\right)\frac{y}{V}.$$

Energy wall functions

The wall-function approach can be extended to the energy equation by using the temperature law-of-the-wall, which is a semi-empirical universal temperature profile obtained in equilibrium wall-bounded flows, when the temperature, $\boldsymbol{T}$, and the wall-normal distance, $\boldsymbol{y}$, are normalized with wall units. The standard temperature wall function is defined as

$$T^+(y^+) = \frac{(T_w - T)\rho c_p u_\tau}{\dot{q}} = \begin{cases} P_r y^+ & \text{if } y^+ \le y^+_{Tc} \\ P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right) & \text{if } y^+ > y^+_{Tc}, \end{cases}$$

where y^+_{Tc} is the intersection point of the viscous-sublayer and the logarithmic layer in the temperature wall-function, T_w is the wall temperature, P_r is the Prandtl number, P_{rT} is the turbulent Prandtl number, $\dot{q}$ is the wall-heat flux, c_p is the specific heat coefficient at constant pressure, and P is computed using the Jayatilleke (1969) expression:

$$P(P_r, P_{rT}) = 9.24\left[\left(\frac{P_r}{P_{rT}}\right)^{3/4} - 1\right]\left(1 + 0.28e^{-0.007P_r/P_{rT}}\right).$$

For the hybrid wall-function approach a continuous temperature wall function proposed by Kader (1981) is implemented

$$T^+(y^+) = e^{-\Gamma}P_r y^+ + e^{-1/\Gamma}P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right),$$

with the blending function defined as

$$\Gamma = \frac{0.01(P_r y^+)^4}{1 + 5P_r^3 y^+}.$$

Finally, the heat flux is obtained from the precomputed v_τ flow properties temperature field and the continuous temperature wall function:

$$\dot{q} = \frac{(T_w - T)\rho c_p v_\tau}{T^+}.$$

Implementation in the energy equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-heat flux in coarse meshes. The wall-heat flux is obtained from the wall-function approach through an effective edge heat conductivity:

$$\kappa_{\text{eff}} = \dot{q}\frac{y}{(T_w - T)}.$$

k–omega SST turbulence model

The k–ω SST turbulence model is a two-equation turbulence model that uses a different dissipation parameter than the k–ε turbulence model; namely, the specific energy dissipation rate, $\omega \approx \varepsilon/k$ introduced by Menter in 1994. Although ω varies even more rapidly than ε and takes on very large values near the wall, the model is relatively insensitive to the wall values. One of the most attractive properties of k–ω models is that they can be employed throughout the viscous sublayer without further modification (as would required by k–ε models). However, a weakness of the standard k–ω model is that it is much more sensitive than k–ε to the values of turbulence parameters in the freestream and, hence, to inflow turbulence values.

Menter (1994) introduced a hybrid model that blends the standard k–ω model near to the wall with a transformed version of the standard k–ε model (into k–ω form) far from the wall. This hybrid model adds an additional cross-diffusion term, in regions away from the wall, to the transport equation for a specific dissipation rate. It is this term that reduces the sensitivity of the model to freestream turbulence values. To complete the k–ω SST model, Menter further added a limiter for turbulent shear stress that prevents excessive shear stress levels in boundary layers. The k–ω SST model essentially contains two sets of coefficients, one for the k–ω part and one for the (transformed) k–ε part. The blending is achieved by functions of two different turbulent Reynolds numbers, which, in turn, require the normal distance from the wall. Abaqus/CFD automatically computes this distance function. The transport equations for the model are

$$\frac{d}{dt}\int_V \rho k dV + \int_S \rho k(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\,dS = \int_S (\mu+\sigma_k\mu_T)\nabla k\cdot\mathbf{n}\,dS$$

$$+\int_V (\tau_{ij}S_{ij}+G_b)\,dV - \int_V \beta^*\rho\omega k\,dV,$$

$$\frac{d}{dt}\int_V \rho\omega\,dV + \int_S \rho\omega(\mathbf{v}-\mathbf{v}_m)\cdot\mathbf{n}\,dS = \int_S (\mu+\sigma_\omega\mu_T)\nabla\omega\cdot\mathbf{n}\,dS$$

$$+\int_V \frac{\rho\gamma}{\mu_T}(\tau_{ij}S_{ij}+C_{\omega 3}G_b)\,dV - \int_V \beta\rho\omega^2\,dV + \int_V 2(1-F_1)\rho\sigma_{\omega 2}\frac{1}{\omega}\nabla k\cdot\nabla\omega\,dV.$$

The k–ω SST model coefficients are computed by blending the original k–ω model by Wilcox (1988) and the standard k–ε model coefficients:

$$\phi = F_1\phi_1 + (1-F_1)\phi_2.$$

The blending function is defined as

$$F_1 = \tanh(\arg_1^4),$$

$$\arg_1 = \min\left(\max\left(\frac{\sqrt{k}}{0.09\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\rho\sigma_{\omega 2}k}{CD_{k\omega}y^2}\right),$$

$$CD_{k\omega} = \max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-10}\right).$$

In the incompressible formulation the Reynolds stresses are closed as

$$\tau_{ij} = 2\mu_T S_{ij},$$

$$S_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i},$$

with the turbulent eddy viscosity computed using the shear-stress transport approach:

$$\mu_T = \frac{\rho a_1 k}{\max(a_1\omega, SF_2)},$$

$$S = \sqrt{2S_{ij}S_{ij}},$$

$$F_2 = \tanh(\arg_2^2),$$

$$\arg_2 = \max\left(\frac{2\sqrt{k}}{0.09\omega y}, \frac{500\nu}{y^2\omega}\right).$$

The k–ω SST model coefficients are shown in Table 6.6.2–4 and Table 6.6.2–5. In addition, a turbulent Prandtl number, Pr_t, can be specified.

Table 6.6.2–4 k–ω SST model coefficients (Wilcox k–ω model).

σ_{k1}	$\sigma_{\omega 1}$	β_1	γ_1	κ
0.5	0.5	0.075	$\beta_1/\beta^* - \sigma_{\omega 1}\kappa^2/\sqrt{\beta^*}$	0.41

Table 6.6.2–5 k–ω SST model coefficients (standard k–ε).

σ_{k2}	$\sigma_{\omega 2}$	β_2	γ_2	β^*
1.0	0.856	0.0828	$\beta_2/\beta^* - \sigma_{\omega 2}\kappa^2/\sqrt{\beta^*}$	0.09

Input File Usage: Use both of the following options:

*CFD
*TURBULENCE MODEL, TYPE=KOMEGA SST

Abaqus/CAE Usage: The k–ω SST turbulence model is not supported in Abaqus/CAE.

Wall functions

The k–ω SST model can be integrated throughout the inner-layer of the turbulent boundary layer. However, this requires fine near-wall resolutions. The same model can also be applied using a wall-function approach, in which the near-wall element centroid is located in the logarithmic (fully turbulent) part of the boundary layer. This greatly reduces the mesh refinement requirements and, hence, computational costs, albeit with a potential loss of accuracy.

The conventional wall-function approach is based on the law-of-the-wall, which is a semi-empirical universal velocity profile obtained in equilibrium wall-bounded flows when the flow velocity, V, and wall normal distance, y, are normalized with the kinematic viscosity, ν, and friction velocity v_τ (known as viscous units or wall units):

$$V^+_{(y^+)} = \begin{cases} y^+ & \text{if } y^+ \le y^+_c \\ \frac{1}{\kappa}\ln(Ey^+) & \text{if } y^+ > y^+_c, \end{cases}$$

$$V^+ = \frac{V}{v_\tau},$$

$$y^+ = \frac{yv_\tau}{\nu},$$

$$v_\tau = \sqrt{\frac{\tau_{wall}}{\rho}},$$

where ν is the kinematic viscosity, ρ is the density, τ_{wall} is the shear stress at the wall, y^+_c is the intersection point of the linear and logarithmic velocity profile, $\kappa = 0.41$ is the Von Karman constant, and $E = 8.4$ is the law of the wall constant.

In equilibrium conditions the wall-shear stress is approximately equal to the Reynolds stress:

$$\tau_{wall} = \rho v_\tau^2 \approx \mu_T \frac{\partial v}{\partial y} = \rho \beta^* \frac{k^2}{\varepsilon} \frac{\partial v}{\partial y}$$

$$= \rho^2 \beta^* \frac{k^2}{\tau_{wall} \frac{\partial v}{\partial y}} \frac{\partial v}{\partial y} = \frac{\rho^2 \beta^* k^2}{\tau_{wall}},$$

$$\tau_{wall}^2 = (\rho v_\tau^2)^2 = \rho^2 \beta^* k^2.$$

Thus, the friction velocity in equilibrium conditions is shown to be

$$v_\tau = \beta^{*1/4} k^{1/2}.$$

The wall-shear stress can be linearized with the wall function as

$$\tau_{wall} = \rho v_\tau \frac{V}{V^+},$$

such that the production of the turbulent kinetic energy

$$P_k = \tau_{ij} S_{ij} = 2 \mu_T S_{ij} S_{ij},$$

can be simplified using the wall-function relations

$$P_k = \tau_{wall} \frac{\partial v}{\partial y} = \frac{\tau_{wall}^2}{\mu} \frac{dv^+}{dy^+},$$

where the gradient of the law-of-the-wall is evaluated in the logarithmic region. Therefore, the final equation is

$$P_k = \frac{1}{\mu \kappa y^+} \left(\rho v_\tau \frac{V}{V^+} \right)^2.$$

The specific energy dissipation rate is defined as

$$\omega = \frac{\varepsilon}{\beta^* \kappa}.$$

Using the wall-function relations and equilibrium assumption, the value of ϵ can be found as

$$\rho \varepsilon = P_k = \tau_{wall} \frac{\partial v}{\partial y} = \rho^2 v_\tau^4 \frac{1}{\mu} \frac{dv^+}{dy^+}.$$

Thus, the value of ω is obtained by evaluating the gradient of the law-of-the-wall in the logarithmic region:

$$\omega = \frac{v_\tau}{\beta^{*1/2}\kappa y}.$$

The previous wall-function relationships are valid only in the logarithmic layer, where the near-wall resolution is $y^+ > 30$. However, since the k–ω model can be implemented throughout the inner layer, it is important to properly handle the fine resolution region $y^+ < 30$. In Abaqus/CFD a hybrid wall-function approach is employed in which the near-wall conditions adjust to their appropriate asymptotic limits for either very fine meshes ($y^+ \sim 1$) or for wall-function-type meshes ($y^+ \geq 30$). Appropriate blending functions are used such that for intermediate meshes (the near-wall element centroid is located in the buffer region) the accuracy is not impaired significantly. For completeness, we first state the appropriate relationships for the viscous sublayer:

$$v^+ = y^+,$$

$$\frac{V}{v^*} = \frac{yv^*}{\nu},$$

$$v^* = \sqrt{\frac{\nu V}{y}},$$

$$P_k = 0,$$

$$\omega = \frac{6\nu}{\beta_1 y^2}.$$

Having defined the viscous-sublayer and the fully turbulent relations, the blending is accomplished in the following form:

$$v_\tau = \sqrt{g\frac{\mu V}{\rho y} + (1-g)\beta^{*1/4}k^{1/2}},$$

$$P_k = (1-g)\frac{1}{\mu\kappa y^+}\tau_{wall}^2,$$

$$\omega = g\frac{6\nu}{\beta_1 y^2} + (1-g)\frac{v_\tau}{\sqrt{\beta^*}\kappa y} + g(1-g)\left(\frac{6\nu}{\beta_1 y^2} + \frac{v_\tau}{\sqrt{\beta^*}\kappa y}\right),$$

where the blending function is defined as

$$g = e^{-R_{ey}/y_c^+},$$

$$R_{ey} = \frac{\sqrt{k}y}{\nu}.$$

Here, R_{ey} is a local Reynolds number, and y_c^+ is the intersection point of the linear and logarithmic velocity profiles of the law-of-the-wall.

The hybrid wall-function approach is independent of the near-wall resolution; therefore, the law-of-the-wall $V^+(y^+)$ implemented needs to accurately predict the viscous-sublayer, the logarithmic-layer, and the buffer layer (region that connects the viscous and logarithmic zones) since the cell-center adjacent to the wall can be located anywhere within the inner layer. Therefore, a single smooth correlation that reproduces the entire law-of-the-wall proposed by Reichardt (1951) is implemented:

$$V^+(y^+) = \frac{1}{\kappa}\ln(1+\kappa y^+) + C\Big[1 - e^{-\frac{y^+}{y_c^+}} - \frac{y^+}{y_c^+}e^{-by^+}\Big],$$

where

$$C = \frac{1}{\kappa}\ln\left(\frac{E}{\kappa}\right),$$

$$b = \frac{1}{2}\left(\frac{y_c^+\kappa}{C} + \frac{1}{D}\right).$$

Implementation in the momentum equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-shear stress in coarse meshes. The wall shear is obtained from the wall-function approach through an effective edge viscosity:

$$\mu_{\text{eff}} = \frac{\tau_{wall}y}{V} = \left(\frac{\rho v_\tau V}{V^+}\right)\frac{y}{V}.$$

Energy wall functions

The wall-function approach can be extended to the energy equation by using the temperature law-of-the-wall which is a semi-empirical universal temperature profile obtained in equilibrium wall-bounded flows, when the temperature, $\boldsymbol{T}$, and the wall-normal distance, $\boldsymbol{y}$, are normalized with wall units. The standard temperature wall function is defined as

$$T^+(y^+) = \frac{(T_w - T)\rho c_p u_\tau}{\dot{q}} = \begin{cases} P_r y^+ & \text{if } y^+ \le y_{Tc}^+ \\ P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right) & \text{if } y^+ > y_{Tc}^+, \end{cases}$$

where y_{Tc}^+ is the intersection point of the viscous-sublayer and logarithmic layer in the temperature wall-function, T_w is the wall temperature, P_r is the Prandtl number, P_{rT} is the turbulent Prandtl number, $\dot{q}$

is the wall-heat flux, c_p is the specific heat coefficient at constant pressure, and P is computed using the Jayatilleke (1969) expression:

$$P(P_r, P_{rT}) = 9.24\left[\left(\frac{P_r}{P_{rT}}\right)^{3/4} - 1\right]\left(1 + 0.28e^{-0.007P_r/P_{rT}}\right).$$

For the hybrid wall-function approach a continuous temperature wall function proposed by Kader (1981) is implemented

$$T^+(y^+) = e^{-\Gamma}P_r y^+ + e^{-1/\Gamma}P_{rT}\left(\frac{1}{\kappa}\ln(Ey^+) + P(P_r, P_{rT})\right),$$

with the blending function defined as

$$\Gamma = \frac{0.01(P_r y^+)^4}{1 + 5P_r^3 y^+}.$$

Finally, the heat flux is obtained from the precomputed v_τ flow properties temperature field and the continuous temperature wall function:

$$\dot{q} = \frac{(T_w - T)\rho c_p v_\tau}{T^+}.$$

Implementation in the energy equation

For cases where the mesh resolution is not enough to capture the near-wall gradients, a near-wall model is required to provide the correct wall-heat flux in coarse meshes. The wall-heat flux is obtained from the wall-function approach through an effective edge heat conductivity:

$$\kappa_{\text{eff}} = \dot{q}\frac{y}{(T_w - T)}.$$

Deforming-mesh ALE (for transient analysis only)

Many industrial CFD/FSI/CHT problems involve moving boundaries or deforming geometries. This class of problem includes prescribed boundary motion that induces fluid flow or where the boundary motion is relatively independent of the fluid flow. Abaqus/CFD uses an arbitrary Lagrangian-Eulerian (ALE) formulation and automated mesh deformation method that preserves element size in boundary layers. The ALE and deforming-mesh algorithms are activated automatically for problems that involve a moving boundary prescribed by the user or identified as a moving boundary in an FSI co-simulation. Abaqus/CFD offers two approaches for mesh deformation: implicit and explicit. In both approaches the mesh motion is governed by linear elasticity equations. For the implicit approach the algorithm is similar to the static stress analysis procedure in Abaqus/Standard. To avoid extra memory allocation for solving the linear elasticity equations, the matrix-free iteration strategy is used. For the explicit approach the algorithm is similar to explicit dynamic analysis in Abaqus/Explicit. Abaqus/CFD also offers distortion

control in the explicit approach to prevent elements from inverting or distorting excessively in fluid mesh movement (see "Controlling the solution accuracy and mesh quality in a deforming-mesh analysis with Abaqus/CFD " in "Commonly used control parameters," Section 7.2.2).

To properly control the mesh motion during a simulation, it is the user's responsibility to prescribe appropriate displacement conditions on the boundary of the computational mesh.

Porous media flows (for transient analysis only)

Flows through fluid-saturated porous media occur in a wide range of industrial and environmental applications. Such flows can be isothermal (no heat transfer) or non-isothermal in nature. Examples include packed-bed heat exchangers, heat pipes, thermal insulation, petroleum reservoirs, nuclear waste repositories, geothermal engineering, thermal management of electronic devices, metal alloy casting, and flow past porous scaffolds in bioreactors.

Isothermal flows

For isothermal flows in porous media, many studies are usually carried out using the Darcy flow model, which is an empirical law for creeping flow through an infinitely extended uniform medium. However, non-Darcian effects such as fluid inertial effects are quite important for certain applications. The model implemented in Abaqus/CFD is based on the volume-averaged Darcy-Brinkman-Forchheimer equations that account for both Darcian and inertial non-Darcian effects. The following assumptions are made in deriving the governing equations:

- the porosity of the medium does not vary with time or the time scale of variation of the porosity is considered to be much larger than the dominant time scales of the fluid motion; and
- the permeability of the porous medium is isotropic and dependent only on the porosity of the medium.

Based on the above assumptions, the volume-averaged mass conservation and the Darcy-Brinkman-Forchheimer momentum equations governing the flow of an incompressible fluid in a fluid-saturated porous media can be written as follows (Nield and Bejan, 2010):

$$\nabla \cdot \mathbf{v} = 0,$$

$$\frac{\rho}{\epsilon}\left[\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla\left(\frac{\mathbf{v}}{\epsilon}\right)\right] = -\nabla p + \frac{\mu}{\epsilon}\nabla \cdot \nabla \mathbf{v} - \frac{\mu}{K}\mathbf{v} - \frac{\rho c_F}{K^{\frac{1}{2}}}|\mathbf{v}|\mathbf{v},$$

where

$\mathbf{v}$ is the extrinsic average or the superficial velocity vector, where the average is taken over a representative volume incorporating both the solid (matrix) and the fluid phases;

p is the intrinsic average of the pressure (average taken only over the fluid-phase);

ρ is the density of the fluid;

μ is the viscosity of the fluid;

ϵ is the porosity (volume fraction of the fluid phase) of the porous medium; and

K is the permeability of the porous medium.

The second term on the right-hand side of the momentum equation is the Brinkman term accounting for the presence of solid boundaries, the third term represents the Darcy drag term (linear in velocity), and the last term represents the inertial (quadratic in velocity) or the Forchheimer drag. The parameter c_F is the inertial drag coefficient (also referred to as the form drag coefficient). Based on Ergun's equation (Nield and Bejan, 2010), $c_F = \frac{C}{\sqrt{\epsilon^3}}$, where C is a constant that is set to a default value of $\frac{1.75}{\sqrt{150}} = 0.142887$. The porous drag forces (namely, the Darcy and Forchheimer drag forces) are activated for a prescribed element set by specifying them as distributed loads (see "Specifying porous drag body force load in Abaqus/CFD" in "Distributed loads," Section 34.4.3).

Thus, the porous media flow problem requires the specification of the porosity, ϵ, and the permeability, K, of the porous medium. The default value of C can also be changed in the material property definition (see "Permeability," Section 26.6.2). For the case of turbulent flow within a porous medium, the fluid viscosity μ includes the contribution of both the molecular and the turbulent eddy viscosities.

For conjugate flows involving domains consisting of both pure fluid regions and fluid-saturated porous media, the pure fluid porosity is set to a value of 1 by default.

Permeability-porosity relationships

The permeability of a porous medium is generally a function of the physical properties of the interconnected pore system such as porosity and tortuosity. Determination of the appropriate permeability-porosity relationship requires a detailed knowledge of the size distribution and spatial arrangement of the pore channels in the porous medium. The permeability-porosity relation can be specified directly in Abaqus/CFD using the material property definition.

Another permeability-porosity relation supported in Abaqus/CFD is the widely accepted Carman-Kozeny model. This relation is given as follows:

$$K = \frac{r_f^2}{4k_{kc}} \frac{\epsilon^3}{(1-\epsilon)^2},$$

where k_{kc} represents the Carman-Kozeny constant and r_f represents the average radius of the porous particles/fibers.

Limitations

- While turbulence can be activated for a porous media flow problem, a rigorous volume-averaging procedure has not been implemented in Abaqus/CFD to account for turbulence transport within the porous media. The equations governing the transport of the turbulence variables are solved by neglecting the effects of the presence of porous medium. In other words, the porous medium remains transparent (fully open) to the transport of turbulence variables.
- When the arbitrary Lagrangian-Eulerian (ALE) and deforming mesh algorithms are activated for a porous flow problem, changes in the porosity of the medium associated with large mesh/domain

deformations are not taken into account. The model is strictly valid only for the case of undeformable porous media.

Non-isothermal flows (heat transfer)

The following assumptions are made in the implementation of the volume-averaged energy equation for porous media in Abaqus/CFD:

- The medium is isotropic.
- Radiative effects, viscous dissipation, and work done by the changes in pressure are negligible.
- Local thermal equilibrium is valid (i.e., solid and fluid phase temperatures are the same).
- No net heat transfer takes place between the different phases in the porous media.

Based on the above assumptions, the effective energy equation for the porous medium can be given as follows (Nield and Bejan, 2010):

$$(\rho C_p)_{\text{eff}} \frac{\partial T}{\partial t} + (\rho C_p)_f \, \mathbf{v} \cdot \nabla T = \nabla \cdot (k_{\text{eff}} \nabla T) + {q_{\text{eff}}}''',$$

where

$$(\rho C_p)_{\text{eff}} = (1-\epsilon) \left(\rho_s C_{p_s}\right) + \epsilon \Big(\rho_f C_{p_f}\Big),$$

and

$$k_{\text{eff}} = (1-\epsilon)\, k_s + \epsilon k_f.$$

Here, $\mathbf{v}$ is the extrinsic average or the superficial velocity vector, and T is the temperature. The subscripts f, s, and eff denote the fluid phase, solid (matrix) phase, and effective medium, respectively. C_p is the specific heat capacity at constant pressure, k is the thermal conductivity, and ${q_{\text{eff}}}'''$ is the effective heat production per unit volume or the heat source ($= (1-\epsilon)\, {q_s}''' + \epsilon {q_f}'''$). For the case of turbulent heat transfer within a porous medium, the fluid conductivity k_f includes the contribution of both the molecular and turbulent eddy conductivities.

As seen from the above equation, the porous media heat transfer problem requires the specification of the following input:

- the thermal properties of the solid (matrix) phase: the density, ρ_s; the conductivity, k_s; and the specific heat capacity, C_{p_s}; and
- the thermal properties of the fluid (matrix) phase: the molecular conductivity, k_f; and specific heat capacity, C_{p_f}, apart from the specification of other fluid properties such as the density, ρ_f, viscosity, μ, and permeability, K.

Linear equation solvers

The solution methods for the momentum and auxiliary transport equations in Abaqus/CFD rely on scalable parallel preconditioned Krylov solvers. The pressure, pressure-increment, and distance function

equations are solved with user-selectable Krylov solvers and a robust algebraic multigrid preconditioner. A set of preselected default convergence criteria and iteration limits are prescribed for all linear equation solvers. The default solver settings should provide computationally efficient and robust solutions across a spectrum of CFD problems. Full access to diagnostic information, convergence criteria, and optional solvers is provided. In practice, the pressure-increment equation may be the most sensitive linear system and could require user intervention based on knowledge of the specific flow problem.

Input File Usage: Use the following option to specify parameters for solving the momentum transport equations:

∗MOMENTUM EQUATION SOLVER

Use the following option to specify parameters for solving the pressure equation:

∗PRESSURE EQUATION SOLVER

Use the following option to specify parameters for solving the energy transport equations:

∗ENERGY EQUATION SOLVER

Use the following option to specify parameters for solving other transport equations, such as the turbulence transport equations:

∗TRANSPORT EQUATION SOLVER

Convergence criteria and diagnostics

Iterative solvers compute an approximate solution to a given set of equations; therefore, convergence criteria are required to determine if the solution is acceptable. While default settings should be adequate for most problems, you can modify the convergence criteria. In addition, convergence history output is available that may be useful to help advanced users tune the solvers for performance or robustness. For the algebraic multigrid preconditioner, diagnostic information (such as the number of grids, grid sparsity, and largest eigenvalue and condition number estimates) is available upon request. The diagnostic information for the algebraic multigrid preconditioner is printed every time the preconditioner is computed.

Specifying convergence criteria

The linear convergence limit (also commonly referred to as the convergence tolerance), the frequency of convergence checking, and the maximum number of iterations can be set. The iterative solver will stop when the relative residual norm of the system of equations and the relative correction of the solution norm fall below the convergence limit.

Input File Usage: Use the following options to specify convergence criteria for the momentum and auxiliary transport equations:

∗MOMENTUM EQUATION SOLVER
max iterations, frequency check, convergence limit

*TRANSPORT EQUATION SOLVER
max iterations, frequency check, convergence limit
*PRESSURE EQUATION SOLVER
max iterations, frequency check, convergence limit

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Momentum Equation**, **Pressure Equation**, or **Transport Equation** tabbed page; enter values for **Iteration limit**, **Convergence checking frequency**, and **Linear convergence limit**

Accessing convergence output

You can monitor the convergence of the iterative solver by accessing convergence output. When you activate the convergence output, the current relative residual norm and the relative solution correction norm are output each time the convergence is checked.

Input File Usage: Use the following options to write convergence output to the log file for the linear equation solvers:

*MOMENTUM EQUATION SOLVER, CONVERGENCE=ON
*TRANSPORT EQUATION SOLVER, CONVERGENCE=ON
*PRESSURE EQUATION SOLVER, CONVERGENCE=ON

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Momentum Equation**, **Pressure Equation**, or **Transport Equation** tabbed page; toggle on **Include convergence output**

Accessing diagnostic information

Diagnostic output is useful only for the algebraic multigrid preconditioner. For other preconditioners, only a solver initialization message is printed for diagnostic output. For the algebraic multigrid preconditioner, the number of grids, grid sparsity, and largest eigenvalue and condition number estimates are output each time the preconditioner is computed.

Input File Usage: Use the following option to write diagnostic output to the log file for the pressure equation solver using the algebraic multigrid preconditioner:

*PRESSURE EQUATION SOLVER, TYPE=AMG, DIAGNOSTICS=ON

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page; toggle on **Include diagnostic output**

Specifying a solver for the pressure equation

Three solver types are available for the solving the pressure equation. The default AMG solver uses an algebraic multigrid preconditioner and offers the choice of three Krylov solvers: conjugate gradient, bi-conjugate gradient stabilized, and flexible generalized minimal residual. The SSORCG solver uses a symmetric successive over-relaxation preconditioner and conjugate gradient Krylov solver. The DSCG solver uses a diagonally scaled preconditioner and conjugate gradient Krylov solver. The AMG solver

provides many additional options that are intended for advanced usage and in cases where convergence difficulties are encountered.

Input File Usage: Use one of the following options to specify the solver type:

```
*PRESSURE EQUATION SOLVER, TYPE=AMG (default)
*PRESSURE EQUATION SOLVER, TYPE=SSORCG
*PRESSURE EQUATION SOLVER, TYPE=DSCG
```

Abaqus/CAE Usage: Use the following option to specify the AMG solver:

Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page: **Solver options**: **Use analysis defaults**

Use the following option to specify the SSORCG solver:

Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page: **Solver options**: **Specify**, **Preconditioner Type**: **Symmetric successive over-relaxation**

The DSCG solver is not supported in Abaqus/CAE.

Specifying the complexity level

For the AMG solver, you can choose from three preset levels or you can specify the Krylov solver and smoother settings directly. The presets are provided for convenience. Preset level 1 is primarily intended for use with meshes with good element aspect ratios and in some cases may provide a performance benefit over the default preset level 2. Preset level 3 is intended for problems that encounter convergence difficulties, which typically have elements with high aspect ratios or highly distorted elements.

Input File Usage: Preset level 1 corresponds to the following:

```
*PRESSURE EQUATION SOLVER, TYPE=AMG
250, 2, 10^-5
CHEBYCHEV, 2, 2, CG
V
```

Preset level 2 (default) corresponds to the following:

```
*PRESSURE EQUATION SOLVER, TYPE=AMG
250, 2, 10^-5
ICC, 1, 1, CG
V
```

Preset level 3 corresponds to the following:

```
*PRESSURE EQUATION SOLVER, TYPE=AMG
250, 2, 10^-5
ICC, 2, 2, BCGS
V
```

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page: **Solver options**: **Specify**, **Preconditioner Type**: **Algebraic multi-grid**

Use one of the following options to choose a preset complexity level:

Complexity Level: **Preset**: **1**, **2**, or **3**

Use the following option to specify the Krylov solver and smoother settings directly:

Complexity Level: **User defined**

Specifying the solver type

Three Krylov solver options are provided for the AMG solver. The default conjugate gradient solver is the fastest; however, in some cases where convergence difficulties are observed, the bi-conjugate gradient stabilized or flexible generalized minimal residual solvers are recommended. These two solvers are more robust but computationally more expensive than the conjugate gradient solver.

Input File Usage: Use the following option to specify the Krylov solver type:

∗PRESSURE EQUATION SOLVER, TYPE=AMG
first data line
, , , *solver type*

where *solver type* is CG for the conjugate gradient solver (default), BCGS for the bi-conjugate gradient squared solver, and FGMRES for the flexible generalized minimum residual solver.

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page: **Solver options**: **Specify**, **Preconditioner Type**: **Algebraic multi-grid**

Use one of the following options to specify the Krylov solver:

Solver Type: **Conjugate gradient**, **Bi-conjugate gradient, stabilized**, or **Flexible generalized minimal residual**

Specifying the residual smoother settings

You can choose between incomplete factorization and polynomial residual smoothers that are used within the AMG preconditioner. While incomplete factorization is computationally more expensive than polynomial smoothing, in many cases this cost is amortized by fast convergence and robustness. Polynomial smoothing is recommended for problems with a very good mesh quality (i.e., no skewed or large aspect ratio elements). The number of pre- and postsmoothing sweeps can also be specified. It is recommended that you apply the same number of pre- and postsweeps. For the polynomial smoother, a minimum of two pre- and postsweeps are recommended.

Input File Usage: Use the following option to specify the residual smoother settings:

∗PRESSURE EQUATION SOLVER, TYPE=AMG
first data line
smoother, *pre-smoothing sweeps*, *postsmoothing sweeps*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Flow**; **Solvers** tabbed page: **Pressure Equation** tabbed page: **Solver options**: **Specify**, **Preconditioner Type**: **Algebraic multi-grid**, **Residual Smoother**: **Incomplete factorization** or **Polynomial**, **Pre-sweeps**: select number, **Post-sweeps**: select number

Time incrementation

Abaqus/CFD uses second-order time-accurate integration by default, where all diffusive terms, advective terms, and body forces are integrated with the trapezoidal rule (Crank-Nicolson method). The default method is "second-order accurate" in that truncation errors within a time increment are proportional to the time increment squared, thus they decrease by a factor of four if the time increment is halved. You can individually select alternative time integrators for each of these terms. A fully implicit advection treatment is also available, which is particularly useful for quickly advancing toward steady-state solutions.

Time increment size control

By default, Abaqus/CFD uses an automatic time incrementation algorithm that continually adjusts the time increment size to satisfy the Courant-Friedrichs-Lewy (CFL) stability condition for advection. The default value, CFL=0.45, guarantees the solution's stability. You can further limit the automatically computed time increment size by specifying a maximum value. You can also specify an initial time increment size. This value is automatically decreased as necessary to satisfy a maximum initial CFL value of 0.45 based on the starting conditions of the flow.

Alternatively, you can select fixed time incrementation and specify the time increment size. In this case the time increment size remains constant throughout the step, but stability is not guaranteed.

Input File Usage: Use the following option to specify automatic time incrementation (default):

∗CFD, INCREMENTATION=FIXED CFL
time increment, time period, scale factor, suggested CFL, check increment, *max allowable time increment*
divergence tolerance, θ, , θ, θ

Use the following option to specify fixed time step incrementation:

∗CFD, INCREMENTATION=FIXED STEP SIZE
time increment, time period,
divergence tolerance, θ, , θ, θ

For both options above, θ can be set to 0.5 for the Crank-Nicolson method (default), 0.6667 for the Galerkin method, or 1 for the first-order backward-Euler method.

Abaqus/CAE Usage: Use the following options to specify automatic time incrementation:

Step module: **Create Step**: **General**: **Flow**; **Basic** tabbed page: enter a value for **Time period**; **Incrementation** tabbed page: **Type**: **Automatic (Fixed CFL)**; enter values for **Initial time increment**, **Maximum CFL number**, **Increment adjustment frequency**, **Time step growth scale factor**, **Divergence tolerance**

Use the following option to specify fixed time step incrementation:

Step module: **Create Step**: **General**: **Flow**; **Basic** tabbed page: enter a value for **Time period**; **Incrementation** tabbed page: **Type**: **Fixed**, enter values for **Time increment** and **Divergence tolerance**

Use the following options to specify the time integration method for viscous/diffusive terms, boundary conditions, and advective terms:

Viscous, **Load/Boundary condition**, or **Advective**: **Trapezoid (1/2)**, **Galerkin (2/3)**, or **Backward-Euler (1)**

Time-accurate analysis

The time integration parameters are all set by default to $\theta = 0.5$, which produces a second order–accurate semi-implicit method suitable for time-accurate transient analysis. When automatic time incrementation is used, you should specify CFL ≤ 2 to maintain stability and time accuracy.

Steady-state solution using the transient solver

In analyses where the goal is to reach a steady-state solution, the fully implicit (backward-Euler) method can be activated by setting all time integration parameters to $\theta = 1.0$. This method is unconditionally stable, allowing you to specify large CFL values to significantly increase the time increment size. Strict guidelines for selecting the maximum allowable CFL number are not available, and this maximum value may vary for different flows and meshes. CFL values of 10 or more have been used successfully for some analyses where only the final result is of interest.

Steady-state analysis using the steady-state solver

In Abaqus/CFD the steady-state solver is implemented using a second-order accurate SIMPLE-based algorithm. The nonlinear transport equations are solved sequentially for a specified number of iterations. It is up to the user to manually terminate the steady-state iterations.

Nonlinear convergence criteria

For the SIMPLE algorithm employed in Abaqus/CFD, the convergence behavior of the coupled nonlinear transport equations along with the pressure correction equation relies on the under-relaxation of the solution updates during successive iterations. Typically, this requires the specification of under-relaxation factors for the momentum, pressure correction, and other scalar transport equations such as temperature, turbulence, etc.

Specifying the number of iterations and under-relaxation factors

Abaqus/CFD solves the nonlinear transport equations sequentially for a specified number of iterations. The default is 10000.

Input File Usage: Use the following options to specify the number of nonlinear iterations:

∗CFD, INCOMPRESSIBLE NAVIER STOKES, STEADY STATE
number of nonlinear iterations

The under-relaxation factors are specified as the last data on the first data line of the corresponding linear equation solvers. Use the following options to specify the under-relaxation factors:

∗MOMENTUM EQUATION SOLVER
data for all linear convergence criteria, under-relaxation factor
∗TRANSPORT EQUATION SOLVER
data for all linear convergence criteria, under-relaxation factor
∗PRESSURE EQUATION SOLVER
data for all linear convergence criteria, under-relaxation factor
∗ENERGY EQUATION SOLVER
data for all linear convergence criteria, under-relaxation factor

Monitoring output variables

Abaqus/CFD provides a number of output variables that are useful for monitoring the health of a calculation and are good indicators for situations where the flow has reached a steady-state condition. These variables are written to the status (`.sta`) file and can be examined as the analysis job is executing. The RMS divergence output variable is useful for determining if a calculation is proceeding normally. Values of the RMS divergence output variable that are O(1) can indicate that the problem is incorrectly specified or that the calculation has become unstable. The global kinetic energy (KE) provides a good indicator for when the flow has reached a steady state; i.e., when the kinetic energy asymptotically approaches a constant value, the flow is typically achieving a steady-state condition where the velocities and pressure do not vary in time. Alternatively, the global kinetic energy can indicate a steady periodic or chaotic flow situation as well.

Initial conditions

Initial conditions for the density, velocity, temperature, turbulent eddy viscosity, turbulent kinetic energy, and dissipation rate can be specified (see "Initial conditions in Abaqus/CFD," Section 34.2.2). If the density is omitted, the specified material density is used for incompressible flow simulations.

For a well-posed incompressible flow problem, the initial velocity must satisfy the boundary conditions and the imposed divergence-free condition; i.e., the solvability conditions. Abaqus/CFD automatically uses the user-defined boundary conditions and tests the specified velocity initial conditions to be sure the solvability conditions are satisfied. If they are not, the initial velocity is projected onto a divergence-free subspace, yielding initial conditions that define a well-posed incompressible

Navier-Stokes problem. Therefore, in some circumstances, user-specified velocity initial conditions may be overridden with velocity conditions that satisfy solvability.

Boundary conditions

Boundary conditions for velocity, temperature, pressure, and eddy viscosity can be defined (see "Boundary conditions in Abaqus/CFD," Section 34.3.2). During the analysis prescribed boundary conditions can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2). All amplitude definitions except smooth step and solution-dependent amplitudes are available. By default, all boundary conditions are applied instantaneously. Velocity and pressure boundary conditions can be specified via user subroutines (see "SMACfdUserPressureBC," Section 1.3.1 of the Abaqus User Subroutines Reference Guide, and "SMACfdUserVelocityBC," Section 1.3.2 of the Abaqus User Subroutines Reference Guide).

Displacement and velocity boundary conditions at FSI interfaces are prescribed automatically by the definition of a co-simulation region; therefore, you should not prescribe these conditions at an FSI interface. Similarly, you should not define the temperature at a CHT interface; the temperature is prescribed automatically by the definition of a co-simulation region. For more information, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

The specification of no-slip/no-penetration boundary conditions at walls requires the specification of the turbulent eddy viscosity and normal-distance function, which is handled automatically by Abaqus/CAE.

Hydrostatic pressure condition

In incompressible flows, the pressure is known only within an arbitrary additive constant value or the hydrostatic pressure. In many practical situations, the pressure at an outflow boundary may be prescribed, which, in effect, sets the hydrostatic pressure level. In cases where there is no pressure prescribed, it is necessary to set the hydrostatic pressure level at a minimum of one node in the mesh.

The fluid reference pressure can be used to specify the hydrostatic pressure level. When there are no prescribed pressure boundary conditions, the fluid reference pressure establishes the hydrostatic pressure level and makes the pressure-increment equation non-singular. If pressure boundary conditions are prescribed in addition to the reference pressure level, the reference pressure simply adjusts the output pressures according to the specified pressure level. For more information, see "Specifying a fluid reference pressure" in "Concentrated loads," Section 34.4.2.

Loads

The loading types for Abaqus/CFD include applied heat flux, volumetric heat-generation sources, general body forces, and gravity loading. Gravity loading defines the gravity vector used with a Boussinesq-type body force in buoyancy driven flow (see "Specifying gravity loading" in "Distributed loads," Section 34.4.3). Gravity loading can be used only in conjunction with the energy equation and is ignored if used without the energy equation. During the analysis prescribed loads can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2). All amplitude definitions except smooth step and solution-dependent amplitudes are available.

Material options

Material definitions in Abaqus/CFD follow the Abaqus conventions but also present several material properties specific to fluid dynamics. In Abaqus/CFD the typical material properties include viscosity, constant-pressure specific heat, density, and coefficient of thermal expansion. The thermal expansion is used with a Boussinesq-type body force in buoyancy driven flow.

In contrast to Abaqus/Standard and Abaqus/Explicit, which use the constant-volume specific heat, the constant-pressure specific heat is required when the energy equation is used for thermal-flow problems. For problems involving an ideal gas, you can optionally specify constant-volume specific heat and the ideal gas constant.

Elements

Abaqus/CFD supports the following element types: the 8-node hexahedral element, FC3D8; the 6-node triangular prism element, FC3D6; the 5-node pyramid element, FC3D5; and the 4-node tetrahedral element, FC3D4 (see "Fluid (continuum) elements," Section 28.2.1).

Output

The output available from Abaqus/CFD for an incompressible fluid dynamic analysis includes both nodal and surface field data and element and surface time-history data. For the nodal and element output, the preselected field and history data include velocity (V), temperature (TEMP), pressure (PRESSURE), and turbulent eddy viscosity (TURBNU). In addition, preselected field data include displacement (U). Preselected data are not available for surface output.

In addition to the preselected output, you can request several derived and auxiliary variables. All of the output variable identifiers are outlined in "Abaqus/CFD output variable identifiers," Section 4.2.3.

Input file template

```
*HEADING
…
*NODE
…
*ELEMENT, TYPE=FC3D4
…
*MATERIAL, NAME=matname
*CONDUCTIVITY
Data lines to define the thermal conductivity
*DENSITY
Data lines to define the fluid density
*SPECIFIC HEAT, TYPE=CONSTANT PRESSURE
Data lines to define the specific heat
*VISCOSITY
```

Data lines to define the fluid viscosity
***INITIAL CONDITIONS, TYPE=TEMPERATURE, ELEMENT AVERAGE**
Data lines to prescribe initial temperatures at the elements
***INITIAL CONDITIONS, TYPE=VELX, ELEMENT AVERAGE**
Data lines to prescribe initial x-velocity at the elements
***INITIAL CONDITIONS, TYPE=VELY, ELEMENT AVERAGE**
Data lines to prescribe initial y-velocity at the elements
***INITIAL CONDITIONS, TYPE=VELY, ELEMENT AVERAGE**
Data lines to prescribe initial y-velocity at the elements
…
***AMPLITUDE, NAME=velxamp, DEFINITION=TABULAR**
Data lines to define amplitude curve to be used for inlet x-velocity
******
***STEP**
**** Incompressible flow example**
***CFD, INCOMPRESSIBLE NAVIER STOKES, INCREMENTATION=FIXED CFL**
Data lines to define incrementation
******
**** Boundary conditions**
******
***FLUID BOUNDARY, TYPE=SURFACE**
inlet_surface, **VELX,** *value for x-velocity*
inlet_surface, **VELY,** *value for y-velocity*
inlet_surface, **VELZ,** *value for z-velocity*
******
***FLUID BOUNDARY, TYPE=SURFACE**
temperature_surface, **TEMP,** *value for temperature*
******
***FLUID BOUNDARY, TYPE=SURFACE**
outlet_surface, **P,** *value for pressure*
******
**** Field output**
******
OUTPUT, FIELD, TIME INTERVAL=interval for field output*
***ELEMENT OUTPUT**
PRESSURE, TEMP, TURBNU, V
***NODE OUTPUT**
PRESSURE, TEMP, TURBNU, V
******
**** History output**
******
OUTPUT, HISTORY, FREQUENCY=interval for history output*

```
*ELEMENT OUTPUT, ELSET=element set for history output, FREQUENCY=SURFACE
…
*END STEP
```

Additional references

- Albets-Chico, X., C. D. Perez-Segarra, A. Olivia, and J. Bredberg, "Analysis of Wall-Function Approaches using Two-Equation Turbulence Models," International Journal of Heat and Mass Transfer, vol. 51, p. 4940–4957, 2008.
- Casey, M., and T. Wintergerste, *ERCOFTAC Special Interest Group on "Quality and Trust in Industrial CFD",* European Research Community on Flow, Turbulence and Combustion (ERCOFTAC), 2000.
- Chen, H. C., and V. C. Patel, "Near-Wall Turbulence Models for Complex Flows Including Separation," AIAA Journal, vol. 26, no. 6, p. 641–648, 1988.
- Craft, T. J., A. V. Gerasimov, H. Iacovides, and B. E. Launder, "Progress in the Generalization of Wall-Function Treatments," International Journal of Heat and Fluid Flow, vol. 23, p. 148–160, 2002.
- Durbin, P. A., "Limiters and wall treatments in applied turbulence modeling," Fluid Dynamics research, vol. 41, p. 1–17, 2009.
- Jayatilleke, C. L., "The Influence of Prandtl Number and Surface Roughness on the Resistance of the Laminar Sub-Layer to Momentum and Heat Transfer," Progress in Heat and Mass Transfer, vol. 1, p. 193–330, 1969.
- Jongen, T., "Simulation and Modeling of Turbulent Incompressible Flows," Ph.D. Thesis Lausanne EPFL, 1998.
- Kader, B., "Temperature and Concentration Profiles in Fully Turbulent Boundary Layers," International Journal of Heat and Mass Transfer, vol. 24, no. 9, p. 1541–1544, 1981.
- Launder, B. E., and D. B. Spalding, "The Numerical Computation of Turbulent Flows," Computer Methods in Applied Mechanics and Engineering, vol. 3, p. 269–289, 1974.
- Menter, F. R., "Influence of Freestream Values on k–ω Turbulence Model Predictions," AIAA Journal, vol. 30, no. 6, p. 1657–1659, 1992.
- Menter, F. R., "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Application," AIAA Journal, vol. 32, no. 8, p. 1598–1605, 1994.
- Nield, D.A., and A. Bejan, *Convection in Porous Media,* Springer, New York, Third edition, 2010.
- Reichardt, H., "Vollstandige Darstellung der turbulenten Geschwindigkeitsverteilung in glatten Leitungen," Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM), vol. 31, p. 208–219, 1951.
- Shih, T. H., W. W. Liou, A. Shabbir, Z. Yang, and J. Zhu, "A New k–ε Eddy Viscosity Model for High Reynolds Number Turbulent Flows," Computers and Fluids, vol. 24, no. 3, p. 227–238, 1995.

- Wilcox, D. C., "Reassessment of the Scale-Determining Equation for Advanced Turbulence Models," AIAA Journal, vol. 26, no. 11, p. 1299–1310, 1988.
- Yakhot, V., S. A. Orszag, S. Thangam, T. B. Gatski, and C. G. Speziale, "Development of Turbulence Models for Shear Flows by a Double Expansion Technique," Physics of Fluids A, vol. 4, no. 7, p. 1510–1520, 1992.

6.7 Electromagnetic analysis

- "Electromagnetic analysis procedures," Section 6.7.1
- "Piezoelectric analysis," Section 6.7.2
- "Coupled thermal-electrical analysis," Section 6.7.3
- "Fully coupled thermal-electrical-structural analysis," Section 6.7.4
- "Eddy current analysis," Section 6.7.5
- "Magnetostatic analysis," Section 6.7.6

6.7.1 ELECTROMAGNETIC ANALYSIS PROCEDURES

Overview

Abaqus/Standard offers several analysis procedures to model piezoelectric, electrical conduction, and electromagnetic phenomena. The distinct electrical phenomena modeled by these procedures is described first, followed by a brief overview of each procedure.

Electrostatic, electrical conduction, magnetostatic, and electromagnetic analyses

Piezoelectric effect is the electromechanical interaction exhibited by some materials. This coupled electrostatic-structural response is modeled using piezoelectric analysis in Abaqus/Standard. In this procedure the electric potential is a degree of freedom and its conjugate is the electric charge.

Coupled thermal-electrical conduction, with or without structural coupling, is modeled using electrical procedures. In these procedures the electric potential is a degree of freedom and its conjugate is the electric current. While transient effects are ignored in electrical conduction, thus making it steady state, thermal fields can be modeled either as transient or steady state.

Magnetostatic analysis is used to compute the magnetic fields due to direct currents. It solves the magnetostatic approximation to Maxwell's equations. The magnetic vector potential is a degree of freedom in a magnetostatic analysis, and its conjugate is the surface current.

Electromagnetic analysis is used to model the full coupling between time-varying electric and magnetic fields by solving Maxwell's equations. In such an analysis the magnetic vector potential is a degree of freedom and its conjugate is the surface current.

Electrostatic procedure

The following electrostatic analysis procedure is available in Abaqus/Standard:

- **Piezoelectric analysis:** In a piezoelectric material an electric potential gradient causes straining, while stress causes an electric potential in the material ("Piezoelectric analysis," Section 6.7.2). This coupling is provided by defining the piezoelectric and dielectric coefficients of a material and can be used in natural frequency extraction, transient dynamic analysis, both linear and nonlinear static stress analysis, and steady-state dynamic analysis procedures. In all procedures, including nonlinear statics and dynamics, the piezoelectric behavior is always assumed to be linear.

Steady electrical conduction procedures

The following electrical conduction analyses procedures are available in Abaqus/Standard:

- **Coupled thermal-electrical analysis:** The electric potential and temperature fields can be solved simultaneously by performing a coupled thermal-electrical analysis ("Coupled thermal-electrical analysis," Section 6.7.3). In these problems the energy dissipated by an electrical current flowing through a conductor is converted into thermal energy, and the electrical conductivity can, in turn, be temperature dependent. Thermal loads can be applied, but deformation of the structure is not considered. Coupled thermal-electrical problems can be linear or nonlinear.

- **Fully coupled thermal-electrical-structural analysis:** A coupled thermal-electrical-structural analysis is used to solve simultaneously for the stress/displacement, the electric potential, and the temperature fields. A coupled analysis is used when the thermal, electrical, and mechanical solutions affect each other strongly. An example of such a process is resistance spot welding, where two or more metal parts are joined by fusion at discrete points at the material interface. The fusion is caused by heat generated due to the current flow at the contact points, which depends on the pressure applied at these points.

  These problems can be transient or steady state and linear or nonlinear. Cavity radiation effects cannot be included in a fully coupled thermal-electrical-structural analysis. See "Fully coupled thermal-electrical-structural analysis," Section 6.7.4, for more details.

Magnetostatic procedure

The following magnetostatic analysis procedure is available in Abaqus/Standard:

- **Magnetostatic analysis:** A magnetostatic analysis is used to solve for the magnetic vector potential, from which the magnetic field is computed in the entire domain. For example, the magnetic field due to the flow of direct current can be modeled. The procedure supports linear as well as nonlinear magnetic material properties. See "Magnetostatic analysis," Section 6.7.6, for more details.

Electromagnetic procedures

Electromagnetic analyses are used to solve for the magnetic vector potential, from which both electric and magnetic fields are computed in the entire domain. The following electromagnetic analysis procedures are available in Abaqus/Standard:

- **Time-harmonic eddy current analysis:** This procedure assumes time-harmonic excitation and response. It supports linear electrical conductivity and linear magnetic material behavior. For example, eddy currents induced in a workpiece that is in the vicinity of a source of excitation (such as a coil carrying alternating current) can be modeled. See "Time-harmonic analysis" in "Eddy current analysis," Section 6.7.5, for more details.
- **Transient eddy current analysis:** This procedure assumes general time variation of the excitation and response. It supports linear electrical conductivity and both linear and nonlinear magnetic material behavior. For example, eddy currents induced in a workpiece that is in the vicinity of a source of excitation (such as a coil carrying time-varying current) can be modeled. See "Transient analysis" in "Eddy current analysis," Section 6.7.5, for more details.

6.7.2 PIEZOELECTRIC ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Piezoelectric behavior," Section 26.5.2
- "Defining an analysis," Section 6.1.2
- "Electromagnetic analysis procedures," Section 6.7.1
- "Defining a concentrated charge," Section 16.9.30 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a surface charge," Section 16.9.31 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a body charge," Section 16.9.32 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Coupled piezoelectric problems:

- are those in which an electric potential gradient causes straining, while stress causes an electric potential gradient in the material;
- are solved using an eigenfrequency extraction, modal dynamic, static, dynamic, or steady-state dynamic procedure;
- require the use of piezoelectric elements and piezoelectric material properties;
- can be performed for continuum problems in one, two, and three dimensions; and
- can be used in both linear and nonlinear analysis (however, in nonlinear analysis the piezoelectric part of the constitutive behavior is assumed to be linear).

Piezoelectric response

The electrical response of a piezoelectric material is assumed to be made up of piezoelectric and dielectric effects:

$$q_i = e^{\varphi}_{ijk}\varepsilon_{jk} + D^{\varphi(\varepsilon)}_{ij}E_j,$$

where

φ is the electrical potential,

q_i is the component of the electric flux vector (also known as the electric displacement) in the ith material direction,

e^{φ}_{ijk} is the piezoelectric stress coupling,

ε_{ij} is a small-strain component,

$D_{ij}^{\varphi(\varepsilon)}$ is the material's dielectric matrix for a fully constrained material, and

E_i is the negative of the gradient of the electrical potential along the *i*th material direction, $-\partial\varphi/\partial x_i$.

Defining piezoelectric and dielectric properties is discussed in "Piezoelectric behavior," Section 26.5.2. The theoretical basis of the piezoelectric analysis capability in Abaqus is defined in "Piezoelectric analysis," Section 2.10.1 of the Abaqus Theory Guide.

Procedures available for piezoelectric analysis

Piezoelectric analysis can be carried out with the following procedures:

- "Static stress analysis," Section 6.2.2
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Transient modal dynamic analysis," Section 6.3.7
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Subspace-based steady-state dynamic analysis," Section 6.3.9

Initial conditions

Initial conditions of piezoelectric quantities cannot be specified. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, for a description of the initial conditions that can be applied in static or dynamic procedures.

Boundary conditions

The electric potential at a node (degree of freedom 9) can be prescribed using a boundary condition (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Displacement and rotation degrees of freedom can also be prescribed by using boundary conditions as described in the relevant static and dynamic analysis procedure sections. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Boundary conditions can be prescribed as functions of time by referring to amplitude curves ("Amplitude curves," Section 34.1.2).

In an eigenfrequency extraction step ("Natural frequency extraction," Section 6.3.5) involving piezoelectric elements, the electric potential degree of freedom must be constrained at least at one node to remove singularities from the dielectric part of the element operator.

Loads

Both mechanical and electrical loads can be applied in a piezoelectric analysis.

Applying mechanical loads

The following types of mechanical loads can be prescribed in a piezoelectric analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3.

Applying electrical loads

The following types of electrical loads can be prescribed, as described in "Electromagnetic loads," Section 34.4.5:

- Concentrated electric charge.
- Distributed surface electric charge and body electric charge.

Loading in mode-based and subspace-based procedures

Electrical charge loads should be used only in conjunction with residual modes in the eigenvalue extraction step, due to the "massless" mode effect. Since the electrical potential degrees of freedom do not have any associated mass, these degrees of freedom are essentially eliminated (similar to Guyan reduction or mass condensation) during the eigenvalue extraction. The residual modes represent the static response corresponding to the electrical charge loads, which will adequately represent the potential degree of freedom in the eigenspace.

Predefined fields

The following predefined fields can be specified in a piezoelectric analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in piezoelectric elements, nodal temperatures can be specified. The specified temperature affects only temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material options

The piezoelectric coupling matrix and the dielectric matrix are specified as part of the material definition for piezoelectric materials, as described in "Piezoelectric behavior," Section 26.5.2. They are relevant only when the material definition is used with coupled piezoelectric elements.

The mechanical behavior of the material can include linear elasticity only ("Linear elastic behavior," Section 22.2.1).

Elements

Piezoelectric elements must be used in a piezoelectric analysis (see "Choosing the appropriate element for an analysis type," Section 27.1.3). The electric potential, φ, is degree of freedom 9 at each node of these elements. In addition, regular stress/displacement elements can be used in parts of the model where piezoelectric effects do not need to be considered.

Output

The following output variables are applicable to the electrical solution in a piezoelectric analysis:

Element integration point variables:

EENER	Electrostatic energy density.
EPG	Magnitude and components of the negative of the electrical potential gradient vector, $-\partial\varphi/\partial\mathbf{x}$.
EPGM	Magnitude of the electrical potential gradient vector.
EPG*n*	Component *n* of the negative of the electrical potential gradient vector (*n*=1, 2, 3).
EFLX	Magnitude and components of the electrical flux (displacement) vector, $\mathbf{q}$.
EFLXM	Magnitude of the electrical flux (displacement) vector.
EFLX*n*	Component *n* of the electrical flux (displacement) vector (*n*=1, 2, 3).

Whole element variables:

CHRGS	Values of distributed electrical charges.
ELCTE	Total electrostatic energy in the element, $\int_v \mathrm{EENER}\ dv$.

Nodal variables:

EPOT	Electrical potential degree of freedom at a node.
RCHG	Reactive electrical nodal charge (conjugate to prescribed electrical potential).
CECHG	Concentrated electrical nodal charge.

Limitations

Abaqus does not account for piezoelectric effects in the total energy balance equation, which can lead to an apparent imbalance of the total energy of the model in some situations. For example, if a piezoelectric truss is fixed at one end point and subjected to a potential difference between its two end points, it deforms due to the piezoelectric effect. Subsequently if the truss is held fixed in this deformed configuration and the potential difference removed, strain energy will be generated due to the constraints. This results in an equivalent increase in the total energy of the model.

Input file template

***HEADING**
…
MATERIAL, NAME=matl*
***ELASTIC**
Data lines to define linear elasticity
***PIEZOELECTRIC**
Data lines to define piezoelectric behavior
***DIELECTRIC**
Data lines to define dielectric behavior
…
AMPLITUDE, NAME=name*
Data lines to define amplitude curve for defining concentrated electric charge
******
***STEP, (optionally NLGEOM)**
***STATIC**
**** or *DYNAMIC, *FREQUENCY, *MODAL DYNAMIC,**
**** *STEADY STATE DYNAMICS (, DIRECT or , SUBSPACE PROJECTION)**
***BOUNDARY**
Data lines to define boundary conditions on electrical potential and displacement (rotation) degrees of freedom
CECHARGE, AMPLITUDE=name*
Data lines to define time-dependent concentrated electric charges
***DECHARGE and/or *DSECHARGE**
Data lines to define distributed electric charges
***CLOAD and/or *DLOAD and/or *DSLOAD**
Data lines to define mechanical loading
***END STEP**

6.7.3 COUPLED THERMAL-ELECTRICAL ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Electromagnetic analysis procedures," Section 6.7.1
- "Electrical conductivity," Section 26.5.1
- *COUPLED THERMAL-ELECTRICAL
- *JOULE HEAT FRACTION
- "Specifying a joule heat fraction," Section 12.10.4 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring a fully coupled, simultaneous heat transfer and electrical procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Coupled thermal-electrical problems:

- are those in which coupling between the electrical potential and temperature fields make it necessary to solve both fields simultaneously;
- require the use of coupled thermal-electrical elements, although pure heat transfer elements can also be used in the model;
- can include a specification of the fraction of electrical energy that will be released as heat;
- can include thermal interactions such as gap radiation, gap conductance, and heat generation between surfaces (see "Thermal contact properties," Section 37.2.1);
- can include cavity radiation effects (see "Cavity radiation," Section 41.1.1);
- can include electrical interactions such as electrical current flowing across surfaces (see "Electrical contact properties," Section 37.3.1);
- allow for transient or steady-state thermal solutions and for steady-state electrical solutions; and
- can be linear or nonlinear.

Coupled thermal-electrical analysis

Joule heating arises when the energy dissipated by an electrical current flowing through a conductor is converted into thermal energy. Abaqus/Standard provides a fully coupled thermal-electrical procedure for analyzing this type of problem: the coupled thermal-electrical equations are solved simultaneously for both temperature and electrical potential at the nodes.

The capability includes the analysis of the electrical problem, the thermal problem, and the coupling between the two problems. Coupling arises from two sources: temperature-dependent electrical conductivity and internal heat generation, which is a function of the electrical current density. The thermal part of the problem can include heat conduction and heat storage ("Thermal properties: overview," Section 26.2.1) as well as cavity radiation effects ("Cavity radiation," Section 41.1.1). Forced convection caused by fluid flowing through the mesh is not considered.

The thermal-electrical equations are unsymmetric; therefore, the unsymmetric solver is invoked automatically if you request coupled thermal-electrical analysis. For problems where coupling between the thermal and electrical solutions is weak or where a pure electrical conduction analysis is required for the entire model, the unsymmetric terms resulting from the interfield coupling may be small or zero. In these problems you can invoke the less costly symmetric storage and solution scheme by solving the thermal and electrical equations separately. The separated technique uses the symmetric solver by default. The thermal-electrical solution schemes are discussed below.

The theoretical basis of coupled thermal-electrical analysis is described in detail in "Coupled thermal-electrical analysis," Section 2.12.1 of the Abaqus Theory Guide.

Governing electric field equation

The electric field in a conducting material is governed by Maxwell's equation of conservation of charge. Assuming steady-state direct current, the equation reduces to

$$\int_S \mathbf{J} \cdot \mathbf{n}\, dS = \int_V r_c\, dV,$$

where V is any control volume whose surface is S, $\mathbf{n}$ is the outward normal to S, $\mathbf{J}$ is the electrical current density (current per unit area), and r_c is the internal volumetric current source per unit volume.

The flow of electrical current is described by Ohm's law:

$$\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E} = -\boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}},$$

where

$\mathbf{E}(\mathbf{x})$	is the electrical field intensity, defined as the negative of the gradient of the electrical potential $\mathbf{E} = -\partial\varphi/\partial\mathbf{x}$,
φ	is the electrical potential,
$\boldsymbol{\sigma}^E(\theta, f^\alpha)$	is the electrical conductivity matrix,
θ	is the temperature, and
$f^\alpha, \alpha = 1, 2...$	are predefined field variables.

Using Ohm's law in the conservation equation, written in variational form, provides the governing equation of the finite element model:

$$\int_V \frac{\partial \delta\varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}}\, dV = \int_S \delta\varphi\, J\, dS + \int_V \delta\varphi\, r_c\, dV,$$

where $J \stackrel{\text{def}}{=} -\mathbf{J} \cdot \mathbf{n}$ is the current density entering the control volume across $\mathcal{S}$.

Defining the electrical conductivity

The electrical conductivity, $\boldsymbol{\sigma}^E$, can be isotropic, orthotropic, or fully anisotropic (see "Electrical conductivity," Section 26.5.1). Ohm's law assumes that the electrical conductivity is independent of the electrical field, $\mathbf{E}$. The coupled thermal-electrical problem is nonlinear when the electrical conductivity depends on temperature.

Specifying the amount of thermal energy generated due to electrical current

Joule's law describes the rate of electrical energy, P_{ec}, dissipated by current flowing through a conductor as

$$P_{ec} = \mathbf{J} \cdot \mathbf{E} = \frac{\partial \varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^E \cdot \frac{\partial \varphi}{\partial \mathbf{x}}.$$

The amount of this energy released as internal heat within the body is $\eta_v \, P_{ec}$, where η_v is an energy conversion factor. You specify η_v in the material definition. It is assumed that all the electrical energy is converted into heat ($\eta_v = 1.0$) if you do not include the joule heat fraction in the material description. The fraction given can include a unit conversion factor, if required.

Input File Usage: *JOULE HEAT FRACTION

Abaqus/CAE Usage: Property module: material editor: **Thermal→Joule Heat Fraction**

Steady-state analysis

Steady-state analysis provides the steady-state solution directly. Steady-state thermal analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. Only direct current is considered in the electrical problem, and it is assumed that the system has negligible capacitance. (Electrical transient effects are so rapid that they can be neglected.)

Input File Usage: *COUPLED THERMAL-ELECTRICAL, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Steady state**

Assigning a "time" scale to the analysis

A steady-state analysis has no intrinsic physically meaningful time scale. Nevertheless, you can assign a "time" scale to the analysis step, which is often convenient for output identification and for specifying prescribed temperatures, electrical potential, and fluxes (heat flux and current density) with varying magnitudes. Thus, when steady-state analysis is chosen, you specify a "time" period and "time" incrementation parameters for the step; Abaqus/Standard then increments through the step accordingly.

Any fluxes or boundary condition changes to be applied during a steady-state step should be given using appropriate amplitude references to specify their "time" variations ("Amplitude curves," Section 34.1.2). If fluxes and boundary conditions are specified for the step without amplitude

references, they are assumed to change linearly with "time" during the step—from their magnitudes at the end of the previous step (or zero, if this is the beginning of the analysis) to their newly specified magnitudes at the end of this step (see "Defining an analysis," Section 6.1.2).

Transient analysis

Alternatively, the thermal portion of the coupled thermal-electrical problem can be considered transient. As in steady-state analysis, electrical transient effects are neglected. See "Uncoupled heat transfer analysis," Section 6.5.2, for a more detailed description of the heat transfer capability in Abaqus/Standard.

Input File Usage: *COUPLED THERMAL-ELECTRICAL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**

Time incrementation

Time integration in the transient heat transfer problem is done with the same backward Euler method used in uncoupled heat transfer analysis. This method is unconditionally stable for linear problems. You can specify the time increments directly, or Abaqus can select them automatically based on a user-prescribed maximum nodal temperature change in an increment. Automatic time incrementation is generally preferred.

Automatic incrementation

The time increment size can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that these values are not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see "Time integration accuracy in transient problems," Section 7.2.4).

Input File Usage: *COUPLED THERMAL-ELECTRICAL, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed incrementation

If you select fixed time incrementation and do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will then be used throughout the analysis.

Input File Usage: *COUPLED THERMAL-ELECTRICAL
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious oscillations due to small time increments

In transient heat transfer analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta \ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Ending a transient analysis

By default, a transient analysis will end when the specified time period has been completed. Alternatively, you can specify that the analysis should continue until steady-state conditions are reached. Steady state is defined by the temperature change rate; when the temperature changes at a rate that is less than the user-specified rate (given as part of the step definition), the analysis terminates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

*COUPLED THERMAL-ELECTRICAL, END=PERIOD (default)

Use the following option to end the analysis based on the temperature change rate:

*COUPLED THERMAL-ELECTRICAL, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Basic**: **Response: Transient**; **Incrementation**: **End step when temperature change is less than**

Fully coupled solution schemes

Abaqus/Standard offers an exact as well as an approximate implementation of Newton's method for coupled thermal-electrical analysis.

Exact implementation

An exact implementation of Newton's method involves a nonsymmetric Jacobian matrix as is illustrated in the following matrix representation of the coupled equations:

$$\begin{bmatrix} \boldsymbol{K}_{\varphi\varphi} & \boldsymbol{K}_{\varphi\theta} \\ \boldsymbol{K}_{\theta\varphi} & \boldsymbol{K}_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta\boldsymbol{\varphi} \\ \Delta\boldsymbol{\theta} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{R}_{\varphi} \\ \boldsymbol{R}_{\theta} \end{Bmatrix},$$

where $\Delta\boldsymbol{\varphi}$ and $\Delta\boldsymbol{\theta}$ are the respective corrections to the incremental electrical potential and temperature, $\boldsymbol{K_{ij}}$ are submatrices of the fully coupled Jacobian matrix, and $\boldsymbol{R}_{\varphi}$ and $\boldsymbol{R}_{\theta}$ are the electrical and thermal residual vectors, respectively.

Solving this system of equations requires the use of the unsymmetric matrix storage and solution scheme. Furthermore, the electrical and thermal equations must be solved simultaneously. The method provides quadratic convergence when the solution estimate is within the radius of convergence of the algorithm. The exact implementation is used by default.

Approximate implementation

Some problems require a fully coupled analysis in the sense that the electrical and thermal solutions evolve simultaneously, but with a weak coupling between the two solutions. In other words, the components in the off-diagonal submatrices $\boldsymbol{K}_{\varphi\theta}$, $\boldsymbol{K}_{\theta\varphi}$ are small compared to the components in the diagonal submatrices $\boldsymbol{K}_{\varphi\varphi}$, $\boldsymbol{K}_{\theta\theta}$. For these problems a less costly solution may be obtained by setting the off-diagonal submatrices to zero, so that we obtain an approximate set of equations:

$$\begin{bmatrix} \boldsymbol{K}_{\varphi\varphi} & 0 \\ 0 & \boldsymbol{K}_{\theta\theta} \end{bmatrix} \begin{Bmatrix} \Delta\boldsymbol{\varphi} \\ \Delta\boldsymbol{\theta} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{R}_{\varphi} \\ \boldsymbol{R}_{\theta} \end{Bmatrix}.$$

As a result of this approximation the electrical and thermal equations can be solved separately, with fewer equations to consider in each subproblem. The savings due to this approximation, measured as solver time per iteration, will be of the order of a factor of two, with similar significant savings in solver storage of the factored stiffness matrix. Further, in situations without strong thermal loading due to cavity radiation, the subproblems may be fully symmetric or approximated as symmetric, so that the less costly symmetric storage and solution scheme can be used. The solver time savings for a symmetric solution is an additional factor of two. Unless you explicitly select the unsymmetric solver for the step ("Defining an analysis," Section 6.1.2), the symmetric solver will be used with this separated technique.

This modified form of Newton's method does not affect solution accuracy since the fully coupled effect is considered through the residual vector $\boldsymbol{R_j}$ at each increment in time. However, the rate of convergence is no longer quadratic and depends strongly on the magnitude of the coupling effect, so more iterations are generally needed to achieve equilibrium than with the exact implementation of Newton's method. When the coupling is significant, the convergence rate becomes very slow and may prohibit the attainment of a solution. In such cases the exact implementation of Newton's method is required. In cases where it is possible to use this approximation, the convergence in an increment will depend strongly on the quality of the first guess to the incremental solution, which you can control by selecting the extrapolation method used for the step (see "Defining an analysis," Section 6.1.2).

Input File Usage: Use the following option to specify a separated solution scheme:

*SOLUTION TECHNIQUE, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electric**: **Other**: **Solution technique: Separated**

Uncoupled electric conduction and heat transfer analysis

The coupled thermal-electrical procedure can also be used to perform uncoupled electric conduction analysis for the whole model or just part of the model (using coupled thermal-electrical elements). Uncoupled electrical analysis is available by omitting the thermal properties from the material description, in which case only the electric potential degrees of freedom are activated in the element and all heat transfer effects are ignored. If heat transfer effects are ignored in the entire model, you should invoke the separated solution technique described above. Use of this technique will then invoke the symmetric storage and solution scheme, which is an exact representation of a purely electrical problem.

Similarly, coupled thermal-electrical elements can be used in an uncoupled heat transfer analysis ("Uncoupled heat transfer analysis," Section 6.5.2), in which case all electric conduction effects are ignored. This feature is useful if a thermal-electrical analysis is followed by a pure heat conduction analysis. A typical example is a welding process, where the electric current is applied instantaneously, followed by a cooldown period during which no electrical effects need to be considered. The symmetric solver is activated by default in an uncoupled heat transfer analysis.

Cavity radiation

Cavity radiation can be activated in a heat transfer step. This feature involves interacting heat transfer between all of the facets of the cavity surface, dependent on the facet temperatures, facet emissivities, and the geometric view factors between each facet pair. When the thermal emissivity is a function of temperature or field variables, you can specify the maximum allowable emissivity change during an increment in addition to the maximum temperature change to control the time incrementation. See "Cavity radiation," Section 41.1.1, for more information.

Input File Usage: Use the following option in the step definition to activate cavity radiation:

*RADIATION VIEW FACTOR

Use the following option to specify the maximum allowable emissivity change:

*HEAT TRANSFER, MXDEM=*max_delta_emissivity*

Abaqus/CAE Usage: You can specify the maximum allowable emissivity change for a heat transfer step.

Step module: **Create Step**: **General**: **Heat transfer**: **Incrementation**: **Max. allowable emissivity change per increment**

Initial conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures or field variables (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Since only steady-state electrical currents are considered, the initial value of the electrical potential is not relevant.

Boundary conditions

Boundary conditions can be used to prescribe the electrical potential, $\varphi = \varphi(\mathbf{x}, t)$ (degree of freedom 9), and the temperature, $\theta = \theta(\mathbf{x}, t)$ (degree of freedom 11), at the nodes. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Boundary conditions can be specified as functions of time by referring to amplitude curves (see "Amplitude curves," Section 34.1.2).

A boundary without any prescribed boundary conditions corresponds to an insulated surface.

Loads

Both thermal and electrical loads can be applied in a coupled thermal-electrical analysis.

Applying thermal loads

The following types of thermal loads can be prescribed in a coupled thermal-electrical analysis, as described in "Thermal loads," Section 34.4.4:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Average-temperature radiation conditions.
- Convective film conditions and radiation conditions.

Applying electrical loads

The following types of electrical loads can be prescribed, as described in "Electromagnetic loads," Section 34.4.5:

- Concentrated current.
- Distributed surface current densities and body current densities.

Predefined fields

Predefined temperature fields are not allowed in coupled thermal-electrical analyses. Boundary conditions should be used instead to specify temperatures, as described above.

Other predefined field variables can be specified in a coupled thermal-electrical analysis. These values affect only field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

Both thermal and electrical properties are active in coupled thermal-electrical analyses. If thermal properties are omitted, an uncoupled electrical analysis will be performed.

All mechanical behavior material models (such as elasticity and plasticity) are ignored in a coupled thermal-electrical analysis.

Thermal material properties

For the heat transfer portion of the analysis, the thermal conductivity must be defined (see "Conductivity," Section 26.2.2). The specific heat must also be defined for transient heat transfer problems (see "Specific heat," Section 26.2.3). If changes in internal energy due to phase changes are important, latent heat can be defined (see "Latent heat," Section 26.2.4). Thermal expansion coefficients ("Thermal expansion," Section 26.1.2) are not meaningful in a coupled thermal-electrical analysis since deformation of the structure is not considered. Internal heat generation can be specified (see "Uncoupled heat transfer analysis," Section 6.5.2).

Electrical material properties

For the electrical portion of the analysis, the electrical conductivity must be defined (see "Electrical conductivity," Section 26.5.1). The electrical conductivity can be a function of temperature and user-defined field variables. The fraction of electrical energy dissipated as heat can also be defined, as explained above.

Elements

The simultaneous solution in a coupled thermal-electrical analysis requires the use of elements that have both temperature (degree of freedom 11) and electrical potential (degree of freedom 9) as nodal variables. The finite element model can also include pure heat transfer elements (so that a pure heat transfer analysis is provided for that part of the model) and coupled thermal-electrical elements for which no thermal properties are given (so that a pure electrical conduction solution is provided for that part of the model).

Coupled thermal-electrical elements are available in Abaqus/Standard in one dimension, two dimensions (planar and axisymmetric), and three dimensions. See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

The following output variables can be used to request output relating to the electric conduction solution:

Element integration point variables:

EPG	Magnitude and components of the electrical potential gradient vector, $\partial\varphi/\partial\mathbf{x}$.
EPGM	Magnitude of the electrical potential gradient vector.
EPG*n*	Component *n* of the electrical potential gradient vector (*n*=1, 2, 3).
ECD	Magnitude and components of the electrical current density vector, ***J***.
JENER	Electrical energy dissipated due to flow of current, $P_{ec}t_{step}$.

Whole element variables:

ECURS	Distributed applied electrical current.
NCURS	Electrical current at nodes due to electric conduction.
ELJD	Total electrical energy dissipated due to flow of current, $\int_v P_{ec}t_{step}\,dv$.

Nodal variables:

EPOT	Electrical potential, φ.
RECUR	Reactive electrical current.
CECUR	Concentrated applied electrical current.

Whole model variables:

ALLJD	Electrical energy summed over the model.

Surface interaction variables (see "Electrical contact properties," Section 37.3.1):

ECD	Electrical current density.
ECDA	ECD multiplied by area.
ECDT	Time integrated ECD.
ECDTA	Time integrated ECDA.
SJD	Heat flux per unit area generated by the electrical current.
SJDA	SJD multiplied by area.
SJDT	Time integrated SJD.
SJDTA	Time integrated SJDA.
WEIGHT	Heat distribution between interface surfaces, f.

Considerations for steady-state coupled thermal-electrical analysis

In a steady-state coupled thermal-electrical analysis the electrical energy dissipated due to flow of electrical current at an integration point (output variable JENER) is computed using the following relationship:

$$E_{ec} = P_{ec} t_{step},$$

where E_{ec} denotes the electrical energy dissipated due to flow of electrical current and t_{step} is the current step time. In the above relationship it is assumed that the rate of the electrical energy dissipation, P_{ec}, has a constant value in the step that is equal to the value currently computed.

The output variable JENER and the derived output variables ELJD and ALLJD contain the values of electrical energies dissipated in the current step only. Similarly, the contribution from the electrical current flow to the output variable ALLWK includes only the external work performed in the current step.

Input file template

```
*HEADING
…
*MATERIAL, NAME=mat1
*CONDUCTIVITY
Data lines to define thermal conductivity
```

***ELECTRICAL CONDUCTIVITY**
Data lines to define electrical conductivity
***JOULE HEAT FRACTION**
Data lines to define the fraction of electric energy released as heat
******
***STEP**
***COUPLED THERMAL-ELECTRICAL**
Data line to define incrementation and steady state
***BOUNDARY**
Data lines to define boundary conditions on electrical potential and temperature degrees of freedom
***CECURRENT**
Data lines to define concentrated currents
***DECURRENT and/or *DSECURRENT**
Data lines to define distributed current densities
***CFLUX and/or *DFLUX and/or *DSFLUX**
Data lines to define thermal loading
***FILM and/or *SFILM and/or *RADIATE and/or *SRADIATE**
Data lines to define convective film and radiation conditions
…
***CONTACT PRINT or *CONTACT FILE**
Data lines to request output of surface interaction variables
***END STEP**

6.7.4 FULLY COUPLED THERMAL-ELECTRICAL-STRUCTURAL ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Fully coupled thermal-stress analysis," Section 6.5.3
- "Coupled thermal-electrical analysis," Section 6.7.3
- *COUPLED TEMPERATURE-DISPLACEMENT
- "Configuring a fully coupled, simultaneous heat transfer, electrical, and structural procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A fully coupled thermal-electrical-structural analysis:

- is performed when coupling between the displacement, temperature, and electrical potential fields makes it necessary to obtain solutions for all three fields simultaneously;
- requires the existence of elements with displacement, temperature, and electrical potential degrees of freedom in the model;
- allows for transient or steady-state thermal solutions, static displacement solutions, and steady-state electrical solutions;
- can include thermal interactions such as gap radiation, gap conductance, and gap heat generation between surfaces (see "Thermal contact properties," Section 37.2.1);
- can include electrical interactions such as gap electrical conductance (see "Electrical contact properties," Section 37.3.1);
- cannot include cavity radiation effects but may include radiation boundary conditions (see "Thermal loads," Section 34.4.4);
- takes into account temperature dependence of material properties only for the properties that are assigned to elements with temperature degrees of freedom;
- neglects inertia effects; and
- can be transient or steady state.

Fully coupled thermal-electrical-structural analysis

A fully coupled thermal-electrical-structural analysis is the union of a coupled thermal-displacement analysis (see "Fully coupled thermal-stress analysis," Section 6.5.3) and a coupled thermal-electrical analysis (see "Coupled thermal-electrical analysis," Section 6.7.3).

Coupling between the temperature and electrical degrees of freedom arises from temperature-dependent electrical conductivity and internal heat generation (Joule heating), which is a function of the electrical current density. The thermal part of the problem can include heat conduction and heat storage ("Thermal properties: overview," Section 26.2.1). Forced convection caused by fluid flowing through the mesh is not considered.

Coupling between the temperature and displacement degrees of freedom arises from temperature-dependent material properties, thermal expansion, and internal heat generation, which is a function of inelastic deformation of the material. In addition, contact conditions exist in some problems where the heat conducted between surfaces may depend strongly on the separation of the surfaces and/or the pressure transmitted across the surfaces as well as friction (see "Mechanical contact properties: overview," Section 37.1.1, and "Thermal contact properties," Section 37.2.1).

Coupling between the electrical and displacement degrees of freedom arises in problems where electricity flows between contact surfaces. The electrical conduction may depend strongly on the separation of the surfaces and/or the pressure transmitted across the surfaces (see "Electrical contact properties," Section 37.3.1).

An example of a simulation that requires a fully coupled thermal-electrical-structural analysis is resistance spot welding. In a typical spot welding process two or more thin metal sheets are pinched between two electrodes. A large current is passed between the electrodes, which melts the metal between the electrodes and forms a weld. The integrity of the weld depends on many parameters including the electrical conductance between the sheets (which can be a function of contact pressure and temperature).

Steady-state analysis

Steady-state analysis provides the steady-state solution directly. Steady-state thermal analysis means that the internal energy term (the specific heat term) in the governing heat transfer equation is omitted. A static displacement solution is assumed. Only direct current is considered in the electrical problem, and it is assumed that the system has negligible capacitance. Electrical transient effects are so rapid that they can be neglected.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Steady state**

Assigning a "time" scale to the analysis

In steady-state cases you should assign an arbitrary "time" scale to the step: you specify a "time" period and "time" incrementation parameters. This time scale is convenient for changing loads and boundary conditions through the step and for obtaining solutions to highly nonlinear (but steady-state) cases; however, for the latter purpose, transient analysis often provides a natural way of coping with the nonlinearity.

Accounting for frictional slip heat generation

Frictional slip heat generation is normally neglected in the steady-state case. However, it can still be accounted for if motions are used to specify translational or rotational nodal velocities in disk brake-type problems or if user subroutine **FRIC** provides the incremental frictional dissipation through the variable **SFD**. If frictional heat generation is present, the heat flux into the two contact surfaces depends on the slip rate of the surfaces. The "time" scale in this case cannot be described as arbitrary, and a transient analysis should be performed.

Transient analysis

Alternatively, you can perform a transient coupled thermal-electrical-structural analysis. As in steady-state analysis, electrical transient effects are neglected and a static displacement solution is assumed. You can control the time incrementation in a transient analysis directly, or Abaqus/Standard can control it automatically. Automatic time incrementation is generally preferred.

Automatic incrementation controlled by a maximum allowable temperature change

The time increments can be selected automatically based on a user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$. Abaqus/Standard will restrict the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis (see "Time integration accuracy in transient problems," Section 7.2.4).

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, DELTMX=$\Delta\theta_{max}$

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$

Fixed incrementation

If you do not specify $\Delta\theta_{max}$, fixed time increments equal to the user-specified initial time increment, Δt_0, will be used throughout the analysis.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**: **Increment size:** Δt_0

Spurious oscillations due to small time increments

In transient analysis with second-order elements there is a relationship between the minimum usable time increment and the element size. A simple guideline is

$$\Delta t > \frac{\rho c}{6k}\Delta\ell^2,$$

where Δt is the time increment, ρ is the density, c is the specific heat, k is the thermal conductivity, and $\Delta \ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used in a mesh of second-order elements, spurious oscillations can appear in the solution, in particular in the vicinity of boundaries with rapid temperature changes. These oscillations are nonphysical and may cause problems if temperature-dependent material properties are present. In transient analyses using first-order elements the heat capacity terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the temperature changes rapidly.

There is no upper limit on the time increment size (the integration procedure is unconditionally stable) unless nonlinearities cause convergence problems.

Automatic incrementation controlled by the creep response

The accuracy of the integration of time-dependent (creep) material behavior is governed by the user-specified accuracy tolerance parameter, $tolerance \geq (\dot{\bar{\varepsilon}}^{cr}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}_{t})\Delta t$. This parameter is used to prescribe the maximum strain rate change allowed at any point during an increment, as described in "Rate-dependent plasticity: creep and swelling," Section 23.2.4. The accuracy tolerance parameter can be specified together with the maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$ (described above); however, specifying the accuracy tolerance parameter activates automatic incrementation even if $\Delta\theta_{max}$ is not specified.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, DELTMX=$\Delta\theta_{max}$, CETOL=*tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle on **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Type: Automatic**: **Max. allowable temperature change per increment:** $\Delta\theta_{max}$, **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*

Selecting explicit creep integration

Nonlinear creep problems ("Rate-dependent plasticity: creep and swelling," Section 23.2.4) that exhibit no other nonlinearities can be solved efficiently by forward-difference integration of the inelastic strains if the inelastic strain increments are smaller than the elastic strains. This explicit method is efficient computationally because, unlike implicit methods, iteration is not required as long as no other nonlinearities are present. Although this method is only conditionally stable, the numerical stability limit of the explicit operator is in many cases sufficiently large to allow the solution to be developed in a reasonable number of time increments.

For most coupled thermal-electrical-structural analyses, however, the unconditional stability of the backward difference operator (implicit method) is desirable. In such cases the implicit integration scheme may be invoked automatically by Abaqus/Standard.

Explicit integration can be less expensive computationally and simplifies implementation of user-defined creep laws in user subroutine **CREEP**; you can restrict Abaqus/Standard to using this method

for creep problems (with or without geometric nonlinearity included). See "Rate-dependent plasticity: creep and swelling," Section 23.2.4, for further details.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, CETOL=*tolerance*, CREEP=EXPLICIT

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle on **Include creep/swelling/viscoelastic behavior**; **Incrementation**: **Creep/swelling/viscoelastic strain error tolerance:** *tolerance*, **Creep/swelling/viscoelastic integration: Explicit**

Excluding creep and viscoelastic response

You can specify that no creep or viscoelastic response will occur during a step even if creep or viscoelastic material properties have been defined.

Input File Usage: *COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL, DELTMX=$\Delta\theta_{max}$, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled thermal-electrical-structural**: **Basic**: **Response: Transient**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers a method of stabilizing this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Units

In coupled problems where two or three different fields are active, take care when choosing the units of the problem. If the choice of units is such that the terms generated by the equations for each field are different by many orders of magnitude, the precision on some computers may be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid ill-conditioned matrices. For example, consider using units of Mpascal instead of pascal for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations, the heat flux continuity equations, and the conservation of charge equations.

Initial conditions

By default, the initial temperature of all nodes is zero. You can specify nonzero initial temperatures. Initial stresses, field variables, etc. can also be defined; "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the initial conditions that are available for a fully coupled thermal-electrical-structural analysis.

Boundary conditions

Boundary conditions can be used to prescribe temperatures (degree of freedom 11), displacements/rotations (degrees of freedom 1–6), or electrical potentials (degree of freedom 9) at nodes in a fully coupled thermal-electrical-structural analysis (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Boundary conditions can be specified as functions of time by referring to amplitude curves ("Amplitude curves," Section 34.1.2).

Loads

The following types of thermal loads can be prescribed in a fully coupled thermal-electrical-structural analysis, as described in "Thermal loads," Section 34.4.4:

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Node-based film and radiation conditions.
- Average-temperature radiation conditions.
- Element and surface-based film and radiation conditions.

The following types of mechanical loads can be prescribed:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3.

The following types of electrical loads can be prescribed, as described in "Electromagnetic loads," Section 34.4.5:

- Concentrated current.
- Distributed surface current densities and body current densities.

Predefined fields

Predefined temperature fields are not allowed in a fully coupled thermal-electrical-structural analysis. Boundary conditions should be used instead to prescribe temperature degree of freedom 11, as described earlier.

Other predefined field variables can be specified in a fully coupled thermal-electrical-structural analysis. These values will affect only field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

The materials in a fully coupled thermal-electrical-structural analysis must have thermal properties (such as conductivity), mechanical properties (such as elasticity), and electrical properties (such as electrical conductivity) defined. See Part V, "Materials," for details on the material models available in Abaqus.

Internal heat generation can be specified; see "Uncoupled heat transfer analysis," Section 6.5.2.

Thermal strain will arise if thermal expansion ("Thermal expansion," Section 26.1.2) is included in the material property definition.

A fully coupled thermal-electrical-structural analysis can be used to analyze static creep and swelling problems, which generally occur over fairly long time periods ("Rate-dependent plasticity: creep and swelling," Section 23.2.4); viscoelastic materials ("Time domain viscoelasticity," Section 22.7.1); or viscoplastic materials ("Rate-dependent yield," Section 23.2.3).

Inelastic energy dissipation as a heat source

You can specify an inelastic heat fraction in a fully coupled thermal-electrical-structural analysis to provide for inelastic energy dissipation as a heat source. The heat flux per unit volume, r^{pl}, that is added into the thermal energy balance is computed using the equation

$$r^{pl} = \eta \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}^{pl},$$

or, in the case when the nonlinear isotropic/kinematic hardening model is used, from the following equation:

$$r^{pl} = \eta(\boldsymbol{\sigma} - \boldsymbol{\alpha}) : \dot{\boldsymbol{\varepsilon}}^{pl},$$

where η is a user-defined factor (assumed constant), $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\alpha}$ is the backstress, and $\dot{\boldsymbol{\varepsilon}}^{pl}$ is the rate of plastic straining.

Inelastic heat fractions are typically used in the simulation of high-speed manufacturing processes involving large amounts of inelastic strain, where the heating of the material caused by its deformation significantly influences temperature-dependent material properties. The generated heat is treated as a volumetric heat flux source term in the heat balance equation.

An inelastic heat fraction can be specified for materials with plastic behavior that use the Mises or Hill yield surface ("Inelastic behavior," Section 23.1.1). It cannot be used with the combined isotropic/kinematic hardening model. The inelastic heat fraction can be specified for user-defined material behavior in Abaqus/Explicit and will be multiplied by the inelastic energy dissipation coded in the user subroutine to obtain the heat flux. In Abaqus/Standard the inelastic heat fraction cannot be used with user-defined material behavior; in this case the heat flux that must be added to the thermal energy balance is computed directly in the user subroutine.

In Abaqus/Standard an inelastic heat fraction can also be specified for hyperelastic material definitions that include time-domain viscoelasticity ("Time domain viscoelasticity," Section 22.7.1).

The default value of the inelastic heat fraction is 0.9. If you do not include the inelastic heat fraction behavior in the material definition, the heat generated by inelastic deformation is not included in the analysis.

Input File Usage: *INELASTIC HEAT FRACTION
η

Abaqus/CAE Usage: Property module: material editor: **Thermal**: **Inelastic Heat Fraction**: **Fraction:** η

Specifying the amount of thermal energy generated due to electrical current

Joule's law describes the rate of electrical energy, P_{ec}, dissipated by current flowing through a conductor as

$$P_{ec} = \mathbf{J} \cdot \mathbf{E} = \frac{\partial \varphi}{\partial \mathbf{x}} \cdot \boldsymbol{\sigma}^{E} \cdot \frac{\partial \varphi}{\partial \mathbf{x}}.$$

The amount of this energy released as internal heat within the body is $\eta_v \, P_{ec}$, where η_v is an energy conversion factor. You specify η_v in the material definition. It is assumed that all the electrical energy is converted into heat ($\eta_v = 1.0$) if you do not include the joule heat fraction in the material description. The fraction given can include a unit conversion factor, if required.

Input File Usage: *JOULE HEAT FRACTION

Abaqus/CAE Usage: Property module: material editor: **Thermal→Joule Heat Fraction**

Elements

Coupled thermal-electrical-structural elements that have displacements, temperatures, and electrical potentials as nodal variables are available. Simultaneous temperature/electrical potential/displacement solution requires the use of such elements; pure displacement and temperature-displacement elements can be used in part of the model in a fully coupled thermal-electrical-structural analysis, but pure heat transfer elements cannot be used.

The first-order coupled thermal-electrical-structural elements in Abaqus use a constant temperature over the element to calculate thermal expansion. The second-order coupled thermal-electrical-structural elements in Abaqus use a lower-order interpolation for temperature than for displacement (parabolic variation of displacements and linear variation of temperature) to obtain a compatible variation of thermal and mechanical strain.

Output

See "Abaqus/Standard output variable identifiers," Section 4.2.1, for a complete list of output variables. The types of output available are described in "Output," Section 4.1.1.

Considerations for steady-state coupled thermal-electrical-structural analysis

In a steady-state coupled thermal-electrical-structural analysis the electrical energy dissipated due to flow of electrical current at an integration point (output variable JENER) is computed using the following relationship:

$$E_{ec} = P_{ec} t_{step},$$

where E_{ec} denotes the electrical energy dissipated due to flow of electrical current and t_{step} is the current step time. In the above relationship it is assumed that the rate of the electrical energy dissipation, P_{ec}, has a constant value in the step that is equal to the value currently computed.

The output variable JENER and the derived output variables ELJD and ALLJD contain the values of electrical energies dissipated in the current step only. Similarly, the contribution from the electrical current flow to the output variable ALLWK includes only the external work performed in the current step.

Input file template

```
*HEADING
…
** Specify the coupled thermal-electrical-structural element type
*ELEMENT, TYPE=Q3D8
…
**
*STEP
*COUPLED TEMPERATURE-DISPLACEMENT, ELECTRICAL
Data line to define incrementation
*BOUNDARY
Data lines to define nonzero boundary conditions on displacement,
temperature or electrical potential degrees of freedom
*CFLUX and/or *CFILM and/or
*CRADIATE and/or *DFLUX and/or
*DSFLUX and/or *FILM and/or
*SFILM and/or *RADIATE and/or
*SRADIATE
Data lines to define thermal loads
*CLOAD and/or *DLOAD and/or *DSLOAD
Data lines to define mechanical loads
*CECURRENT
Data lines to define concentrated currents
*DECURRENT and/or *DSECURRENT
Data lines to define distributed current densities
*FIELD
Data lines to define field variable values
*END STEP
```

6.7.5 EDDY CURRENT ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Mapping thermal and magnetic loads," Section 3.2.27
- "Electromagnetic analysis procedures," Section 6.7.1
- "Electrical conductivity," Section 26.5.1
- "Magnetic permeability," Section 26.5.3
- "Electromagnetic loads," Section 34.4.5
- "Predefined loads for sequential coupling," Section 16.1.3
- *ELECTROMAGNETIC
- *D EM POTENTIAL
- *DECURRENT
- *DSECURRENT
- *MOTION
- "UDECURRENT," Section 1.1.24 of the Abaqus User Subroutines Reference Guide
- "UDEMPOTENTIAL," Section 1.1.25 of the Abaqus User Subroutines Reference Guide
- "UDSECURRENT," Section 1.1.27 of the Abaqus User Subroutines Reference Guide
- "Configuring a time-harmonic electromagnetic analysis" in "Configuring linear perturbation analysis procedures," Section 14.11.2 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a magnetic vector potential boundary condition," Section 16.10.17 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Eddy current problems:

- involve coupling between electric and magnetic fields, which are solved for simultaneously;
- solve Maxwell's equations describing electromagnetic phenomena under the low-frequency assumption that neglects the effects of displacement currents;
- require the use of electromagnetic elements in the whole domain;
- require that magnetic permeability is specified in the whole domain and electrical conductivity is specified in the conducting regions;
- allow for both time-harmonic and transient electromagnetic solutions;
- allow predefined conductor translation and rotation;

- calculate as output variables, rate of Joule heating and intensity of magnetic body forces associated with eddy currents, and these output variables can be transferred from a time-harmonic electromagnetic solution to drive a subsequent heat transfer, coupled temperature-displacement, or stress/displacement analysis, thereby allowing for the coupling of electromagnetic fields with thermal and/or mechanical fields in a sequentially coupled manner; and
- can be solved using continuum elements in two- and three-dimensional space.

Eddy current analysis

Eddy currents are generated in a metal workpiece when it is placed within a time-varying magnetic field. Joule heating arises when the energy dissipated by the eddy currents flowing through the workpiece is converted into thermal energy. This heating mechanism is usually referred to as induction heating; the induction cooker is an example of a device that uses this mechanism. The time-varying magnetic field is usually generated by a coil that is placed close to the workpiece. The coil carries either a known amount of total current or an unknown amount of current under a known potential (voltage) difference. The current in the coil is assumed to be alternating at a known frequency for a time-harmonic eddy current analysis but may have an arbitrary variation in time for a transient eddy current analysis.

The time-harmonic eddy current analysis procedure is based on the assumption that a time-harmonic excitation with a certain frequency results in a time-harmonic electromagnetic response with the same frequency everywhere in the domain. In other words, both the electric and the magnetic fields oscillate at the same frequency as that of the alternating current in the coil. The transient eddy current analysis does not make any assumption regarding the time-variation of the current in the coil; in fact any arbitrary time variation can be specified, and the electric and magnetic fields follow from the solution to Maxwell's equations in the time domain.

The eddy current analysis provides output, such as Joule heat dissipation or magnetic body force intensity, that can be transferred, from a time-harmonic eddy current analysis only, to drive a subsequent heat transfer, coupled temperature-displacement, or stress/displacement analysis. This allows for modeling the interactions of the electromagnetic fields with thermal and/or mechanical fields in a sequentially coupled manner. See "Mapping thermal and magnetic loads," Section 3.2.27, and "Predefined loads for sequential coupling," Section 16.1.3, for details.

Electromagnetic elements must be used to model the response of all the regions in an eddy current analysis including the coil, the workpiece, and the space in between and surrounding them. To obtain accurate solutions, the outer boundary of the space (surrounding the coil and the workpiece) being modeled must be at least a few characteristic length scales away from the device on all sides.

The electromagnetic elements use an element edge-based interpolation of the fields instead of the standard node-based interpolation. The user-defined nodes only define the geometry of the elements; and the degrees of freedom of the element are not associated with these nodes, which has implications for applying boundary conditions (see "Boundary conditions" below).

Governing field equations

The electric and magnetic fields are governed by Maxwell's equations describing electromagnetic phenomena. The formulation is based on the low-frequency assumption, which neglects the

displacement current correction term in Ampere's law. This assumption is appropriate when the wavelength of the electromagnetic waves corresponding to the excitation frequency is large compared to typical length scales over which the response is computed. In the following discussion, the governing equations are written for a linear medium.

Time-harmonic analysis

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$. The solution procedure seeks a time-harmonic electromagnetic response, $\mathbf{A}_0 \exp(i\omega t)$, with frequency ω radians/sec when the system is subjected to a time-harmonic excitation of the same frequency; for example, through an impressed oscillating volume current density, $\mathbf{J}_0 \exp(i\omega t)$. In the preceding expressions the vectors $\mathbf{A}_0$ and $\mathbf{J}_0$ represent the amplitudes of the magnetic vector potential and applied volume current density vector, respectively, while the exponential factors (with $i = \sqrt{-1}$) represent the corresponding phases. Under these assumptions, Maxwell's equations in the absence of conductor motion reduce to

$$\nabla \times (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}_0) + i\omega \boldsymbol{\sigma}^E \cdot \mathbf{A}_0 = \mathbf{J}_0$$

in terms of the amplitudes of the field quantities, $\mathbf{A}_0$ and $\mathbf{J}_0$; the magnetic permeability tensor, $\boldsymbol{\mu}$; and the electrical conductivity tensor, $\boldsymbol{\sigma}^E$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$, while the electrical conductivity relates the volume current density, $\mathbf{J}$, and the electric field, $\mathbf{E}$, by Ohm's law: $\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A}_0 \cdot (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}_0)\, dV + i\,\omega \int_V \delta\mathbf{A}_0 \cdot \boldsymbol{\sigma}^E \cdot \mathbf{A}_0\, dV = \int_V \delta\mathbf{A}_0 \cdot \mathbf{J}_0\, dV + \int_S \delta\mathbf{A}_0 \cdot \mathbf{K}_0\, dS,$$

where $\delta\mathbf{A}_0$ represents the variation of the magnetic vector potential, and $\mathbf{K}_0$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the in-phase (real) and out-of-phase (imaginary) components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential.

Transient analysis

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, assumed to be a function of spatial position and time, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$. The solution procedure seeks a time-dependent electromagnetic response, $\mathbf{A}(\mathbf{x}, t)$, when the system is subjected to a time-dependent excitation; for example, through an impressed distribution of volume current density, $\mathbf{J}(\mathbf{x}, t)$. Under these assumptions, Maxwell's equations in the absence of conductor motion reduce to

$$\nabla \times (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}) + \boldsymbol{\sigma}^E \cdot \partial\mathbf{A}/\partial t = \mathbf{J}$$

in terms of the field quantities, $\mathbf{A}$ and $\mathbf{J}$; the magnetic permeability tensor, $\boldsymbol{\mu}$; and the electrical conductivity tensor, $\boldsymbol{\sigma}^E$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$, while the electrical conductivity relates the volume current density, $\mathbf{J}$, and the electric field, $\mathbf{E}$, by Ohm's law: $\mathbf{J} = \boldsymbol{\sigma}^E \cdot \mathbf{E}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A} \cdot (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A})\, dV + \int_V \delta\mathbf{A} \cdot \boldsymbol{\sigma}^E \cdot \partial\mathbf{A}/\partial t\, dV = \int_V \delta\mathbf{A} \cdot \mathbf{J}\, dV + \int_S \delta\mathbf{A} \cdot \mathbf{K}\, dS,$$

where $\delta\mathbf{A}$ represents the variation of the magnetic vector potential, and $\mathbf{K}$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential.

Predefined conductor motion

Electric fields induced in a conductor have two parts: the first part due to the changing magnetic flux (Faraday's law of induction), which is already accounted for in the formulation above, and the second part due to the motion of the conductor in the magnetic field. The second part modifies the governing equation as follows (shown only for a transient procedure, although the capability is available for the time-harmonic procedure as well):

$$\nabla \times (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}) + \boldsymbol{\sigma}^E \cdot \partial\mathbf{A}/\partial t - \boldsymbol{\sigma}^E \cdot \mathbf{v} \times (\nabla \times \mathbf{A}) = \mathbf{J}$$

in terms of the prescribed velocity of motion $\mathbf{v}$. You can prescribe both translational and rotational velocity of motion. The formulation assumes that the conductor is uniform in the direction of motion; in other words, it does not have any geometric features in the direction of motion.

Conductor motion results in unsymmetric contributions toward the element operator. By default, the unsymmetric storage and solution scheme is used with conductor motion.

Defining the magnetic behavior

The magnetic behavior of the electromagnetic medium can be linear or nonlinear. However, only linear magnetic behavior is available for time-harmonic eddy current analysis. Linear magnetic behavior is characterized by a magnetic permeability tensor that is assumed to be independent of the magnetic field. It is defined through direct specification of the absolute magnetic permeability tensor, $\boldsymbol{\mu}$, which can be isotropic, orthotropic, or fully anisotropic (see "Magnetic permeability," Section 26.5.3). The magnetic permeability can also depend on temperature and/or predefined field variables. For a time-harmonic eddy current analysis, the magnetic permeability can also depend on frequency.

Nonlinear magnetic behavior, which is available only for transient eddy current analysis, is characterized by magnetic permeability that depends on the strength of the magnetic field. The nonlinear magnetic material model in Abaqus is suitable for ideally soft magnetic materials characterized by a monotonically increasing response in B–H space, where B and H refer to the strengths of the

magnetic flux density vector and the magnetic field vector, respectively. Nonlinear magnetic behavior is defined through direct specification of one or more B–H curves that provide B as a function of H and, optionally, temperature and/or predefined field variables, in one or more directions. Nonlinear magnetic behavior can be isotropic, orthotropic, or transversely isotropic (which is a special case of the more general orthotropic behavior).

Defining the electrical conductivity

The electrical conductivity, σ^E, can be isotropic, orthotropic, or fully anisotropic (see "Electrical conductivity," Section 26.5.1). The electrical conductivity can also depend on temperature and/or predefined field variables. For a time-harmonic eddy current analysis, the electrical conductivity can also depend on frequency. Ohm's law assumes that the electrical conductivity is independent of the electrical field, $\mathbf{E}$.

Time-harmonic analysis

The eddy current analysis procedure provides the time-harmonic solution directly at a given excitation frequency. You can specify one or more excitation frequencies, one or more frequency ranges, or a combination of excitation frequencies and ranges.

Input File Usage: *ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
lower_freq1, upper_freq1, num_pts1
lower_freq2, upper_freq2, num_pts2
...
single_freq1
single_freq2
...

For example, the following input illustrates the simplest case of specifying excitation at a single frequency:

*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
single_freq1

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Electromagnetic, Time harmonic**; enter data in table, and add rows as necessary

Transient analysis

The eddy current analysis procedure provides the transient solution to a given arbitrary time-dependent excitation.

Input File Usage: *ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Time incrementation

Time integration in the transient eddy current analysis is done with the backward Euler method. This method is unconditionally stable for linear problems but may lead to inaccuracies if time increments are too large. The resulting system of equations can be nonlinear in general, and Abaqus/Standard uses Newton's method to solve the system. The solution usually is obtained as a series of increments, with iterations to obtain equilibrium within each increment. Increments must sometimes be kept small to ensure accuracy of the time integration procedure. The choice of increment size is also a matter of computational efficiency: if the increments are too large, more iterations are required. Furthermore, Newton's method has a finite radius of convergence; too large an increment can prevent any solution from being obtained because the initial state is too far away from the equilibrium state that is being sought—it is outside the radius of convergence. Thus, there is an algorithmic restriction on the increment size.

Automatic incrementation

In most cases the default automatic incrementation scheme is preferred because it will select increment sizes based on computational efficiency. However, you must ensure that the time increments are such that the time integration results in an accurate solution. Abaqus/Standard does not have any built in checks to ensure integration accuracy.

Input File Usage: *ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Direct incrementation

Direct user control of the increment size is also provided; if you have considerable experience with a particular problem, you may be able to select a more economical approach.

Input File Usage: *ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT, DIRECT

Abaqus/CAE Usage: A transient eddy current analysis is not supported in Abaqus/CAE.

Ill-conditioning in eddy current analyses with electrically nonconductive regions

In an eddy current analysis it is very common that large portions of the model consist of electrically nonconductive regions, such as air and/or a vacuum. In such cases it is well known that the associated stiffness matrix can be very ill-conditioned; i.e., it can have many singularities (Bíró, 1999). Abaqus uses a special iterative solution technique to prevent the ill-conditioned matrix from negatively impacting the computed electric and magnetic fields. The default implementation works well for many problems. However, there can be situations in which the default numerical scheme fails to converge or results in a noisy solution. In such cases adding a "small" amount of artificial electrical conductivity to the nonconductive domain may help regularize the problem and allow Abaqus to converge to the correct solution. The artificial electrical conductivity should be chosen such that the electromagnetic waves propagating through these regions undergo little modification and, in particular, do not experience the sharp exponential decay that is typical when such fields impinge upon a real conductor. It is

recommended that you set the artificial conductivity to be about five to eight orders of magnitude less than that of any of the conductors in the model.

As an alternative to specifying electrical conductivity in the nonconductive domain, Abaqus also provides a stabilization scheme to help mitigate the effects of the ill-conditioning. You can provide input to this stabilization algorithm by specifying the stabilization factor, which is assumed to be 1.0 by default if the stabilization scheme is used. Higher values of the stabilization factor lead to more stabilization, while lower values of the stabilization factor lead to less stabilization.

Input File Usage: Use the following to use stabilization in a time-harmonic procedure:

*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC, STABILIZATION=*stabilization factor*

Use the following to use stabilization in a transient procedure:

*ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT, STABILIZATION=*stabilization factor*

Abaqus/CAE Usage: The default stabilization factor cannot be modified in Abaqus/CAE.

Prescribed conductor motion

You can specify conductor motion by prescribing the direction and magnitude of the translational or rotational velocity vector over an element set representing the conductor. Only a single conductor motion is allowed in a step.

Input File Usage: Use the following to prescribe a translational velocity:

*MOTION, ELEMENT, TRANSLATION

Use the following to prescribe a rotational velocity:

*MOTION, ELEMENT, ROTATION

Abaqus/CAE Usage: Specifying conductor motion is not supported in Abaqus/CAE.

Initial conditions

Initial values of temperature and/or predefined field variables can be specified. These values affect only temperature and/or field-variable-dependent material properties, if any. Initial conditions on the electric and/or magnetic fields cannot be specified in an eddy current analysis.

Boundary conditions

Electromagnetic elements use an element edge-based interpolation of the fields. The degrees of freedom of the element are not associated with the user-defined nodes, which only define the geometry of the element. Consequently, the standard node-based method of specifying boundary conditions cannot be used with electromagnetic elements. The method used for specifying boundary conditions for electromagnetic elements is described in the following paragraphs.

Boundary conditions in Abaqus typically refer to what are traditionally known as Dirichlet-type boundary conditions in the literature, where the values of the primary variable are known on the whole

boundary or on a portion of the boundary. The alternative, Neumann-type boundary conditions, refer to situations where the values of the conjugate to the primary variable are known on portions of the boundary. In Abaqus Neumann-type boundary conditions are represented as surface loads in the finite-element formulation.

For electromagnetic boundary value problems, Dirichlet boundary conditions on an enclosing surface must be specified as $\mathbf{A} \times \mathbf{n}$, where $\mathbf{n}$ is the outward normal to the surface, as discussed in this section. Neumann boundary conditions must be specified as the surface current density vector, $\mathbf{K} = \mathbf{H} \times \mathbf{n}$, as discussed in "Loads" below.

In Abaqus Dirichlet boundary conditions are specified as magnetic vector potential, $\mathbf{A}$, on (element-based) surfaces that represent symmetry planes and/or external boundaries in the model; Abaqus computes $\mathbf{A} \times \mathbf{n}$ for the representative surfaces. In applications where the electromagnetic fields are driven by a current-carrying coil that is close to the workpiece, the model may span a domain that is up to 10 times the characteristic length scale associated with the coil/workpiece assembly. In such cases, the electromagnetic fields are assumed to have decayed sufficiently in the far-field, and the value of the magnetic vector potential can be set to zero in the far-field boundary. On the other hand, in applications such as one where a conductor is embedded in a uniform (but varying time-harmonically in a time-harmonic eddy current analysis or with a more general time variation in a transient eddy current analysis) far-field magnetic field, it may be necessary to specify nonzero values of the magnetic vector potential on some portions of the external boundary. In this case an alternative method to model the same physical phenomena is to specify the corresponding unique value of surface current density, $\mathbf{K}$, on the far-field boundary (see "Loads" below). $\mathbf{K}$ can be computed based on known values of the far-field magnetic field.

A surface without any prescribed boundary condition corresponds to a surface with zero surface currents, or no loads.

Nonuniform boundary conditions can be defined with user subroutine **UDEMPOTENTIAL**.

Prescribing boundary conditions in a time-harmonic eddy current analysis

In a time-harmonic eddy current analysis the boundary conditions are assumed to be time harmonic and are applied simultaneously to both the real and imaginary parts of the magnetic vector potential. It is not possible to specify Dirichlet boundary conditions on the real parts and Neumann boundary conditions on the imaginary parts and vice versa. Abaqus automatically restrains both the real and imaginary parts even if only one part is prescribed explicitly. The unspecified part is assumed to have a magnitude of zero.

When you prescribe the boundary condition on an element-based surface for a time-harmonic eddy current analysis (see "Element-based surface definition," Section 2.3.2), you must specify the surface name, the region type label (S), the boundary condition type label, an optional orientation name, the magnitude of the real part of the boundary condition, the direction vector for the real part of the boundary condition, the magnitude of the imaginary part of the boundary condition, and the direction vector for the imaginary part of the boundary condition. The optional orientation name defines the local coordinate system in which the components of the magnetic vector potential are defined. By default, the components are defined with respect to the global directions. The specified direction vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

During a time-harmonic eddy current analysis, frequency-dependent boundary conditions can be prescribed as described in "Frequency-dependent boundary conditions in a time-harmonic eddy current analysis" below.

Input File Usage: Use the following option in a time-harmonic eddy current analysis to define both the real (in-phase) and imaginary (out-of-phase) parts of the boundary condition on element-based surfaces:

∗D EM POTENTIAL
surface name, S, bc type label, orientation, magnitude of real part, direction vector of real part, magnitude of imaginary part, direction vector of imaginary part

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Electrical/Magnetic** for the **Category** and **Magnetic vector potential** for the **Types for Selected Step**; **Distribution**: **Uniform** or **User-defined**; *real components + imaginary components*

Prescribing boundary conditions in a transient eddy current analysis

The method of specification of the boundary condition for a transient eddy current analysis is substantially similar to that of the time-harmonic eddy current analysis, except that the concepts of real and imaginary are not relevant any more. In this case you specify the magnitude of the magnetic vector potential, followed by its direction vector. The specified direction vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

During a transient eddy current analysis, prescribed boundary conditions can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2).

Input File Usage: Use the following option in a transient eddy current analysis to define the boundary condition on element-based surfaces:

∗D EM POTENTIAL
surface name, S, bc type label, orientation, magnitude, direction vector

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Abaqus/CAE Usage: Transient eddy current analysis is not supported in Abaqus/CAE.

Frequency-dependent boundary conditions in a time-harmonic eddy current analysis

An amplitude definition can be used to specify the amplitude of a boundary condition as a function of frequency ("Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

∗AMPLITUDE, NAME=*name*
∗D EM POTENTIAL, AMPLITUDE=*name*

Abaqus/CAE Usage: Load or Interaction module: **Create Amplitude**: **Name**: *amplitude_name*
Load module: **Create Boundary Condition**: choose **Electrical/Magnetic** for the **Category** and **Magnetic vector potential** for the **Types for Selected Step**; **Amplitude**: *amplitude_name*

Loads

The following types of electromagnetic loads can be applied in an eddy current analysis (see "Prescribing electromagnetic loads for eddy current and/or magnetostatic analyses" in "Electromagnetic loads," Section 34.4.5, for details):

- Element-based distributed volume current density vector: $\mathbf{J}_0(\mathbf{x})$ in a time-harmonic eddy current analysis, and $\mathbf{J}(\mathbf{x}, t)$ in a transient eddy current analysis
- Surface-based distributed surface current density vector: $\mathbf{K}_0(\mathbf{x})$ in a time-harmonic eddy current analysis, and $\mathbf{K}(\mathbf{x}, t)$ in a transient eddy current analysis

All loads in a time-harmonic eddy current analysis are assumed to be time-harmonic with the excitation frequency. During a transient eddy current analysis all loads can be varied using an amplitude definition (see "Amplitude curves," Section 34.1.2).

Nonuniform loads can be specified using user subroutines **`UDECURRENT`** and **`UDSECURRENT`**.

Frequency-dependent loading in a time-harmonic eddy current analysis

In a time-harmonic eddy current analysis, an amplitude definition can be used to specify the amplitude of a load as a function of frequency ("Amplitude curves," Section 34.1.2).

Predefined fields

Predefined temperature and field variables can be specified in an eddy current analysis; however, user-defined fields that allow the value of field variables at a material point to be redefined via user subroutine **`USDFLD`** are not supported. These values affect only temperature and/or field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

Magnetic material behavior (see "Magnetic permeability," Section 26.5.3) must be specified everywhere in the model. Only linear magnetic behavior is supported in a time-harmonic eddy current analysis, but nonlinear magnetic behavior is also supported in a transient eddy current analysis. Linear magnetic behavior can be defined by specifying the magnetic permeability directly, while nonlinear magnetic behavior is defined in terms of one or more B–H curves. Electrical conductivity (see "Electrical conductivity," Section 26.5.1) must be specified in conductor regions. All other material properties are ignored in an eddy current analysis.

Both magnetic permeability and electrical conductivity can be functions of frequency, predefined temperature, and field variables in a time-harmonic eddy current analysis. In a transient eddy current analysis, all material behavior can be functions of predefined temperature and/or field variables.

Permanent magnets (see "Magnetic permeability," Section 26.5.3) can be included in a transient eddy current analysis.

Elements

Electromagnetic elements must be used to model all regions in an eddy current analysis. Unlike conventional finite elements, which use node-based interpolation, these elements use edge-based interpolation with the tangential components of the magnetic vector potential along element edges serving as the primary degrees of freedom.

Electromagnetic elements are available in Abaqus/Standard in two dimensions (planar only) and three dimensions (see "Choosing the appropriate element for an analysis type," Section 27.1.3). The planar elements are formulated in terms of an in-plane magnetic vector potential, thereby the magnetic flux density and magnetic field vectors only have an out-of-plane component. The electric field and the current density vectors are in-plane for the planar elements.

Output

Eddy current analysis provides output only to the output database (`.odb`) file (see "Output to the output database," Section 4.1.3). Output to the data (`.dat`) file and to the results (`.fil`) file is not available. For the first four vector quantities listed below (which are derived from the magnetic vector potential and the constitutive equations) and the applied volume current density vector, the magnitude and components of the real and imaginary parts are output in a time-harmonic eddy current procedure.

Element centroidal variables:

EMB	Magnitude and components of the magnetic flux density vector, $\mathbf{B}$.
EMH	Magnitude and components of the magnetic field vector, $\mathbf{H}$.
EME	Magnitude and components of the electric field vector, $\mathbf{E}$.
EMCD	Magnitude and components of the eddy current density vector, $\sigma\mathbf{E}$, in conducting regions.
EMCDA	Magnitude and components of the applied volume current density vector.
EMBF	Magnetic body force intensity vector (force per unit volume per unit time) due to flow of current.
EMBFC	Complex magnetic body force intensity vector (real and imaginary parts of the force per unit volume) due to flow of current. Only available in a time-harmonic eddy current analysis.
EMJH	Rate of Joule heating (amount of heat per unit volume per unit time) due to flow of current.
TEMP	Temperature at the centroid of the element. For a time-harmonic eddy current analysis, this value represents the temperature that is used for evaluating the temperature-dependent material properties.

Whole element variables:

ELJD	Total rate of Joule heating (amount of heat per unit time) due to flow of current in an element.
EVOL	Element volume.

Whole model variables:

ALLJD	Rate of Joule heating (amount of heat per unit time) summed over the model or an element set.

Input file template

The following is an input file template that makes use of linear magnetic material behavior in a time-harmonic eddy current analysis:

```
*HEADING
…
*MATERIAL, NAME=mat1
*MAGNETIC PERMEABILITY
Data lines to define magnetic permeability
*ELECTRICAL CONDUCTIVITY
Data lines to define electrical conductivity in the conductor region
**
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
Data line to specify excitation frequencies
*D EM POTENTIAL
Data lines to define boundary conditions on magnetic vector potential
*DECURRENT
Data lines to define element-based distributed volume current density vector
*DSECURRENT
Data lines to define surface-based distributed surface current density vector
*OUTPUT, FIELD or HISTORY
Data lines to request element-based output
*ENERGY OUTPUT
Data line to request whole model Joule heat dissipation output
*END STEP
```

The following is an input file template that makes use of nonlinear magnetic material behavior in a transient eddy current analysis:

```
*HEADING
…
*MATERIAL, NAME=mat1
*MAGNETIC PERMEABILITY, NONLINEAR
*NONLINEAR BH, DIR=direction
Data lines to define nonlinear B-H curve
*ELECTRICAL CONDUCTIVITY
Data lines to define electrical conductivity in the conductor region
```

```
**
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TRANSIENT
*D EM POTENTIAL
```
Data lines to define boundary conditions on magnetic vector potential
```
*DECURRENT
```
Data lines to define element-based distributed volume current density vector
```
*DSECURRENT
```
Data lines to define surface-based distributed surface current density vector
```
*OUTPUT, FIELD or HISTORY
```
Data lines to request element-based output
```
*ENERGY OUTPUT
```
Data line to request whole model Joule heat dissipation output
```
*END STEP
```

Additional reference

- Bíró, O., "Edge Element Formulation of Eddy Current Problems," Computer Methods in Applied Mechanics and Engineering, vol. 169, pp. 391–405, 1999.

6.7.6 MAGNETOSTATIC ANALYSIS

Product: Abaqus/Standard

References

- "Electromagnetic analysis procedures," Section 6.7.1
- "Magnetic permeability," Section 26.5.3
- "Electromagnetic loads," Section 34.4.5
- *MAGNETOSTATIC
- *D EM POTENTIAL
- *DECURRENT
- *DSECURRENT
- "UDECURRENT," Section 1.1.24 of the Abaqus User Subroutines Reference Guide
- "UDEMPOTENTIAL," Section 1.1.25 of the Abaqus User Subroutines Reference Guide
- "UDSECURRENT," Section 1.1.27 of the Abaqus User Subroutines Reference Guide

Overview

Magnetostatic problems:

- solve the magnetostatic approximation of Maxwell's equations describing electromagnetic phenomena and compute the magnetic fields due to direct currents;
- involve only magnetic fields, which are assumed to be vary slowly in time such that electromagnetic coupling can be neglected;
- require the use of electromagnetic elements in the whole domain;
- require that magnetic permeability is specified in the whole domain;
- can be solved with nonlinear magnetic behavior; and
- can be solved using continuum elements in two- and three-dimensional space.

Magnetostatic analysis

A direct current creates a static magnetic field in the space surrounding the current carrying region. For applications where the magnitude of the direct current can be assumed to be a constant or to vary slowly with time, coupling between magnetic and electric fields can be neglected. The magnetostatic approximation to Maxwell's equations involves the magnetic fields only. Magnetostatic analysis provides a solution for applications where the above assumptions are valid.

Electromagnetic elements must be used to model the response of all the regions in a magnetostatic analysis, including regions such as current carrying coils and the surrounding space. To obtain accurate solutions, the outer boundary of the space being modeled must be at least a few characteristic length scales away from the region of interest on all sides.

Electromagnetic elements use an element edge-based interpolation of the fields instead of the standard node-based interpolation. The user-defined nodes only define the geometry of the elements; and the degrees of freedom of the element are not associated with these nodes, which has implications for applying boundary conditions (see "Boundary conditions" below).

Governing field equations

The magnetic fields are governed by the magnetostatic approximation to Maxwell's equations describing electromagnetic phenomena.

It is convenient to introduce a magnetic vector potential, $\mathbf{A}$, such that the magnetic flux density vector $\mathbf{B} = \nabla \times \mathbf{A}$. The solution procedure seeks a static magnetic response due to, for example, an impressed direct volume current density distribution, $\mathbf{J}$, in some regions of the model. The magnetostatic approximation to Maxwell's equations is given by

$$\nabla \times (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A}) = \mathbf{J}$$

in terms of the field quantities, $\mathbf{A}$ and $\mathbf{J}$, and the magnetic permeability tensor, $\boldsymbol{\mu}$. The magnetic permeability relates the magnetic flux density, $\mathbf{B}$, to the magnetic field, $\mathbf{H}$, through a constitutive equation of the form: $\mathbf{B} = \boldsymbol{\mu} \cdot \mathbf{H}$.

The variational form of the above equation is

$$\int_V \nabla \times \delta\mathbf{A} \cdot (\boldsymbol{\mu}^{-1} \cdot \nabla \times \mathbf{A})\, dV = \int_V \delta\mathbf{A} \cdot \mathbf{J}\, dV + \int_S \delta\mathbf{A} \cdot \mathbf{K}\, dS,$$

where $\delta\mathbf{A}$ represents the variation of the magnetic vector potential, and $\mathbf{K}$ represents the applied tangential surface current density, if any, at the external surfaces.

Abaqus/Standard solves the variational form of Maxwell's equations for the components of the magnetic vector potential. The other field quantities are derived from the magnetic vector potential. In the following discussion, the governing equations are written for a linear medium.

Defining the magnetic behavior

The magnetic behavior of the electromagnetic medium can be linear or nonlinear. Linear magnetic behavior is characterized by a magnetic permeability tensor that is assumed to be independent of the magnetic field. It is defined through direct specification of the absolute magnetic permeability tensor, $\boldsymbol{\mu}$, which can be isotropic, orthotropic, or fully anisotropic (see "Magnetic permeability," Section 26.5.3). The magnetic permeability can also depend on temperature and/or predefined field variables.

Nonlinear magnetic behavior is characterized by magnetic permeability that depends on the strength of the magnetic field. The nonlinear magnetic material model in Abaqus is suitable for ideally soft magnetic materials characterized by a monotonically increasing response in B–H space, where B and H refer to the strengths of the magnetic flux density vector and the magnetic field vector, respectively. Nonlinear magnetic behavior is defined through direct specification of one or more B–H curves that provide B as a function of H and, optionally, temperature and/or predefined field variables, in one or more directions. Nonlinear magnetic behavior can be isotropic, orthotropic, or transversely isotropic (which is a special case of the more general orthotropic behavior).

Magnetostatic analysis

Magnetostatic analysis provides the magnetic flux density and the magnetic field at a given value of the impressed direct current.

Input File Usage: *MAGNETOSTATIC

Ill-conditioning in magnetostatic analyses

In magnetostatic analysis the stiffness matrix can be very ill-conditioned; i.e., it can have many singularities. Abaqus uses a special iterative solution technique to prevent the ill-conditioned matrix from negatively impacting the computed magnetic fields. The default implementation works well for many problems. However, there can be situations in which the default numerical scheme fails to converge. Abaqus provides a stabilization scheme to help mitigate the effects of the ill-conditioning. You can provide input to this stabilization algorithm by specifying the stabilization factor, which is assumed to be 1.0 by default if the stabilization scheme is used. Higher values of the stabilization factor lead to more stabilization, while lower values of the stabilization factor lead to less stabilization.

Input File Usage: *MAGNETOSTATIC, STABILIZATION=*stabilization factor*

Initial conditions

Initial values of temperature and/or predefined field variables can be specified. These values affect only temperature and/or field-variable-dependent material properties, if any. Initial conditions on magnetic fields cannot be specified in a magnetostatic analysis.

Boundary conditions

Electromagnetic elements use an element edge-based interpolation of the fields. The degrees of freedom of the element are not associated with the user-defined nodes, which only define the geometry of the element. Consequently, the standard node-based method of specifying boundary conditions cannot be used with electromagnetic elements.

Boundary conditions in Abaqus typically refer to what are traditionally known as Dirichlet-type boundary conditions in the literature, where the values of the primary variable are known on the whole boundary or on a portion of the boundary. The alternative, Neumann-type boundary conditions, refer to situations where the values of the conjugate to the primary variable are known on portions of the boundary. In Abaqus, Neumann-type boundary conditions are represented as surface loads in the finite element formulation.

For electromagnetic boundary value problems, including magnetostatic problems, Dirichlet boundary conditions on an enclosing surface must be specified as $\mathbf{A} \times \mathbf{n}$, where $\mathbf{n}$ is the outward normal to the surface, as discussed in this section. Neumann boundary conditions must be specified as the surface current density vector, $\mathbf{K} = \mathbf{H} \times \mathbf{n}$, as discussed in "Loads" below.

In Abaqus, Dirichlet boundary conditions are specified as magnetic vector potential, $\mathbf{A}$, on (element-based) surfaces that represent symmetry planes and/or external boundaries in the model; Abaqus computes $\mathbf{A} \times \mathbf{n}$ for the representative surfaces. The model may span a domain that is up to 10 times some characteristic length scale for the problem. In such cases the magnetic fields are assumed to have decayed sufficiently in the far-field, and the value of the magnetic vector potential can be set to zero in the far-field boundary. On the other hand, in applications such as one where a magnetic material is embedded in a uniform far-field magnetic field, it may be necessary to specify nonzero values of the magnetic vector potential on some portions of the external boundary. In this case an alternative method to model the same physical phenomena is to specify the corresponding unique value of surface current density, $\mathbf{K}$, on the far-field boundary (see "Loads" below). $\mathbf{K}$ can be computed based on known values of the far-field magnetic field.

In a magnetostatic analysis the boundary conditions are assumed to be either constant or varying slowly with time. The time variation can be specified using an amplitude definition ("Amplitude curves," Section 34.1.2)

A surface without any prescribed boundary condition corresponds to a surface with zero surface currents or no loads.

When you prescribe the boundary condition on an element-based surface (see "Element-based surface definition," Section 2.3.2), you must specify the surface name, the region type label (S), the boundary condition type label, an optional orientation name, the magnitude of the magnetic vector potential, and the direction vector for the magnetic vector potential. The optional orientation name defines the local coordinate system in which the components of the magnetic vector potential are defined. By default, the components are defined with respect to the global directions.

The specified vector components are normalized by Abaqus and, thus, do not contribute to the magnitude of the boundary condition.

Nonuniform boundary conditions can be defined with user subroutine **UDEMPOTENTIAL**.

Input File Usage: Use the following option to define both the real (in-phase) and imaginary (out-of-phase) parts of the boundary condition on element-based surfaces:

*D EM POTENTIAL
surface name, S, bc type label, orientation, magnitude, direction vector

where the boundary condition type label (*bc type label*) can be MVP for a uniform boundary condition or MVPNU for a nonuniform boundary condition.

Loads

The following types of electromagnetic loads can be applied in a magnetostatic analysis (see "Prescribing electromagnetic loads for eddy current and/or magnetostatic analyses" in "Electromagnetic loads," Section 34.4.5, for details):

- Element-based distributed volume current density vector, $\mathbf{J}(\mathbf{x})$
- Surface-based distributed surface current density vector, $\mathbf{K}(\mathbf{x})$

During the analysis the prescribed load can be varied using an amplitude definition ("Amplitude curves," Section 34.1.2).

Predefined fields

Predefined temperature and field variables can be specified in a magnetostatic analysis; however, user-defined fields that allow the value of field variables at a material point to be redefined via user subroutine **USDFLD** are not supported. These values affect only temperature and/or field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

The magnetic behavior (see "Magnetic permeability," Section 26.5.3) must be defined everywhere in the model, either by specifying the absolute magnetic permeability tensor for linear magnetic behavior or by specifying the B–H curve-based response for nonlinear magnetic behavior. All other material properties, including electrical conductivity, are ignored in a magnetostatic analysis. The magnetic behavior can be functions of predefined temperature and/or field variables.

Permanent magnets (see "Magnetic permeability," Section 26.5.3) can be included in magnetostatic analyses.

Elements

Electromagnetic elements must be used to model all regions in a magnetostatic analysis. Unlike conventional finite elements, which use node-based interpolation, these elements use edge-based interpolation with the tangential components of the magnetic vector potential along element edges serving as the primary degrees of freedom.

Electromagnetic elements are available in Abaqus/Standard in two dimensions (planar only) and three dimensions (see "Choosing the appropriate element for an analysis type," Section 27.1.3). The planar elements are formulated in terms of an in-plane magnetic vector potential, thereby the magnetic flux density and magnetic field vectors have only an out-of-plane component.

Output

Magnetostatic analysis provides output only to the output database (**.odb**) file (see "Output to the output database," Section 4.1.3). Output to the data (**.dat**) file and to the results (**.fil**) file is not available.

Element centroidal variables:

EMB	Magnitude and components of the magnetic flux density vector, $\mathbf{B}$.
EMCDA	Magnitude and components of the applied volume current density vector.
EMH	Magnitude and components of the magnetic field vector, $\mathbf{H}$.
TEMP	Temperature at the centroid of the element.

Whole element variables:

EVOL	Element volume.

Input file template

***HEADING**
…
MATERIAL, NAME=mat1*
***MAGNETIC PERMEABILITY, NONLINEAR**
Data lines to define magnetic permeability for linear magnetic behavior; no data required here for nonlinear magnetic behavior
NONLINEAR BH, DIR=direction*
Data lines to define nonlinear B-H curve
******
***STEP**
***MAGNETOSTATIC**
Data line to define time incrementation
***D EM POTENTIAL**
Data lines to define boundary conditions on magnetic vector potential
***DECURRENT**
Data lines to define element-based distributed volume current density vector
***DSECURRENT**
Data lines to define surface-based distributed surface current density vector
***OUTPUT, FIELD or HISTORY**
Data lines to request element-based output
***END STEP**

6.8 Coupled pore fluid flow and stress analysis

- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1
- "Geostatic stress state," Section 6.8.2

6.8.1 COUPLED PORE FLUID DIFFUSION AND STRESS ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Pore fluid flow properties," Section 26.6.1
- *SOILS
- "Defining pore fluid expansion" in "Defining a fluid-filled porous material," Section 12.12.3 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring an effective stress analysis for fluid-filled porous media" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A coupled pore fluid diffusion/stress analysis:

- is used to model single phase, partially or fully saturated fluid flow through porous media;
- can be performed in terms of either total pore pressure or excess pore pressure by including or excluding the pore fluid weight;
- requires the use of pore pressure elements with associated pore fluid flow properties defined;
- can, optionally, also model heat transfer due to conduction in the soil skeleton and the pore fluid, and convection due to the flow of the pore fluid, through the use of coupled temperature–pore pressure displacement elements;
- can be transient or steady-state;
- can be linear or nonlinear; and
- can include pore pressure contact between bodies (see "Pore fluid contact properties," Section 37.4.1).

Typical applications

Some of the more common coupled pore fluid diffusion/stress (and, optionally, thermal) analysis problems that can be analyzed with Abaqus/Standard are:

- **Saturated flow:** Soil mechanics problems generally involve fully saturated flow, since the solid is fully saturated with ground water. Typical examples of saturated flow include consolidation of soils under foundations and excavation of tunnels in saturated soil.
- **Partially saturated flow:** Partially saturated flow occurs when the wetting liquid is absorbed into or exsorbed from the medium by capillary action. Irrigation and hydrology problems generally include partially saturated flow.

- **Combined flow:** Combined fully saturated and partially saturated flow occurs in problems such as seepage of water through an earth dam, where the position of the phreatic surface (the boundary between fully saturated and partially saturated soil) is of interest.
- **Moisture migration:** Although not normally associated with soil mechanics, moisture migration problems can also be solved using the coupled pore fluid diffusion/stress procedure. These problems may involve partially saturated flow in polymeric materials such as paper towels and sponge-like materials; in the biomedical industry they may also involve saturated flow in hydrated soft tissues.
- **Combined heat transfer and pore fluid flow:** In some applications, such as a source of heat buried in soil, it is important to model the coupling between the mechanical deformation, pore fluid flow, and heat transfer. In such problems the difference in the thermal expansion coefficients between the soil and the pore fluid often plays an important role in determining the rate of diffusion of the pore fluid and heat from the source.

Flow through porous media

A porous medium is modeled in Abaqus/Standard by a conventional approach that considers the medium as a multiphase material and adopts an effective stress principle to describe its behavior. The porous medium modeling provided considers the presence of two fluids in the medium. One is the "wetting liquid," which is assumed to be relatively (but not entirely) incompressible. Often the other is a gas, which is relatively compressible. An example of such a system is soil containing ground water. When the medium is partially saturated, both fluids exist at a point; when it is fully saturated, the voids are completely filled with the wetting liquid. The elementary volume, dV, is made up of a volume of grains of solid material, dV_g; a volume of voids, dV_v; and a volume of wetting liquid, $dV_w \leq dV_v$, that is free to move through the medium if driven. In some systems (for example, systems containing particles that absorb the wetting liquid and swell in the process) there may also be a significant volume of trapped wetting liquid, dV_t.

The porous medium is modeled by attaching the finite element mesh to the solid phase; fluid can flow through this mesh. The mechanical part of the model is based on the effective stress principle defined in "Effective stress principle for porous media," Section 2.8.1 of the Abaqus Theory Guide.

The model also uses a continuity equation for the mass of wetting fluid in a unit volume of the medium. This equation is described in "Continuity statement for the wetting liquid phase in a porous medium," Section 2.8.4 of the Abaqus Theory Guide. It is written with pore pressure (the average pressure in the wetting fluid at a point in the porous medium) as the basic variable (degree of freedom 8 at the nodes). The conjugate flux variable is the volumetric flow rate at the node, $\mathbf{f}$. The porous medium is partially saturated when the pore liquid pressure, u_w, is negative.

Coupled flow and heat transfer through porous media

Optionally, heat transfer due to conduction in the soil skeleton and pore fluid, as well as convection in the pore fluid, can also be modeled. This capability represents an enhancement to the basic pore fluid flow capabilities discussed in the earlier paragraphs and requires the use of coupled temperature–pore pressure elements that have temperature as an additional degree of freedom (degree of freedom 11 at the nodes) in addition to the pore pressure and the displacement components. When you use the coupled

temperature–pore pressure elements, Abaqus solves the heat transfer equation in addition to and in a fully coupled manner with the continuity equation and the mechanical equilibrium equations. Only linear brick, first-order axisymmetric, and second-order modified tetrahedrons are available for modeling coupled heat transfer with pore fluid flow and mechanical deformation.

Total and excess pore fluid pressure

The coupled pore fluid diffusion/stress analysis capability can provide solutions either in terms of total or "excess" pore fluid pressure. The excess pore fluid pressure at a point is the pore fluid pressure in excess of the hydrostatic pressure required to support the weight of pore fluid above the elevation of the material point. The difference between total and excess pore pressure is relevant only for cases in which gravitational loading is important; for example, when the loading provided by the hydrostatic pressure in the pore fluid is large or when effects like "wicking" (transient capillary suction of liquid into a dry column) are being studied. Total pore pressure solutions are provided when the gravity distributed load is used to define the gravity load on the model. Excess pore pressure solutions are provided in all other cases; for example, when gravity loading is defined with body force distributed loads.

Steady-state analysis

Steady-state coupled pore pressure/effective stress analysis assumes that there are no transient effects in the wetting liquid continuity equation; that is, the steady-state solution corresponds to constant wetting liquid velocities and constant volume of wetting liquid per unit volume in the continuum. Thus, for example, thermal expansion of the liquid phase has no effect on the steady-state solution: it is a transient effect. Therefore, the time scale chosen during steady-state analysis is relevant only to rate effects in the constitutive model used for the porous medium (excluding creep and viscoelasticity, which are disabled in steady-state analysis).

Mechanical loads and boundary conditions can be changed gradually over the step by referring to an amplitude curve to accommodate possible geometric nonlinearities in the response.

The steady-state coupled equations are strongly unsymmetric; therefore, the unsymmetric matrix solution and storage scheme is used automatically for steady-state analysis steps (see "Defining an analysis," Section 6.1.2).

If heat transfer is modeled using the coupled temperature–pore pressure elements, the steady-state solution neglects all transient effects in the heat transfer equation and provides only the steady-state temperature distribution.

Input File Usage: *SOILS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Steady state**

Incrementation

You can specify a fixed time increment size in a coupled pore fluid diffusion/stress analysis, or Abaqus/Standard can select the time increment size automatically. Automatic incrementation is recommended because the time increments in a typical diffusion analysis can increase by several

orders of magnitude during the simulation. If you do not activate automatic incrementation, fixed time increments will be used.

Input File Usage: Use the following option to activate automatic incrementation in steady-state analysis:

∗SOILS, UTOL=*any arbitrary nonzero value*

The solution does not depend on the value specified for UTOL; this value is simply a flag for automatic incrementation.

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Steady state**; **Incrementation**: **Type: Automatic**

Transient analysis

In a transient coupled pore pressure/effective stress analysis the backward difference operator is used to integrate the continuity equation and the heat transfer equation (if heat transfer is modeled): this operator provides unconditional stability so that the only concern with respect to time integration is accuracy. You can provide the time increments, or they can be selected automatically.

The coupled partially saturated flow equations are strongly unsymmetric, so the unsymmetric solver is used automatically if you request partially saturated analysis (by including absorption/exsorption behavior in the material definition). The unsymmetric solver is also activated automatically when gravity distributed loading is used during a soils consolidation analysis.

For fully saturated flow analyses in which finite-sliding coupled pore pressure-displacement contact is modeled using contact pairs, certain contributions to the model's stiffness matrix are unsymmetric. Using the unsymmetric solver can sometimes improve convergence in such cases since Abaqus does not automatically do so.

For fully saturated flow analyses in which heat transfer is also modeled, the contributions to the model's stiffness matrix arising from convective heat transfer due to pore fluid flow are unsymmetric. Using the unsymmetric solver can sometimes improve convergence in such cases since Abaqus does not automatically do so.

Spurious oscillations due to small time increments

The integration procedure used in Abaqus/Standard for consolidation analysis introduces a relationship between the minimum usable time increment and the element size, as shown below for fully saturated and partially saturated flows. If time increments smaller than these values are used, spurious oscillations may appear in the solution (except for partially saturated cases when linear elements or modified triangular elements are used; in these cases Abaqus/Standard uses a special integration scheme for the wetting liquid storage term to avoid the problem). These nonphysical oscillations may cause problems if pressure-sensitive plasticity is used to model the porous medium and may lead to convergence difficulties in partially saturated analyses. If the problem requires analysis with smaller time increments than the relationships given below allow, a finer mesh is required. Generally there is no upper limit on the time step except accuracy, since the integration procedure is unconditionally stable unless nonlinearities cause convergence problems.

Fully saturated flow

A simple guideline that can be used for the minimum usable time increment in the case of fully saturated flow is

$$\Delta t > \frac{\gamma_w(1+\beta v_w)}{6E\mathrm{k}}\left(1-\frac{E}{K_g}\right)^2(\Delta\ell)^2,$$

where

- Δt is the time increment,
- γ_w is the specific weight of the wetting liquid,
- E is the Young's modulus of the soil,
- k is the permeability of the soil (see "Permeability," Section 26.6.2),
- v_w is the magnitude of the velocity of the pore fluid,
- β is the velocity coefficient in Forchheimer's flow law ($\beta = 0$ in the case of Darcy flow),
- K_g is the bulk modulus of the solid grains (see "Porous bulk moduli," Section 26.6.3), and
- $\Delta\ell$ is a typical element dimension.

Partially saturated flow

In partially saturated flow cases the corresponding guideline for the minimum time increment is

$$\Delta t > \frac{\gamma_w n^0(1+\beta v_w)}{6k_s\mathrm{k}}\frac{ds}{du_w}(\Delta\ell)^2,$$

where

- s is the saturation;
- k_s is the permeability-saturation relationship;
- ds/du_w is the rate of change of saturation with respect to pore pressure (see "Sorption," Section 26.6.4);
- n^0 is the initial porosity of the material; and the other parameters are as defined for the case of fully saturated flow.

Fixed incrementation

If you choose fixed time incrementation, fixed time increments equal to the size of the user-specified initial time increment, Δt_0, will be used. Fixed incrementation is not generally recommended because the time increments in a typical diffusion analysis can increase over several orders of magnitude during the simulation; automatic incrementation is usually a better choice.

Input File Usage: ∗SOILS, CONSOLIDATION
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **Type: Fixed**, **Increment size:** Δt_0

Automatic incrementation

If you choose automatic time incrementation, you must specify two (three if heat transfer is also modeled) tolerance parameters.

The accuracy of the time integration of the flow continuity equations is governed by the maximum wetting liquid pore pressure change, Δu_w^{max}, allowed in an increment. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment in the analysis.

If heat transfer is modeled, the accuracy of time integration is also governed by the maximum temperature change, $\Delta\theta_{max}$, allowed in an increment. Abaqus/Standard restricts the time increments to ensure that this value is not exceeded at any node (except nodes with boundary conditions) during any increment of the analysis.

The accuracy of the integration of the time-dependent (creep) material behavior is governed by the maximum strain rate change allowed at any point during an increment, $errtol \geq (\dot{\bar{\varepsilon}}^{cr}_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}_{t})\Delta t$, as described in "Rate-dependent plasticity: creep and swelling," Section 23.2.4.

Input File Usage: If heat transfer is not modeled:

*SOILS, CONSOLIDATION, UTOL=Δu_w^{max}, , CETOL=*errtol*

If heat transfer is modeled:

*SOILS, CONSOLIDATION, UTOL=Δu_w^{max}, DELTMX=$\Delta\theta_{max}$, CETOL=*errtol*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **Type: Automatic**, **Max. pore pressure change per increment:** Δu_w^{max}, **Creep/swelling/viscoelastic strain error tolerance:** *errtol*

Specifying the maximum temperature change per increment is not supported in Abaqus/CAE.

Ending a transient analysis

Transient soils analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or the time period ends, whichever comes first. When heat transfer is not modeled, steady state is defined by a maximum permitted rate of change of pore pressure with time: when all pore pressures are changing at less than the user-defined rate, the analysis terminates. However, with heat transfer included, the analysis terminates only when both the pore pressure and temperature are changing at less than the user-defined rates.

Input File Usage: Use the following option to end the analysis when the time period is reached:

*SOILS, CONSOLIDATION, END=PERIOD (default)

Use the following option to end the analysis based on the pore pressure and, if heat transfer is modeled, temperature change rate:

*SOILS, CONSOLIDATION, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**; **Incrementation**: **End step when pore pressure change rate is less than**

If heat transfer is modeled, directly specifying the temperature change rate to define steady state is not supported in Abaqus/CAE.

Neglecting creep during a transient analysis

You can specify that creep or viscoelastic response should be neglected during a consolidation analysis, even if creep or viscoelastic material properties have been defined.

Input File Usage: *SOILS, CONSOLIDATION, CREEP=NONE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Soils**: **Basic**: **Pore fluid response: Transient consolidation**, toggle off **Include creep/swelling/viscoelastic behavior**

Unstable problems

Some types of analyses may develop local instabilities, such as surface wrinkling, material instability, or local buckling. In such cases it may not be possible to obtain a quasi-static solution, even with the aid of automatic incrementation. Abaqus/Standard offers the option to stabilize this class of problems by applying damping throughout the model in such a way that the viscous forces introduced are sufficiently large to prevent instantaneous buckling or collapse but small enough not to affect the behavior significantly while the problem is stable. The available automatic stabilization schemes are described in detail in "Automatic stabilization of unstable problems" in "Solving nonlinear problems," Section 7.1.1.

Optional modeling of coupled heat transfer

When coupled temperature–pore pressure elements are used, heat transfer is modeled in these elements by default. However, you may optionally choose to switch off heat transfer within these elements during some steps in the analysis. This feature may be helpful in reducing computation time during certain phases in the analysis when heat transfer is not an important part of the overall physics of the problem.

Input File Usage: Use the following option either during a transient or a steady-state procedure to suppress heat transfer modeling:

*SOILS, CONSOLIDATION, HEAT=NO

Abaqus/CAE Usage: Switching off the heat transfer part of the physics is not supported in Abaqus/CAE.

Units

In coupled problems where two or more different fields are being solved, you must be careful when choosing the units of the problem. If the choice of units is such that the numbers generated by the equations for the different fields differ by many orders of magnitude, the precision on some computers may be insufficient to resolve the numerical ill-conditioning of the coupled equations. Therefore, choose units that avoid badly conditioned matrices. For example, consider using units of Mpascal instead of pascal for the stress equilibrium equations to reduce the disparity between the magnitudes of the stress equilibrium equations and the pore flow continuity equations.

Initial conditions

Initial conditions can be applied as defined in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Defining initial pore fluid pressures

Initial values of pore fluid pressures, u_w, can be defined at the nodes.

Input File Usage: *INITIAL CONDITIONS, TYPE=PORE PRESSURE

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Pore pressure** for the **Types for Selected Step**

Defining initial void ratios

Initial values of the void ratio, e, can be given at the nodes. The void ratio is defined as the ratio of the volume of voids to the volume of solid material (see "Effective stress principle for porous media," Section 2.8.1 of the Abaqus Theory Guide). The evolution of void ratio is governed by the deformation of the different phases of the material, as discussed in detail in "Constitutive behavior in a porous medium," Section 2.8.3 of the Abaqus Theory Guide.

Input File Usage: *INITIAL CONDITIONS, TYPE=RATIO

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Void ratio** for the **Types for Selected Step**

Defining initial saturation

Initial saturation values, s, can be given at the nodes. Saturation is defined as the ratio of wetting fluid volume to void volume (see "Effective stress principle for porous media," Section 2.8.1 of the Abaqus Theory Guide).

Input File Usage: *INITIAL CONDITIONS, TYPE=SATURATION

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Saturation** for the **Types for Selected Step**

Defining initial stresses

An initial (effective) stress field can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Most geotechnical problems begin from a geostatic state, which is a steady-state equilibrium configuration of the undisturbed soil or rock body under geostatic loading and usually includes both horizontal and vertical components. It is important to establish these initial conditions correctly so that the problem begins from an equilibrium state. The geostatic procedure can be used to verify that the user-defined initial stresses are indeed in equilibrium with the given geostatic loads and boundary conditions (see "Geostatic stress state," Section 6.8.2).

Input File Usage: Use one of the following options:

*INITIAL CONDITIONS, TYPE=STRESS
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Mechanical** for the **Category** and **Stress** or **Geostatic stress** for the **Types for Selected Step**

Defining initial temperature

Initial temperature values can be defined at the nodes.

Input File Usage: *INITIAL CONDITIONS, TYPE=TEMPERATURE

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Temperature** for the **Types for Selected Step**

Boundary conditions

Boundary conditions can be applied to displacement degrees of freedom 1–6 and to pore pressure degree of freedom 8 ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). In addition, boundary conditions can also be applied to temperature degree of freedom 11 if heat transfer is modeled using coupled temperature–pore pressure elements. During the analysis prescribed boundary conditions can be varied by referring to an amplitude curve ("Amplitude curves," Section 34.1.2). If no amplitude reference is given, the default variation of a boundary condition in a coupled pore fluid diffusion/stress analysis step is as defined in "Defining an analysis," Section 6.1.2.

If the pore pressure is prescribed with a boundary condition, fluid is assumed to enter and leave through the node as needed to maintain the prescribed pressure. Likewise, if the temperature is prescribed with a boundary condition, heat is assumed to enter and leave through the node as needed to maintain the prescribed temperature.

Loads

The following loading types can be prescribed in a coupled pore fluid diffusion/stress analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements." The magnitude and direction of gravitational loading are usually defined by using the gravity distributed load type.
- Pore fluid flow is controlled as described in "Pore fluid flow," Section 34.4.7.

If heat transfer is modeled, the following types of thermal loading can also be prescribed ("Thermal loads," Section 34.4.4). These loads are not supported in Abaqus/CAE during a coupled thermal pore pressure/stress analysis.

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Predefined fields

The following predefined fields can be prescribed, as described in "Predefined fields," Section 34.6.1:

- For a coupled pore fluid diffusion/stress analysis that does not model heat transfer and uses regular pore pressure elements, temperature is not a degree of freedom and nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- Predefined temperature fields are not allowed in coupled pore fluid diffusion/stress analysis that also models heat transfer. Boundary conditions should be used instead to specify temperatures, as described earlier.
- The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any.

Material options

Any of the mechanical constitutive models available in Abaqus/Standard can be used to model the porous material.

In problems formulated in terms of total pore pressure, you must include the density of the dry material in the material definition (see "Density," Section 21.2.1).

You can use a permeability material property to define the specific weight of the wetting liquid, γ_w; the permeability, $\mathbf{k}$, and its dependence on the void ratio, e, and saturation, k_s; and the flow velocity, β (see "Permeability," Section 26.6.2).

You can define the compressibility of the solid grains and of the permeating fluid in both fully and partially saturated flow problems (see "Elastic behavior of porous materials," Section 22.3.1). If you do not specify the porous bulk moduli, the materials are assumed to be fully incompressible.

For partially saturated flow you must define the porous medium's absorption/exsorption behavior (see "Sorption," Section 26.6.4).

Gel swelling ("Swelling gel," Section 26.6.5) and volumetric moisture swelling of the solid skeleton ("Moisture swelling," Section 26.6.6) can be included in partially saturated cases. These effects are usually associated with modeling of moisture migration in polymeric systems rather than with geotechnical systems.

Thermal properties if heat transfer is modeled

In problems that model heat transfer, the thermal conductivity for either the solid material or the permeating fluid, or more commonly for both phases, must be defined. Only isotropic conductivity can be specified for the pore fluid. The specific heat and density of the phases must also be defined for transient heat transfer problems. Latent heat for the phases can be defined if changes in internal energy due to phase changes are important. See "Thermal properties: overview," Section 26.2.1, for details on defining thermal properties in Abaqus. Examples of problems that model fully coupled heat transfer along with pore fluid diffusion and mechanical deformation can be found in "Consolidation around a cylindrical heat source," Section 1.15.7 of the Abaqus Benchmarks Guide, and "Permafrost thawing–pipeline interaction," Section 10.1.6 of the Abaqus Example Problems Guide.

The thermal properties can be defined separately for the solid material and the permeating fluid.

Input File Usage: To define the conductivity, specific heat, density, and latent heat of the permeating fluid, use the following options:

```
*CONDUCTIVITY, TYPE=ISO, PORE FLUID
*SPECIFIC HEAT, PORE FLUID
*LATENT HEAT, PORE FLUID
*DENSITY, PORE FLUID
```

To define the conductivity, specific heat, density, and latent heat of the solid material, use the following options:

```
*EXPANSION, TYPE=ISO or ORTHO or ANISO
*SPECIFIC HEAT
*DENSITY
*LATENT HEAT
```

Abaqus/CAE Usage: Defining the thermal properties and the density of the permeating fluid is not supported in Abaqus/CAE.

To define the conductivity, specific heat, density, and latent heat of the solid material, use the following options:

Property module: material editor:
Thermal→Conductivity: **Type**: **Isotropic**
Thermal→Specific Heat
General→Density
Thermal→Latent Heat

Thermal expansion

Thermal expansion can be defined separately for the solid material and for the permeating fluid. In such a case you should repeat the expansion material property, with the necessary parameters, to define the different thermal expansion effects (see "Thermal expansion," Section 26.1.2). Thermal expansion will be active only in a consolidation (transient) analysis.

Input File Usage: To define the thermal expansion of the permeating fluid:

*EXPANSION, TYPE=ISO, PORE FLUID

To define the thermal expansion of the solid material:

*EXPANSION, TYPE=ISO or ORTHO or ANISO

Abaqus/CAE Usage: To define the thermal expansion of the permeating fluid:

Property module: material editor: **Other→Pore Fluid→Pore Fluid Expansion**

To define the thermal expansion of the solid material:

Property module: material editor: **Mechanical→Expansion**

Elements

The analysis of flow through porous media in Abaqus/Standard is available for plane strain, axisymmetric, and three-dimensional problems. The modeling of coupled heat transfer effects is available only for axisymmetric and three-dimensional problems. Continuum pore pressure elements are provided for modeling fluid flow through a deforming porous medium in a coupled pore fluid diffusion/stress analysis. These elements have pore pressure degree of freedom 8 in addition to displacement degrees of freedom 1–3. Heat transfer through the porous medium can also be modeled using continuum coupled temperature–pore pressure elements. These elements have temperature degree of freedom 11 in addition to pore pressure degree of freedom 8 and displacement degrees of freedom 1–3. Stress/displacement elements can be used in parts of the model without pore fluid flow. See "Choosing the appropriate element for an analysis type," Section 27.1.3, for more information.

Output

The element output available for a coupled pore fluid diffusion/stress analysis includes the usual mechanical quantities such as (effective) stress; strain; energies; and the values of state, field, and user-defined variables. In addition, the following quantities associated with pore fluid flow are available:

VOIDR	Void ratio, e.
POR	Pore pressure, u_w.
SAT	Saturation, s.
GELVR	Gel volume ratio, n_t.
FLUVR	Total fluid volume ratio, n_f.
FLVEL	Magnitude and components of the pore fluid effective velocity vector, $\mathbf{f}$.

FLVELM	Magnitude, $\mathbf{f}$, of the pore fluid effective velocity vector.
FLVEL*n*	Component *n* of the pore fluid effective velocity vector (*n*=1, 2, 3).

If heat transfer is modeled, the following element output variables associated with heat transfer are also available:

HFL	Magnitude and components of the heat flux vector.
HFL*n*	Component *n* of the heat flux vector (*n*=1, 2, 3).
HFLM	Magnitude of the heat flux vector.
TEMP	Integration point temperatures.

The nodal output available includes the usual mechanical quantities such as displacements, reaction forces, and coordinates. In addition, the following quantities associated with pore fluid flow are available:

CFF	Concentrated fluid flow at a node.
POR	Pore pressure at a node.
RVF	Reaction fluid volume flux due to prescribed pressure. This flux is the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pressure boundary condition. A positive value of RVF indicates that fluid is entering the model.
RVT	Reaction total fluid volume (computed only in a transient analysis). This value is the time integrated value of RVF.

If heat transfer is modeled, the following nodal output variables associated with heat transfer are also available:

NT	Nodal point temperatures.
RFL	Reaction flux values due to prescribed temperature.
RFL*n*	Reaction flux value *n* at a node (*n*=11, 12, …).
CFL	Concentrated flux values.
CFL*n*	Concentrated flux value *n* at a node (*n*=11, 12, …).

All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

```
*HEADING
…
**********************************
**
** Material definition
**
**********************************
```

***MATERIAL, NAME=soil**
Data lines to define mechanical properties of the solid material
…
***EXPANSION**
Data lines to define the thermal expansion coefficient of the solid grains
***EXPANSION, TYPE=ISO, PORE FLUID**
Data lines to define the thermal expansion coefficient of the permeating fluid
***PERMEABILITY, SPECIFIC=**γ_w
Data lines to define permeability, $\mathbf{k}$*, as a function of the void ratio,* $\boldsymbol{e}$
***PERMEABILITY, TYPE=SATURATION**
Data lines to define the dependence of permeability on saturation, $k_s(s)$
***PERMEABILITY, TYPE=VELOCITY**
Data lines to define the velocity coefficient, $\beta(e)$
***POROUS BULK MODULI**
Data line to define the bulk moduli of the solid grains and the permeating fluid
***SORPTION, TYPE=ABSORPTION**
Data lines to define absorption behavior
***SORPTION, TYPE=EXSORPTION**
Data lines to define exsorption behavior
***SORPTION, TYPE=SCANNING**
Data lines to define scanning behavior (between absorption and exsorption)
***GEL**
Data line to define gel behavior in partially saturated flow
***MOISTURE SWELLING**
Data lines to define moisture swelling strain as a function of saturation in partially saturated flow
***CONDUCTIVITY**
Data lines to define thermal conductivity of the solid grains if heat transfer is modeled
***CONDUCTIVITY,TYPE=ISO, PORE FLUID**
Data lines to define thermal conductivity of the permeating fluid if heat transfer is modeled
***SPECIFIC HEAT**
Data lines to define specific heat of the solid grains if transient heat transfer is modeled
***SPECIFIC HEAT,PORE FLUID**
Data lines to define specific heat of the permeating fluid if transient heat transfer is modeled
***DENSITY**
Data lines to define density of the solid grains if transient heat transfer is modeled
***DENSITY,PORE FLUID**
Data lines to define density of the permeating fluid if transient heat transfer is modeled
***LATENT HEAT**
Data lines to define latent heat of the solid grains if phase change due to temperature change is modeled
***LATENT HEAT,PORE FLUID**

Data lines to define latent heat of the permeating fluid if phase change due to temperature change is modeled

…

```
**********************************
**
** Boundary conditions and initial conditions
**
**********************************
*BOUNDARY
```

Data lines to specify zero-valued boundary conditions

```
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC
```

Data lines to specify initial stresses

```
*INITIAL CONDITIONS, TYPE=PORE PRESSURE
```

Data lines to define initial values of pore fluid pressures

```
*INITIAL CONDITIONS, TYPE=RATIO
```

Data lines to define initial values of the void ratio

```
*INITIAL CONDITIONS, TYPE=SATURATION
```

Data lines to define initial saturation

```
*INITIAL CONDITIONS, TYPE=TEMPERATURE
```

Data lines to define initial saturation

```
*AMPLITUDE, NAME=name
```

Data lines to define amplitude variations

```
**********************************
**
** Step 1: Optional step to ensure an equilibrium
** geostatic stress field
**
**********************************
*STEP
*GEOSTATIC
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
```

Data lines to specify mechanical loading

```
*FLOW and/or *SFLOW and/or *DFLOW and/or *DSFLOW
```

Data lines to specify pore fluid flow

```
*CFLUX and/or *DFLUX
```

Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled

```
*BOUNDARY
```

Data lines to specify displacements or pore pressures

```
*END STEP
**********************************
**
** Step 2: Coupled pore diffusion/stress analysis step
```

```
**
**********************************
*STEP (,NLGEOM)
** Use NLGEOM to include geometric nonlinearities
*SOILS
```
Data line to define incrementation
```
*CLOAD and/or *DLOAD and/or *DSLOAD
```
Data lines to specify mechanical loading
```
*FLOW and/or *SFLOW and/or *DFLOW and/or *DSFLOW
```
Data lines to specify pore fluid flow
```
*CFLUX and/or *DFLUX
```
Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled
```
*FILM
```
Data lines referring to film property table if heat transfer is modeled
```
*BOUNDARY
```
Data lines to specify displacements, pore pressures, or temperatures
```
*END STEP
```

6.8.2 GEOSTATIC STRESS STATE

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1
- *GEOSTATIC
- "Configuring a geostatic stress field procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A geostatic stress field procedure:

- is used to verify that the initial geostatic stress field is in equilibrium with applied loads and boundary conditions and to iterate, if necessary, to obtain equilibrium;
- accounts for pore pressure degrees of freedom when pore pressure elements are used, and accounts for temperature degrees of freedom when coupled temperature–pore pressure elements are used;
- is usually the first step of a geotechnical analysis, followed by a coupled pore fluid diffusion/stress (with or without heat transfer) or static analysis procedure; and
- can be linear or nonlinear.

Establishing geostatic equilibrium

The geostatic procedure is normally used as the first step of a geotechnical analysis; in such cases gravity loads are applied during this step. Ideally, the loads and initial stresses should exactly equilibrate and produce zero deformations. However, in complex problems it may be difficult to specify initial stresses and loads that equilibrate exactly.

Abaqus/Standard provides two procedures for establishing the initial equilibrium. The first procedure is applicable to problems for which the initial stress state is known at least approximately. The second, enhanced, procedure is also applicable for cases in which the initial stresses are not known; it is supported for only a limited number of elements and materials.

Establishing equilibrium when the initial stress state is approximately known

The geostatic procedure requires that the initial stresses are close to the equilibrium state; otherwise, the displacements corresponding to the equilibrium state might be large. Abaqus/Standard checks for equilibrium during the geostatic procedure and iterates, if needed, to obtain a stress state that equilibrates the prescribed boundary conditions and loads. This stress state, which is a modification of the stress field defined by the initial conditions ("Initial conditions in Abaqus/Standard and Abaqus/Explicit,"

Section 34.2.1), is then used as the initial stress field in a subsequent static or coupled pore fluid diffusion/stress (with or without heat transfer) analysis.

If the stresses given as initial conditions are far from equilibrium under the geostatic loading and there is some nonlinearity in the problem definition, this iteration process may fail. Therefore, you should ensure that the initial stresses are reasonably close to equilibrium.

If the deformations produced during the geostatic step are significant compared to the deformations caused by subsequent loading, the definition of the initial state should be reexamined.

If heat transfer is modeled during the geostatic step through the use of coupled temperature–pore pressure elements, the initial temperature field and thermal loads, if specified, must be such that the system is relatively close to a state of thermal equilibrium. Steady-state heat transfer is assumed during a geostatic step.

Input File Usage: *GEOSTATIC

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Geostatic**

Establishing equilibrium when the initial stress state is unknown

To obtain equilibrium in cases when the initial stress state is unknown or is known only approximately, you can invoke an enhanced procedure. Abaqus automatically computes the equilibrium corresponding to the initial loads and the initial configuration, allowing only small displacements within user-specified tolerances. (The default tolerance is 10^{-5}.) The procedure is available with a limited number of elements and materials and is intended to be used in analyses in which the material response is primarily elastic; that is, inelastic deformations are small.

The procedure is supported for both geometrically linear and geometrically nonlinear analyses. However, in general, the performance in the geometrically linear case will be better. Therefore, it might be advantageous to obtain the initial equilibrium in a geometrically linear step, even though a geometrically nonlinear analysis is performed in subsequent steps.

Input File Usage: Use the following option to invoke the enhanced procedure:

*GEOSTATIC, UTOL=*displacement tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Geostatic**: **Incrementation**
tabbed page: **Automatic**: **Max. displacement change**

Limitations

The following limitations apply to the enhanced procedure:

- It is supported only for a limited number of elements (see “Elements” below) and materials (see “Material options” below). When the procedure is used with nonsupported elements or material models, Abaqus issues a warning message. In this case it is the user’s responsibility to ensure that the displacement tolerances are larger than the displacements in the analysis; otherwise, convergence problems may occur.
- It can be used in a restart analysis only if it had been used in the previous analysis.

Optional modeling of coupled heat transfer

When coupled temperature–pore pressure elements are used, heat transfer is modeled in these elements by default. However, you may optionally choose to switch off heat transfer within these elements during a geostatic step. This feature may be helpful in reducing computation time if temperature and associated heat flow effects are not important.

Input File Usage: Use the following option to suppress heat transfer modeling:

∗GEOSTATIC, HEAT=NO

Abaqus/CAE Usage: Switching off the heat transfer part of the physics is not supported in Abaqus/CAE.

Vertical equilibrium in a porous medium

Most geotechnical problems begin from a geostatic state, which is a steady-state equilibrium configuration of the undisturbed soil or rock body under geostatic loading. The equilibrium state usually includes both horizontal and vertical stress components. It is important to establish these initial conditions correctly so that the problem begins from an equilibrium state. Since such problems often involve fully or partially saturated flow, the initial void ratio of the porous medium, e^0, the initial pore pressure, u_w, and the initial effective stress must all be defined.

If the magnitude and direction of the gravitational loading are defined by using the gravity distributed load type, a total, rather than excess, pore pressure solution is used (see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1). This discussion is based on the total pore pressure formulation.

The *z*-axis points vertically in this discussion, and atmospheric pressure is neglected. We assume that the pore fluid is in hydrostatic equilibrium during the geostatic procedure so that

$$\frac{du_w}{dz} = -\gamma_w,$$

where γ_w is the user-defined specific weight of the pore fluid (see "Permeability," Section 26.6.2). (The pore fluid is not in hydrostatic equilibrium if there is significant steady-state flow of pore fluid through the porous medium: in that case a steady-state coupled pore fluid diffusion/stress analysis must be performed to establish the initial conditions for any subsequent transient calculations—see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1.) If we also take γ_w to be independent of *z* (which is usually the case, since the fluid is almost incompressible), this equation can be integrated to define

$$u_w = \gamma_w(z_w^0 - z),$$

where z_w^0 is the height of the *phreatic surface*, at which $u_w = 0$ and above which $u_w < 0$ and the pore fluid is only partially saturated.

We usually assume that there are no significant shear stresses τ_{xz}, τ_{yz}. Then, equilibrium in the vertical direction is

$$\frac{d\sigma_{zz}}{dz} = \rho g + sn^0\gamma_w,$$

where ρ is the dry density of the porous solid material (the dry mass per unit volume), g is the gravitational acceleration, n^0 is the initial porosity of the material, and s is the saturation, $0 \leq s \leq 1.0$ (see "Permeability," Section 26.6.2). Since porosity is the ratio of pore volume to total volume and the void ratio is the ratio of pore volume to solids volume, n^0 is defined from the initial void ratio by

$$n^0 = \frac{e^0}{1+e^0}.$$

Abaqus/Standard requires that the initial value of the effective stress, $\overline{\boldsymbol{\sigma}}$, be given as an initial condition ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Effective stress is defined from the total stress, $\boldsymbol{\sigma}$, by

$$\overline{\boldsymbol{\sigma}} = \boldsymbol{\sigma} + su_w\mathbf{I},$$

where $\mathbf{I}$ is a unit matrix. Combining this definition with the equilibrium statement in the z-direction and hydrostatic equilibrium in the pore fluid gives

$$\frac{d\overline{\sigma}_{zz}}{dz} = \rho g - \gamma_w\left(s(1-n^0) - \frac{ds}{dz}(z_w^0 - z)\right) \quad \text{for} \quad z < z_1^0, \quad \text{and}$$

$$\frac{d\overline{\sigma}_{zz}}{dz} = \rho g \quad \text{for} \quad z_1^0 \leq z,$$

again using the assumption that γ_w is independent of z. z_1^0 is the position of the surface that separates the dry soil from the partially saturated soil. The soil is assumed to be dry ($s = 0$) for $z_1^0 < z$, and it is assumed to be partially saturated for $z_w^0 < z < z_1^0$ and fully saturated for $z \leq z_w^0$.

In many cases s is constant. For example, in fully saturated flow $s = 1.0$ everywhere below the phreatic surface. If we further assume that the initial porosity, n^0, and the dry density of the porous medium, ρ, are also constant, the above equation is readily integrated to give

$$\overline{\sigma}_{zz} = \rho g(z - z^0) - \gamma_w s(1-n^0)(z - z_w^0) \quad \text{for} \quad z < z_1^0, \quad \text{and}$$

$$\overline{\sigma}_{zz} = \rho g(z - z^0) \quad \text{for} \quad z_1^0 \leq z,$$

where z^0 is the position of the surface of the porous medium, $z_w^0 < z^0$.

In more complicated cases where s, n^0, and/or ρ vary with height, the equation must be integrated in the vertical direction to define the initial values of $\overline{\sigma}_{zz}(z)$.

Horizontal equilibrium in a porous medium

In many geotechnical applications there is also horizontal stress, typically caused by tectonic action. If the pore fluid is under hydrostatic equilibrium and $\tau_{xz} = \tau_{yz} = 0$, equilibrium in the horizontal directions requires that the horizontal components of effective stress do not vary with horizontal position: $\overline{\sigma}_h(z)$ only, where $\overline{\sigma}_h$ is any horizontal component of effective stress.

Initial conditions

The initial effective geostatic stress field, $\overline{\boldsymbol{\sigma}}$, is given by defining initial stress conditions. Unless the enhanced procedure is used, the initial state of stress must be close to being in equilibrium with the applied loads and boundary conditions. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

You can specify that the initial stresses vary only with elevation, as described in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1. In this case the horizontal stress is typically assumed to be a fraction of the vertical stress: those fractions are defined in the x- and y-directions.

In problems involving partially or fully saturated porous media, initial pore fluid pressures, u_w, void ratios, e^0, and saturation values, s, must be given (see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1).

In partially saturated cases the initial pore pressure and saturation values must lie on or between the absorption and exsorption curves (see "Sorption," Section 26.6.4). A partially saturated problem is illustrated in "Wicking in a partially saturated porous medium," Section 1.9.3 of the Abaqus Benchmarks Guide.

You may also specify initial temperatures in the model if heat transfer is modeled during the geostatic procedure.

Boundary conditions

Boundary conditions can be applied to displacement degrees of freedom 1–6 and to pore pressure degree of freedom 8 ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). If coupled temperature–pore pressure elements are used, boundary conditions on temperature degree of freedom 11 can also be applied to nodes belonging to these elements. If the enhanced procedure is used and nonzero boundary conditions are applied, it is the user's responsibility to ensure that the displacements corresponding to the tolerances specified are larger than the displacements in the analysis; otherwise, the displacements at the nonzero boundary nodes will be reset to zero with the tolerances specified.

The boundary conditions should be in equilibrium with the initial stresses and applied loads. If the horizontal stress is nonzero, horizontal equilibrium must be maintained by fixing the boundary conditions on any nonhorizontal edges of the finite element model in the horizontal direction or by using infinite elements ("Infinite elements," Section 28.3.1). If heat transfer is modeled, the temperature boundary conditions should be in equilibrium with the initial temperature field and applied thermal loads.

Loads

The following loading types can be prescribed in a geostatic stress field procedure:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can also be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements." The magnitude and direction of gravitational loading are defined by using the gravity or body force distributed load types.
- Pore fluid flow is controlled as described in "Pore fluid flow," Section 34.4.7.

If heat transfer is modeled, the following types of thermal loading can also be prescribed ("Thermal loads," Section 34.4.4). These loads are not supported in Abaqus/CAE during a geostatic analysis.

- Concentrated heat fluxes.
- Body fluxes and distributed surface fluxes.
- Convective film conditions and radiation conditions; film properties can be made a function of temperature.

Predefined fields

The following predefined fields can be specified in a geostatic stress field procedure, as described in "Predefined fields," Section 34.6.1:

- For a geostatic analysis that does not model heat transfer and uses regular pore pressure elements, temperature is not a degree of freedom and nodal temperatures can be specified.
- Predefined temperature fields are not allowed in a geostatic analysis that also models heat transfer. Boundary conditions should be used instead to specify temperatures, as described earlier.
- The values of user-defined field variables can be specified; these values affect only field-variable-dependent material properties, if any.

Material options

Any of the mechanical constitutive models available in Abaqus/Standard can be used to model the porous solid material. However, the enhanced procedure can be used only with the elastic, porous elastic, extended Cam-clay plasticity, and Mohr-Coulomb plasticity models. Use of a nonsupported material model with this procedure may lead to poor convergence or no convergence if displacements are larger than the displacements corresponding to the tolerances specified. Abaqus will issue a warning message if the procedure is used with a nonsupported material model.

If a porous medium will be analyzed subsequent to the geostatic procedure, pore fluid flow quantities such as permeability and sorption should be defined (see "Pore fluid flow properties," Section 26.6.1).

If heat transfer is modeled, thermal properties such as conductivity, specific heat, and density should be defined for both the solid and the pore fluid phases (see "Thermal properties if heat transfer is modeled"

in "Coupled pore fluid diffusion and stress analysis," Section 6.8.1, for details on how to specify separate thermal properties for the two phases).

Elements

Any of the stress/displacement elements in Abaqus/Standard can be used in a geostatic procedure. Continuum pore pressure elements can also be used for modeling fluid in a deforming porous medium. These elements have pore pressure degree of freedom 8 in addition to displacement degrees of freedom 1–3. However, the enhanced procedure can be used only with continuum and cohesive elements with pore pressure degrees of freedom and the corresponding stress/displacements elements. Use of nonsupported elements with this procedure may lead to poor convergence or no convergence if displacements are larger than the displacements corresponding to the tolerances specified. Abaqus will issue a warning message if the procedure is used with a nonsupported element.

Continuum elements that couple temperature, pore pressure, and displacement can be used if heat transfer needs to be modeled. These elements have temperature degree of freedom 11 in addition to pore pressure degree of freedom 8 and displacement degrees of freedom 1–3. See "Choosing the appropriate element for an analysis type," Section 27.1.3, for more information.

Output

The element output available for a coupled pore fluid diffusion/stress analysis includes the usual mechanical quantities such as (effective) stress; strain; energies; and the values of state, field, and user-defined variables. In addition, the following quantities associated with pore fluid flow are available:

VOIDR	Void ratio, e.
POR	Pore pressure, u_w.
SAT	Saturation, s.
GELVR	Gel volume ratio, n_t.
FLUVR	Total fluid volume ratio, n_f.
FLVEL	Magnitude and components of the pore fluid effective velocity vector, $\mathbf{f}$.
FLVELM	Magnitude, $\mathbf{f}$, of the pore fluid effective velocity vector.
FLVEL*n*	Component *n* of the pore fluid effective velocity vector (*n*=1, 2, 3).

If heat transfer is modeled, the following element output variables associated with heat transfer are also available:

HFL	Magnitude and components of the heat flux vector.
HFL*n*	Component *n* of the heat flux vector (*n*=1, 2, 3).
HFLM	Magnitude of the heat flux vector.
TEMP	Integration point temperatures.

The nodal output available includes the usual mechanical quantities such as displacements, reaction forces, and coordinates. In addition, the following quantities associated with pore fluid flow are available:

POR	Pore pressure at a node.
RVF	Reaction fluid volume flux due to prescribed pressure. This flux is the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pressure boundary condition. A positive value of RVF indicates fluid is entering the model.

If heat transfer is modeled, the following nodal output variables associated with heat transfer are also available:

NT	Nodal point temperatures.
RFL	Reaction flux values due to prescribed temperature.
RFL*n*	Reaction flux value *n* at a node (*n*=11, 12, …).
CFL	Concentrated flux values.
CFL*n*	Concentrated flux value *n* at a node (*n*=11, 12, …).

All of the output variable identifiers are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

```
*HEADING
…
*MATERIAL, NAME=mat1
Data lines to define mechanical properties of the solid material
…
*DENSITY
Data lines to define the density of the dry material
*PERMEABILITY, SPECIFIC=γw
Data lines to define permeability, k, as a function of the void ratio, e
*CONDUCTIVITY
Data lines to define thermal conductivity of the solid grains if heat transfer is modeled
*CONDUCTIVITY,TYPE=ISO, PORE FLUID
Data lines to define thermal conductivity of the permeating fluid if heat transfer is modeled
*SPECIFIC HEAT
Data lines to define specific heat of the solid grains if transient heat transfer is modeled in a
subsequent step
*SPECIFIC HEAT,PORE FLUID
Data lines to define specific heat of the permeating fluid if transient heat transfer is modeled in a subsequent step
*DENSITY
Data lines to define density of the solid grains if transient heat transfer is modeled in a subsequent
step
*DENSITY,PORE FLUID
Data lines to define density of the permeating fluid if transient heat transfer is modeled in a
subsequent step
```

***LATENT HEAT**
Data lines to define latent heat of the solid grains if phase change due to temperature change is modeled
***LATENT HEAT,PORE FLUID**
Data lines to define latent heat of the permeating fluid if phase change due to temperature change is modeled
…
***INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC**
Data lines to define the initial stress state
***INITIAL CONDITIONS, TYPE=PORE PRESSURE**
Data lines to define initial values of pore fluid pressures
***INITIAL CONDITIONS, TYPE=RATIO**
Data lines to define initial values of the void ratio
***INITIAL CONDITIONS, TYPE=SATURATION**
Data lines to define initial saturation
***INITIAL CONDITIONS, TYPE=TEMPERATURE**
Data lines to define initial temperature
***BOUNDARY**
Data lines to define zero-valued boundary conditions
******
***STEP**
***GEOSTATIC**
***CLOAD and/or *DLOAD and/or *DSLOAD**
Data lines to specify mechanical loading
***FLOW and/or *SFLOW and/or *DFLOW and/or *DSFLOW**
Data lines to specify pore fluid flow
***CFLUX and/or *DFLUX**
Data lines to define concentrated and/or distributed heat fluxes if heat transfer is modeled
***BOUNDARY**
Data lines to specify displacements or pore pressures
***END STEP**

6.9 Mass diffusion analysis

- “Mass diffusion analysis,” Section 6.9.1

6.9.1 MASS DIFFUSION ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- *MASS DIFFUSION
- "Configuring a mass diffusion procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Creating and modifying prescribed conditions," Section 16.4 of the Abaqus/CAE User's Guide
- "Defining a concentrated concentration flux," Section 16.9.33 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a body concentration flux," Section 16.9.35 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a surface concentration flux," Section 16.9.34 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A mass diffusion analysis:

- models the transient or steady-state diffusion of one material through another, such as the diffusion of hydrogen through a metal;
- requires the use of mass diffusion elements; and
- can be used to model temperature and/or pressure-driven mass diffusion.

Governing equations

The governing equations for mass diffusion are an extension of Fick's equations: they allow for nonuniform solubility of the diffusing substance in the base material and for mass diffusion driven by gradients of temperature and pressure. The basic solution variable (used as the degree of freedom at the nodes of the mesh) is the "normalized concentration" (often also referred to as the "activity" of the diffusing material), $\phi \stackrel{\text{def}}{=} c/s$, where c is the mass concentration of the diffusing material and s is its solubility in the base material. Therefore, when the mesh includes dissimilar materials that share nodes, the normalized concentration is continuous across the interface between the different materials.

For example, a diatomic gas that dissociates during diffusion can be described using Sievert's law: $c = sp^{\frac{1}{2}}$, where p is the partial pressure of the diffusing gas. Combining Sievert's law with the definition of normalized concentration given earlier, $\phi = c/s = p^{\frac{1}{2}}$. Equilibrium requires the partial pressure to be continuous across an interface, so normalized concentration will be continuous as well. If an expression

other than Sievert's law defines the relationship between concentration and partial pressure for a diffusing material, solubility should be defined accordingly.

The diffusion problem is defined from the requirement of mass conservation for the diffusing phase:

$$\int_V \frac{dc}{dt}\, dV + \int_S \mathbf{n} \cdot \mathbf{J}\, dS = 0,$$

where V is any volume whose surface is S, $\mathbf{n}$ is the outward normal to S, $\mathbf{J}$ is the flux of concentration of the diffusing phase, and $\mathbf{n} \cdot \mathbf{J}$ is the concentration flux leaving S.

Diffusion is assumed to be driven by the gradient of a general chemical potential, which gives the behavior

$$\mathbf{J} = -s\mathbf{D} \cdot \left[\frac{\partial \phi}{\partial \mathbf{x}} + \kappa_s \frac{\partial}{\partial \mathbf{x}} \left(\ln(\theta - \theta^Z) \right) + \kappa_p \frac{\partial p}{\partial \mathbf{x}} \right],$$

where $\mathbf{D}(c, \theta, \mathbf{f})$ is the diffusivity; $s(\theta, \mathbf{f})$ is the solubility; $\kappa_s(c, \theta, \mathbf{f})$ is the "Soret effect" factor, providing diffusion because of temperature gradient; θ is the temperature; θ^Z is the value of absolute zero on the temperature scale being used; $\kappa_p(c, \theta, \mathbf{f})$ is the pressure stress factor, providing diffusion driven by the gradient of the equivalent pressure stress, $p \overset{\text{def}}{=} -\text{trace}(\boldsymbol{\sigma})/3$; $\boldsymbol{\sigma}$ is stress; and $\mathbf{f}$ are any predefined field variables.

Whenever D, κ_s, or κ_p depends on concentration, the problem becomes nonlinear and the system of equations becomes nonsymmetric. In practical cases the dependence on concentration is quite strong, so the nonsymmetric matrix storage and solution scheme is invoked automatically when a mass diffusion analysis is performed (see "Defining an analysis," Section 6.1.2).

Fick's law

Mass diffusion behavior is often described by Fick's law (Crank, 1956):

$$\mathbf{J} = -\mathbf{D} \cdot \frac{\partial c}{\partial \mathbf{x}}.$$

Fick's law is offered in Abaqus/Standard as a special case of the general chemical potential relation. To establish the relationship between Fick's law and the general chemical potential, we write Fick's law as

$$\mathbf{J} = -\mathbf{D} \cdot \left(s \frac{\partial \phi}{\partial \mathbf{x}} + \phi \frac{\partial s}{\partial \mathbf{x}} \right).$$

In most practical cases $s = s(\theta)$, and we can write

$$\mathbf{J} = -s\mathbf{D} \cdot \frac{\partial \phi}{\partial \mathbf{x}} - \mathbf{D} \cdot \frac{c}{s} \frac{\partial s}{\partial \theta} \frac{\partial \theta}{\partial \mathbf{x}}.$$

The two terms in this equation describe the normalized concentration and temperature-driven diffusion, respectively. The normalized concentration-driven diffusion term is identical to that given in

the general relation. The temperature-driven diffusion term in Fick's law is recovered in the general relation if

$$\kappa_s = \frac{c\,(\theta - \theta^Z)}{s^2}\frac{\partial s}{\partial \theta}\,.$$

This conversion is done automatically in Abaqus/Standard when you request Fick's law (see "Diffusivity," Section 26.4.1).

An extended form of Fick's law can also be chosen by specifying a nonzero value for κ_p:

$$\mathbf{J} = -\mathbf{D}\cdot\left(\frac{\partial c}{\partial \mathbf{x}} + s\kappa_p\frac{\partial p}{\partial \mathbf{x}}\right).$$

In this case Abaqus/Standard will still define κ_s automatically as discussed earlier.

Units

The units of concentration are commonly given as parts per million ($\boldsymbol{P}$). On the basis of the applicability of Sievert's law to the mass diffusion, the units of solubility are $PLF^{-1/2}$, where $\boldsymbol{F}$ is force and $\boldsymbol{L}$ is length. The units of the Soret effect factor are $F^{1/2}L^{-1}$. The units of the pressure stress factor are $LF^{-1/2}$, and the units of equivalent pressure stress are FL^{-2}. The diffusivity, $\mathbf{D}$, has units of L^2T^{-1}, where $\boldsymbol{T}$ is time. The concentration flux, $\mathbf{J}$, then has units of PLT^{-1}; and the concentration volumetric flux, $Q = \int_S \mathbf{n}\cdot\mathbf{J}\,dS$, has units of PL^3T^{-1}.

Steady-state analysis

Steady-state mass diffusion analysis provides the steady-state solution directly: the rate of change of concentration with respect to time is omitted from the governing diffusion equation in steady-state analysis. In nonlinear cases iteration may be necessary to achieve a converged solution.

Since the rate term is removed from the governing equations, the steady-state problem has no intrinsic physically meaningful time scale; nevertheless, you may assign a "time" scale to the analysis step. This time scale is often convenient for output identification and for specifying prescribed normalized concentrations and fluxes with varying magnitudes. Thus, when steady-state analysis is chosen, you specify a "time" increment and a "time" period for the step; Abaqus/Standard then increments through the step accordingly. If a steady-state analysis step is to be followed by a transient analysis step and total time is used in amplitude definitions ("Amplitude curves," Section 34.1.2), the time period should be defined to be negligibly small in the steady-state step. For more details on time scales and time stepping, see "Defining an analysis," Section 6.1.2.

Input File Usage: *MASS DIFFUSION, STEADY STATE

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Steady state**

Transient analysis

Time integration in transient diffusion analysis is done with the backward Euler method (also referred to as the modified Crank-Nicholson operator). This method is unconditionally stable for linear problems.

Automatic or fixed time incrementation can be used for transient analysis. The automatic time incrementation scheme is generally preferred because the response is usually simple diffusion: the rate of change of normalized concentration varies widely during the step and requires different time increments to maintain accuracy in the time integration.

Spurious oscillations due to small time increments

In transient mass diffusion analysis with second-order elements there is a relationship between the minimum usable time step and the element size. A simple guideline is

$$\Delta t > \frac{1}{6D}\Delta\ell^2,$$

where Δt is the time increment, D is the diffusivity, and $\Delta\ell$ is a typical element dimension (such as the length of a side of an element). If time increments smaller than this value are used, spurious oscillations can appear in the solution. Abaqus/Standard provides no check on the initial time increment defined for a mass diffusion analysis; you must ensure that the given value does not violate the above criterion.

In transient analysis using first-order elements the solubility terms are lumped, which eliminates such oscillations but can lead to locally inaccurate solutions for small time increments. If smaller time increments are required, a finer mesh should be used in regions where the normalized concentration changes occur.

Generally there is no upper limit on the time increment because the integration procedure is unconditionally stable unless nonlinearities cause numerical problems.

Automatic incrementation

The automatic time incrementation scheme for mass diffusion problems is based on the user-specified maximum normalized concentration change allowed at any node during an increment, ΔC_{max}.

Input File Usage: *MASS DIFFUSION, DCMAX=ΔC_{max}

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **Max. allowable normalized concentration change:** ΔC_{max}

Fixed time incrementation

If you choose fixed time incrementation, fixed time increments equal to the size of the user-specified initial time increment, Δt_0, will be used.

Input File Usage: *MASS DIFFUSION
Δt_0

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Fixed**, **Increment size:** Δt_0

Ending a transient analysis

Transient mass diffusion analysis can be terminated by completing a specified time period, or it can be continued until steady-state conditions are reached. By default, the analysis will end when the given time period has been completed. Alternatively, you can specify that the analysis will end when steady state is reached or the time period ends, whichever comes first. Steady state is defined as the point in time when all normalized concentrations change at less than a user-defined rate.

Input File Usage: Use the following option to end the analysis when the time period is reached:

∗MASS DIFFUSION, END=PERIOD (default)

Use the following option to end the analysis based on the concentration change rate:

∗MASS DIFFUSION, END=SS

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Mass diffusion**: **Basic**: **Response: Transient**; **Incrementation**: **Type: Automatic**: **End step when normalized concentration change rate is less than**

Initial conditions

An initial normalized concentration of the diffusing material at specific nodes that belong to mass diffusion elements can be defined (“Initial conditions in Abaqus/Standard and Abaqus/Explicit,” Section 34.2.1). For an analysis in which mass diffusion is driven by gradients of temperature and/or pressure (“Diffusivity,” Section 26.4.1), the initial temperature and pressure stress fields in a model can also be defined.

Input File Usage: Use the following options:

∗INITIAL CONDITIONS, TYPE=CONCENTRATION for initial concentrations
∗INITIAL CONDITIONS, TYPE=TEMPERATURE for initial temperatures
∗INITIAL CONDITIONS, TYPE=PRESSURE STRESS for initial equivalent pressure stress

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Temperature** for the **Types for Selected Step**

Initial concentration and equivalent pressure stress are not supported in Abaqus/CAE.

Boundary conditions

Boundary conditions can be applied to nodal degree of freedom 11 in any mass diffusion element to prescribe values of normalized concentration (“Boundary conditions in Abaqus/Standard and Abaqus/Explicit,” Section 34.3.1). Such values can be specified as functions of time.

Any boundary condition changes to be applied during a mass diffusion step should be given in the respective step using appropriate amplitude definitions to specify their "time" variations ("Amplitude curves," Section 34.1.2). If boundary conditions are specified for the step without amplitude references, they are assumed to change either linearly with "time" during the step or instantly at the start of the step, according to the user-specified or default time variation associated with the step (see "Defining an analysis," Section 6.1.2).

Loads

Concentration fluxes are the only loads that can be applied in a mass diffusion analysis step.

Input File Usage: Use the following option to specify a concentrated concentration flux at a node:

∗CFLUX
node number or node set name, *degree of freedom*, *concentrated flux magnitude*

Use the following option to specify a distributed concentration flux acting on entire elements (body flux) or just on element faces (surface flux):

∗DFLUX
element number or element set name, BF or S*n*, *distributed flux magnitude*

Abaqus/CAE Usage: Use the following input to define a concentrated concentration flux at a node:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Concentrated concentration flux** for the **Types for Selected Step**: select region: **Magnitude**: *concentrated flux magnitude*

Use the following input to define a distributed concentration flux acting on entire elements (body flux) or just on element faces (surface flux):

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Body concentration flux** or **Surface concentration flux** for the **Types for Selected Step**: **Distribution**: **Uniform** or select an analytical field, **Magnitude**: *distributed flux magnitude*

Modifying or removing concentration fluxes

Concentrated or distributed concentration fluxes can be added, modified, or removed as described in "Applying loads: overview," Section 34.4.1.

Specifying time-dependent concentration fluxes

The magnitude of a concentrated or a distributed concentration flux can be controlled by referring to an amplitude curve (see "Amplitude curves," Section 34.1.2). If different magnitude variations are needed for different fluxes, the flux definitions can be repeated, with each referring to its own amplitude curve.

Defining nonuniform distributed concentration fluxes in a user subroutine

To define nonuniform distributed concentration fluxes, the variation of the flux magnitude throughout a step can be defined in user subroutine **DFLUX**. If a reference flux magnitude is specified directly, it will be ignored. As a result, any amplitude reference in the flux definition is also ignored.

Input File Usage: Use the following option to define a nonuniform distributed concentration body flux:

∗DFLUX
element number or element set, BFNU

Use the following option to define a nonuniform distributed concentration surface flux:

∗DFLUX
element number or element set, S*n*NU

Abaqus/CAE Usage: Use the following input to define a nonuniform distributed concentration body flux:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Body concentration flux** for the **Types for Selected Step**: select region: **Distribution: User-defined**

Use the following input to define a nonuniform distributed concentration surface flux:

Load module: **Create Load**: choose **Mass diffusion** for the **Category** and **Surface concentration flux** for the **Types for Selected Step**: select region: **Distribution: User-defined**

Predefined fields

Predefined temperatures, equivalent pressure stresses, and field variables can be specified in a mass diffusion analysis.

Prescribing temperatures

Temperatures are applied to nodes in temperature-driven mass diffusion analyses by defining a temperature field; absolute zero on the temperature scale used is defined as described in "Specifying the value of absolute zero" in "Thermal loads," Section 34.4.4. Alternatively, the temperature field can be obtained from a previous heat transfer analysis. Time-dependent temperature variations are possible with either approach.

A simple interface is provided that uses the Abaqus/Standard results file from the heat transfer analysis to define the temperature field at different times in the mass diffusion analysis. Abaqus/Standard assumes that the nodes in the mass diffusion analysis have the same numbers as the nodes in the previous heat transfer analysis. Values in the results file are ignored at nodes that exist in the heat transfer analysis but not in the mass diffusion analysis, and the temperatures at nodes that did not exist in the heat transfer analysis will not be set by reading the results file.

For specific details on prescribing temperatures, see “Predefined temperature” in “Predefined fields,” Section 34.6.1.

Prescribing equivalent pressure stresses

Equivalent pressure stress values can be given at nodes by specifying them directly as a predefined field in the mass diffusion analysis or indirectly by reading the equivalent pressure stresses from the results file of a previous stress/displacement, fully coupled temperature-displacement, or fully coupled thermal-electrical-structural analysis. Regardless of the manner in which they are specified, pressures should be entered according to the Abaqus convention that equivalent pressure stresses are positive when they are compressive.

A simple interface is provided that uses the Abaqus/Standard results file from a mechanical analysis to define the equivalent pressure stresses at different times in the mass diffusion analysis. Abaqus/Standard assumes that the nodes in the mass diffusion analysis have the same numbers as the nodes in the previous mechanical analysis. Values in the results file are ignored at nodes that exist in the mechanical analysis but not in the mass diffusion analysis, and the pressures at nodes that did not exist in the mechanical analysis will not be set by reading the results file.

For specific details on prescribing equivalent pressure stresses, see “Predefined pressure stress” in “Predefined fields,” Section 34.6.1.

Specifying predefined field variables

You can specify values of predefined field variables during a mass diffusion analysis. These values affect only field-variable-dependent material properties, if any. See “Predefined field variables” in “Predefined fields,” Section 34.6.1.

Material options

Both diffusivity (“Diffusivity,” Section 26.4.1) and solubility (“Solubility,” Section 26.4.2) must be defined in a mass diffusion analysis. Optionally, a Soret effect factor and a pressure stress factor can be defined to introduce mass diffusion caused by temperature and pressure gradients, respectively. The use of Fick’s law also introduces temperature-driven mass diffusion since a Soret effect factor is calculated automatically.

Elements

Mass diffusion analysis can be performed using only the two-dimensional, three-dimensional, and axisymmetric solid elements that are included in the Abaqus/Standard heat transfer/mass diffusion element library.

Output

In addition to the standard output identifiers available in Abaqus/Standard (“Abaqus/Standard output variable identifiers,” Section 4.2.1), the following variables have special meaning in mass diffusion analyses:

Element integration point variables:

CONC	Mass concentration.
ISOL	Amount of solute at the integration point, calculated as the product of the mass concentration and the integration point volume.
MFL	Magnitude and components of the concentration flux vector (excluding the terms due to pressure and temperature gradients).
MFLM	Magnitude of the concentration flux vector.
MFL*n*	Component *n* of the concentration flux vector (*n* = 1, 2, 3).
TEMP	Magnitude of the applied temperature field.

Whole element variables:

ESOL	Amount of solute in the element, calculated as the sum of ISOL over all the element integration points.
NFLUX	Fluxes at the nodes of the element caused by mass diffusion in the element.
FLUXS	Distributed mass flux applied to an element.

Whole or partial model variables:

SOL	Amount of solute in the model or specified element set, calculated as the sum of ESOL over all the elements in the model or set.

Nodal variables:

CFL	All concentrated flux values.
CFL*n*	Concentrated flux value *n* at a node (*n* = 11).
NNC	All normalized concentration values at a node.
NNC*n*	Normalized concentration degree of freedom *n* at a node (*n* = 11).
RFL	All reaction flux values (conjugate to normalized concentration).
RFL*n*	Reaction flux value *n* at a node (*n* = 11) (conjugate to normalized concentration).

Input file template

The following template is representative of a three-step mass diffusion analysis. The first step establishes an initial steady-state concentration distribution of a diffusing material. In the second step equivalent pressure stresses are read from a fully coupled temperature-displacement analysis and the transient mass diffusion response is obtained for the case of mechanical loading of the body. In the final step a temperature field is read from a fully coupled temperature-displacement analysis and the transient mass diffusion response is calculated for the case of heating and cooling the body in which diffusion occurs. An example problem that follows this template is "Thermomechanical diffusion of hydrogen in a bending beam," Section 1.10.1 of the Abaqus Benchmarks Guide.

```
*HEADING
...
*MATERIAL,NAME=mat1
```

***SOLUBILITY**
Data lines to define solubility
***DIFFUSIVITY**
Data lines to define diffusivity
***KAPPA,TYPE=TEMP**
Data lines to define diffusion driven by temperature gradients
***KAPPA,TYPE=PRESS**
Data lines to define diffusion driven by gradients of equivalent pressure stress
***INITIAL CONDITIONS,TYPE=TEMPERATURE**
Data lines to define an initial temperature field
***INITIAL CONDITIONS,TYPE=CONCENTRATION**
Data lines to define initial nodal values of normalized concentration
***INITIAL CONDITIONS,TYPE=PRESSURE STRESS**
Data lines to define initial nodal values of equivalent pressure stress
AMPLITUDE,NAME=name*
Data lines to define amplitude variations
******
***STEP**
Step 1 - steady-state solution
***MASS DIFFUSION,STEADY STATE**
Data line to define incrementation
***BOUNDARY**
Data lines to prescribe nodal values of normalized concentration
***EL FILE**
Data lines to define element integration output to the results file
***NODE FILE**
Data lines to define nodal output to the results file
***END STEP**
******
***STEP**
Step 2 - transient analysis driven by pressure stress gradients
MASS DIFFUSION,DCMAX=dcmax***,END=SS**
Data line to define incrementation
***BOUNDARY**
Data lines to prescribe nodal values of normalized concentration
PRESSURE STRESS,FILE=name*
***EL FILE**
Data lines to define element integration output to the results file
***NODE FILE**
Data lines to define nodal output to the results file
***END STEP**
******

***STEP**
Step 3 - transient analysis driven by temperature gradients
MASS DIFFUSION,DCMAX=dcmax***,END=SS**
Data line to define incrementation
***BOUNDARY**
Data lines to prescribe nodal values of normalized concentration
TEMPERATURE,FILE=name*
***EL FILE**
Data lines to define element integration output to the results file
***NODE FILE**
Data lines to define nodal output to the results file
***END STEP**

Additional reference

- Crank, J., *The Mathematics of Diffusion,* Clarendon Press, Oxford, 1956.

6.10 Acoustic and shock analysis

- “Acoustic, shock, and coupled acoustic-structural analysis,” Section 6.10.1

6.10.1 ACOUSTIC, SHOCK, AND COUPLED ACOUSTIC-STRUCTURAL ANALYSIS

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Acoustic medium," Section 26.3.1
- "Acoustic and shock loads," Section 34.4.6
- "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1
- "ALE adaptive meshing: overview," Section 12.2.1
- "Steady-state transport analysis," Section 6.4.1
- *ACOUSTIC FLOW VELOCITY
- *ACOUSTIC WAVE FORMULATION
- *ADAPTIVE MESH
- *BEAM FLUID INERTIA
- *CONWEP CHARGE PROPERTY
- *IMPEDANCE
- *IMPEDANCE PROPERTY
- *INCIDENT WAVE
- *INCIDENT WAVE INTERACTION
- *INITIAL CONDITIONS
- *SIMPEDANCE
- *TIE
- "Defining an acoustic pressure boundary condition," Section 16.10.19 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Creating the submodel boundary condition," Section 38.4 of the Abaqus/CAE User's Guide

Overview

Analyses performed using acoustic elements, an acoustic medium, and a dynamic procedure can simulate a variety of engineering phenomena including low-amplitude wave phenomena involving fluids such as air and water and "shock" analysis involving higher amplitude waves in fluids interacting with structures.

An acoustic analysis:

- is used to model sound propagation, emission, and radiation problems;
- can include incident wave loading to model effects such as underwater explosion (UNDEX) on structures interacting with fluids, airborne blast loading on structures, or sound waves impinging on a structure;

- in Abaqus/Explicit can include fluid undergoing cavitation when the absolute pressure drops to a limit value;
- is performed using one of the dynamic analysis procedures ("Dynamic analysis procedures: overview," Section 6.3.1);
- can be used to model an acoustic medium alone, as in the study of the natural frequencies of vibration of a cavity containing an acoustic fluid;
- can be used to model a coupled acoustic-structural system, as in the study of the noise level in a vehicle;
- can be used to model the sound transmitted through a coupled acoustic-structural system;
- requires the use of acoustic elements and, for coupled acoustic-structural analysis, a surface-based interaction using a tie constraint or, in Abaqus/Standard, acoustic interface elements;
- can be used to obtain the scattered wave solution directly under incident wave loading when the mechanical behavior of the fluid is linear;
- can be used to obtain a total wave solution (sum of the incident and the scattered waves) by selecting the total wave formulation, particularly when nonlinear fluid behavior such as cavitation is present in the acoustic medium;
- can be used to model problems where the acoustic medium interacts with a structure subjected to large static deformation;
- in Abaqus/Standard can be used with symmetric model generation ("Symmetric model generation," Section 10.4.1) and symmetric results transfer ("Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh," Section 10.4.2);
- in Abaqus/Standard can be used with steady-state transport ("Steady-state transport analysis," Section 6.4.1) and an acoustic flow velocity ("*ACOUSTIC FLOW VELOCITY," Section 1.2 of the Abaqus Keywords Reference Guide) to model acoustic perturbations of a moving fluid;
- in Abaqus/Standard can include a coupled structural-acoustic substructure that was previously defined ("Defining substructures," Section 10.1.2);
- can be used to model both interior problems, where a structure surrounds one or more acoustic cavities, and exterior problems, where a structure is located in a fluid medium extending to infinity; and
- is applicable to any vibration or dynamic problem in a medium where the effects of shear stress are negligible.

A shock analysis:

- is used to model blast effects on structures;
- often requires double precision to avoid roundoff error when Abaqus/Explicit is used;
- may include acoustic elements to model the effects of fluid inertia and compressibility;
- may include virtual mass effects to model the effect of an incompressible fluid interacting with a pipe structure;
- is performed using one of the dynamic analysis procedures ("Dynamic analysis procedures: overview," Section 6.3.1);

- can be used to model both interior problems, where a structure surrounds one or more fluid cavities, and exterior problems, where a structure is located in a fluid medium extending to infinity; and
- in Abaqus/Explicit can include air blast loading on structures using the CONWEP model.

Procedures available for acoustic analysis

Acoustic elements model the propagation of acoustic waves and are active only in dynamic analysis procedures. They are most commonly used in the following procedures:

- Direct solution, steady-state, harmonic analysis. See "Direct-solution steady-state dynamic analysis," Section 6.3.4.
- Frequency analysis. See "Natural frequency extraction," Section 6.3.5.
- Subspace-based steady-state dynamic analysis. See "Subspace-based steady-state dynamic analysis," Section 6.3.9.
- Explicit dynamic analysis. See "Explicit dynamic analysis," Section 6.3.3.

Acoustic analysis can also be performed using:

- Direct time integration analysis. See "Implicit dynamic analysis using direct integration," Section 6.3.2.
- Complex frequency analysis. See "Natural frequency extraction," Section 6.3.5.
- Mode-based transient dynamic analysis. See "Transient modal dynamic analysis," Section 6.3.7.
- Mode-based steady-state dynamic analysis. See "Mode-based steady-state dynamic analysis," Section 6.3.8.
- Dynamic fully coupled temperature-displacement analysis. See "Fully coupled thermal-stress analysis," Section 6.5.3.

In general, analysis with acoustic elements should be thought of as small-displacement linear perturbation analysis, in which the strain in the acoustic elements is strictly (or overwhelmingly) volumetric and small. In many applications the base state for the linear perturbation is simply ignored: for solid structures interacting with air or water, the initial stress (if any) in the air or water has negligible physical effect on the acoustic waves. Most engineering acoustic analyses, transient or steady state, are of this type.

An important exception is when the acoustic perturbation occurs in a gas or liquid with high-speed underlying flow. If the magnitude of the flow velocity is significant compared to the speed of sound in the fluid (i.e., the Mach number is much greater than zero), the propagation of waves is facilitated in the direction of flow and impeded in the direction against the flow. This phenomenon is the source of the well-known "Doppler effect." In Abaqus/Standard underlying flow effects are prescribed for nodes making up acoustic elements by specifying an acoustic flow velocity.

Acoustic elements can be used in a static analysis, but all acoustic effects will be ignored. A typical example is the air cavity in a tire/wheel assembly. In such a simulation the tire is subjected to inflation, rim mounting, and footprint loads prior to the coupled acoustic-structural analysis in which the acoustic response of the air cavity is determined. See "Defining ALE adaptive mesh domains in

Abaqus/Standard," Section 12.2.6, and "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7, for more information.

Acoustic elements also can be used in a substructure generation procedure to generate coupled structural-acoustic substructures. Only structural degrees of freedom can be retained. The retained eigenmodes must be selected when an acoustic-structural substructure is generated. In a static analysis involving a substructure containing acoustic elements, the results will differ from the results obtained in an equivalent static analysis without substructures. The reason is that the acoustic-structural coupling is taken into account in the substructure (leading to hydrostatic contributions of the acoustic fluid), while the coupling is ignored in a static analysis without substructures. More details on coupled structural-acoustic substructures can be found in "Defining substructures," Section 10.1.2.

A volumetric drag coefficient, γ, can be defined to simulate fluid velocity-dependent pressure amplitude losses. These occur, for example, when the acoustic medium flows through a porous matrix that causes some resistance (see "Acoustic medium," Section 26.3.1), such as a sound-deadening material like fiberglass insulation. For direct time integration dynamic analysis we assume there are no significant spatial discontinuities in the quantity γ/ρ_f, where ρ_f is the density of the fluid (acoustic medium), and that the volumetric drag is small at acoustic-structural boundaries. These assumptions, which can limit the applicability of the analysis, are discussed further in "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide.

The direct-solution steady-state dynamic harmonic response procedure is advantageous for acoustic-structural sound propagation problems, because the gradient of γ/ρ_f need not be small and because acoustic-structural coupling and damping are not restricted in this formulation. If there is no damping or if damping can be neglected, factoring a real-only matrix can reduce computational time significantly; see "Direct-solution steady-state dynamic analysis," Section 6.3.4, for details.

Some fluid-solid interaction analyses involve long-duration dynamic effects that more closely resemble structural dynamic analysis than wave propagation; that is, the important dynamics of the structure occur at a time scale that is long compared to the compressional wave speed of the solid medium and the acoustic wave speed of the fluid. Equivalently, in such cases, disturbances of interest in the structure propagate very slowly in comparison to waves in the fluid and compressional waves in the structure. In such instances, modeling of the structure using beams is common. When these structural elements interact with a surrounding fluid, the important fluid effect is due to motions associated with incompressible flow (see "Loading due to an incident dilatational wave field," Section 6.3.1 of the Abaqus Theory Guide). These motions result in a perceived inertia added to the structural beam; therefore, this case is usually referred to as the "virtual mass approximation." For this case Abaqus allows you to modify the inertia properties of beam and pipe elements, as described below. Loads on the structure associated with incident waves in the fluid can be accommodated under this approximation as well.

Natural frequency extraction

Abaqus can compute both real and complex eigensolutions for purely acoustic or structural-acoustic systems, with or without damping. Exterior acoustic problems may also be solved.

Selecting an eigensolver

In a coupled acoustic-structural model, real-valued coupled modes are extracted by default using the Lanczos eigenfrequency extraction procedure. Coupling may be suppressed in the frequency extraction step; in this case the structural elements behave as though the interface with the acoustic elements were free (as though this surface were "in vacuo"), and the acoustic elements behave as though the boundary with the structural elements were rigid. Extracting the coupled acoustic-structural modes is also available for the AMS eigensolver.

Structural-acoustic coupling is ignored if the subspace iteration eigensolver is used.

When applying the AMS eigensolver to a coupled structural-acoustic model, Abaqus by default projects and stores the acoustic coupling matrix during the natural frequency extraction, for later use in coupled forced response analyses. The structural and acoustic regions are not actually coupled during the eigenanalysis; Abaqus solves the two regions separately but computes and stores the projected coupling operator for use in subsequent steady-state dynamic steps. Only structural-acoustic coupling defined using tied contact is supported. You can suppress this coupling if desired. Damping due to acoustic volumetric drag is also projected by default during an eigenanalysis and is restored by default in subsequent steady-state dynamic steps. Projecting and storing the acoustic coupling matrix during the natural frequency extraction is also available for the Lanczos eigensolver based on the SIM architecture.

Damping and inertia effects in an acoustic natural frequency extraction

Since damping is not taken into account in real-valued modal extraction, the volumetric drag effect is not considered, except for its small contribution to any nonreflecting boundaries (see "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide). The damping contributions due to any impedance boundary conditions (element-based or surface-based) or acoustic infinite elements are not included in an eigenfrequency extraction step, but the contributions to the acoustic element mass and stiffness matrices are included. Similarly, the (symmetrized) stiffness and mass contributions of acoustic infinite elements are included in an eigenfrequency extraction step, but the damping effects are neglected.

Modal analysis of damped and radiating acoustic systems can be performed in Abaqus as well. Using the complex eigenvalue extraction procedure, the damping contributions of acoustic infinite elements, nonreflecting impedance conditions, and general impedance layers are restored to the element operators.

If an underlying flow field is defined for the acoustic region by specifying an acoustic flow velocity, the natural frequencies and modes are affected. However, in real-valued frequency extraction only the acoustic element mass and stiffness matrices contribute to the solution. Since the formulation for acoustics in the presence of a flow field requires a complex part in the element operator (damping matrix), the real-valued procedure can include the effects of flow only to a limited degree. The complex frequency procedure in Abaqus/Standard includes the damping matrix contribution and is, therefore, required when modes of a system with moving fluid are sought. The complex frequency procedure can be used only following the Lanczos real-valued frequency procedure.

Virtual mass effects defined for beams by adding inertia ("Additional inertia due to immersion in fluid" in "Beam section behavior," Section 29.3.5) are included in modal analysis: their effect is simply to add inertia to a beam element.

Interpreting the extracted modes in a coupled structural-acoustic natural frequency analysis

While all the modes extracted in a coupled Lanczos structural-acoustic natural frequency analysis include the effects of fluid-solid interaction, some of them may have predominantly structural contributions while others may have predominantly acoustic contributions. Coupled structural-acoustic eigenmodes can be categorized as follows:

- Most generally, an individual mode may exhibit participation in both the fluid and the solid media; this is referred to as a "coupled mode."
- Second, there are the "structural resonance" modes. These are modes corresponding to the eigenmodes of the structure without the presence of the acoustic fluid. The presence of the acoustic fluid has a relatively small effect on these eigenfrequencies and the mode shapes.
- Third, there are the "acoustic cavity resonance" modes. These are nonzero eigenfrequency coupled modes that have a significant contribution in the resulting dynamics of the acoustic pressure in mode-based dynamic procedures.
- Fourth, if insufficient boundary conditions are specified on the structural part of a model, the frequency extraction procedure will extract rigid body modes. These modes have zero eigenfrequencies (sometimes they appear as either small positive or even negative eigenvalues). However, if sufficient structural degrees of freedom are constrained, these rigid body modes disappear.
- Finally, there are the singular acoustic modes, which have zero eigenfrequencies and constant acoustic pressure; they are mathematically analogous to structural rigid body modes. The structural part of the singular acoustic modes corresponds to the quasi-static structural response to constant pressure in unconstrained acoustic regions. These eigenmodes are predominantly acoustic and are important in representing the (low-frequency) acoustic response in mode-based analysis in the presence of acoustic loads, in the same way that rigid body modes are important in the representation of structural motion. As is true for the structural rigid body modes, if a sufficient number of constrained acoustic degrees of freedom is specified (one degree of freedom 8 per acoustic region is enough), the singular acoustic modes will disappear. In models with only one unconstrained acoustic region (the most common case) only one singular acoustic mode will be computed. In general there are as many singular acoustic modes as there are independent unconstrained acoustic regions. If these modes are present, they are always reported first by the Lanczos eigensolver; and a note at the bottom of the eigenfrequency table in the data file provides information about the number of singular acoustic modes.

The generalized masses and effective masses can help distinguish between the various types of modes and can be used to assess which modes are important for subsequent mode-based analyses. In addition, the acoustic contribution to the generalized masses is reported as a fraction for each eigenmode. The closer the value of this fraction is to unity, the more pronounced is the acoustic component of this eigenmode. An acoustic effective mass is also computed for each eigenmode. This scalar quantity is scaled such that when all eigenmodes in a model are extracted, the sum of all acoustic effective masses is equal to 1.0 (minus the contributions from nodes with restrained acoustic degrees of freedom). The acoustic effective mass can be compared between different modes: the higher the acoustic effective

mass, the more important (typically) the mode is for accurate representation of the acoustic pressure. For example, the fluid cavity acoustic resonance modes will have larger acoustic effective masses compared to the other modes.

Modal superposition procedures

In Abaqus acoustic domains are handled quite similarly to solid and structural domains. Real-valued eigenmodes, resulting from a previous real-valued eigenfrequency extraction procedure with or without coupling effects included, are used as a basis for modal solutions. The mode-based steady-state dynamic procedure is the most computationally efficient alternative to compute the steady-state response of structural-acoustic systems. Structural-acoustic coupling and damping effects in these analyses depend on the type of modal procedure and the eigensolver that was used to compute the eigenfrequencies.

Structural-acoustic coupling in modal analyses using the Lanczos eigensolver without the SIM architecture

If coupled modes are computed using the Lanczos eigensolver, both the mode-based and subspace projection steady-state dynamic procedures will include structural-acoustic coupled effects. If uncoupled Lanczos modes are computed, coupling can be restored only by using subspace projection. It is sufficient to project at a single frequency (constant subspace) to resolve the acoustic coupling for all frequencies.

Acoustic medium damping in modal analyses using the Lanczos eigensolver without the SIM architecture

In subspace-based steady-state dynamic analysis, acoustic medium damping and structural material damping are considered, and the structural-acoustic interaction, infinite element, and impedance boundary terms are also included.

Acoustic medium damping is not considered in the procedures that base the response prediction directly on the system's eigenmodes, such as transient modal dynamic analysis or the mode-based steady-state dynamic procedure. These methods should, therefore, be used with caution for problems with impedance boundary conditions. Modal damping can be used in these procedures ("Material damping," Section 26.1.1) to model material damping and volumetric drag effects; however, modal damping usually cannot be used to model the fluid-solid coupling or the impedance boundary effects accurately.

Structural-acoustic coupling and damping in modal analyses using the subspace iteration eigensolver

The subspace iteration eigensolver neglects the effects of structural-acoustic coupling; therefore, coupling effects are not included in subsequent modal procedures.

As with analyses using the Lanczos eigensolver, acoustic medium damping and structural material damping are considered in subsequent subspace-based steady-state dynamic procedures, but these damping effects are not considered in subsequent transient modal or mode-based steady-state dynamic procedures.

Structural-acoustic coupling and damping in modal analyses using the AMS eigensolver or the Lanczos eigensolver based on the SIM architecture

The structural-acoustic modes, extracted using the AMS eigensolver or the Lanczos eigensolver, can be used in modal analyses using the SIM architecture. When uncoupled modes are computed using the AMS eigensolver or the Lanczos eigensolver based on the SIM architecture with projection of the structural-acoustic coupling specified, the coupling and acoustic damping operators are projected and stored during the natural frequency extraction. Subsequent coupled forced response analyses using modal steady-state dynamics automatically restore the effects of structural-acoustic coupling and damping by automatically using these projected matrices; if the matrices were not projected, the steady-state dynamic step would not include these effects. A mode-based steady-state dynamic step cannot use unsymmetric damping, such as from acoustic flow velocity or infinite element effects. To take these effects into account, a subspace-based steady-state dynamic analysis should be used.

Defining translational or rotational underlying flow velocity in Abaqus/Standard

As described above, acoustic analysis in Abaqus/Standard can be performed as a linear perturbation of a high-speed flow field. The flow velocity field affects the propagation of acoustic waves in the medium through the effect of the flow velocity on the speed of the wave propagation. Waves travel faster along the direction of the local flow vector and are correspondingly impeded in the direction against the flow direction. It is sufficient for you to define the velocity field in the affected acoustic region; the accelerations do not play a role in the formulation.

You specify the flow in the acoustic finite element region as history data within a dynamic linear perturbation step. The flow field can be described either by direct input of the velocity components or by defining rotating motion associated with a reference frame. In the former case, each node in the acoustic region where flow occurs is assigned a Cartesian velocity defined by specifying the components of the velocity vector, $\mathbf{c}$. In the latter case, the rotational velocity for the nodes in the acoustic region is defined by specifying the magnitude of an angular rotation velocity, Ω, and the position and orientation of the axis of rotation in the current configuration. The position and orientation of the axis are applied at the beginning of the step and remain fixed during the step.

The specified underlying flow is active only for acoustic finite elements; other elements with acoustic degrees of freedom, such as acoustic infinite and interface elements, are unaffected by the specified flow velocity. The effect of underlying flow on the acoustic finite elements depends also on the procedure used: the effects are present only in frequency-domain dynamic procedures and natural frequency extraction. For complex-valued procedures, such as complex frequency extraction and steady-state dynamics, the presence of underlying flow affects the acoustic finite element stiffness matrices and adds a significant contribution to the element damping matrix. For real-valued procedures, such as eigenfrequency extraction and steady-state dynamics analysis in which a real-only system matrix is factored, the presence of underlying flow affects only the acoustic finite element stiffness matrices; the damping matrix is ignored. Consequently, the effect of flow on the acoustic field is fully realized only in complex-valued procedures.

For rotating systems, solid and acoustic materials are treated differently in Abaqus. Flow of solid material through a mesh may induce significant deformation and is handled by using steady-state

transport; subsequent linear perturbation steps are analyzed about this deformed state (see "Steady-state transport analysis," Section 6.4.1). Flow of material through an acoustic mesh is handled entirely within linear perturbation steps by specifying an acoustic flow velocity; a preliminary nonlinear steady-state transport analysis is not required. For coupled acoustic-structural systems undergoing rotation, such as tires, the model may be subjected to a steady-state transport step, which deforms the solid medium, followed by linear perturbation dynamic steps. The effect of the rotation of the solid is included in the linear perturbation steps in this case; to include the effect of the rotation of the acoustic fluid, specify an acoustic flow velocity in the linear perturbation steps.

Input File Usage: Use the following option to define a translating fluid velocity:

*ACOUSTIC FLOW VELOCITY, TRANSLATION

Use the following option to define a rotating fluid velocity:

*ACOUSTIC FLOW VELOCITY, ROTATION

Abaqus/CAE Usage: Acoustic flow velocity is not supported in Abaqus/CAE.

Updating the acoustic domain during a large-displacement Abaqus/Standard analysis

By default, the acoustic-structural coupling calculations are based on the original configuration of the fluid domain. However, acoustic elements can also be used in an analysis where the structural domain experiences large deformation prior to the coupled analysis. A typical example is the interior cavity of a tire subjected to structural loads such as inflation, rim mounting, and footprint pressure.

The acoustic elements in Abaqus do not have displacement degrees of freedom and, therefore, cannot model the deformation of the fluid when the structure undergoes large deformation. Abaqus/Standard solves the problem of computing the current configuration of the acoustic domain by periodically creating a new acoustic mesh. The new mesh uses the same topology (elements and connectivity) throughout the simulation, but the nodal locations are adjusted so that the acoustic domain conforms to the structural domain on the boundary.

A new acoustic mesh is computed when adaptive meshing is specified and nonlinear geometric effects are considered in any non-perturbation Abaqus/Standard analysis procedure in which acoustic effects are ignored.

The adaptive meshing features for acoustic analysis are described in detail in "Defining ALE adaptive mesh domains in Abaqus/Standard," Section 12.2.6, and "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7.

Initial conditions

In Abaqus/Standard the initial acoustic static pressure (hydrostatic or ambient) is not modeled by the acoustic formulation and cannot be specified as an initial condition.

In Abaqus/Explicit the initial acoustic pressure corresponding to the initial static equilibrium (hydrostatic or ambient) can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) and is meaningful only when the acoustic fluid is capable of undergoing cavitation. In problems with possible fluid cavitation the initial acoustic static pressure is taken into account in the cavitation condition; i.e., the sum of the dynamic and static acoustic pressures

needs to drop to the cavitation pressure limit for the cavitation to occur. The specified acoustic static pressure is used only in the cavitation condition and does not apply any static loads to the acoustic or structural meshes at their common wetted interface. In addition, the acoustic static pressure is not included in the nodal acoustic pressure degree of freedom.

The initial temperature and field variable values can be specified ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) for the direct time integration dynamic, explicit dynamic, dynamic fully coupled temperature-displacement, and mode-based transient dynamic analysis procedures. Changes in these variables during the analysis will affect any temperature-dependent or field-variable-dependent acoustic medium properties.

Boundary conditions

The various boundary conditions that can be applied to an acoustic medium are described below. These include acoustic domain boundaries with stationary rigid walls or symmetry planes, prescribed pressure values such as a free surface with zero dynamic pressure, specified impedance (see "Acoustic and shock loads," Section 34.4.6), and structural interfaces such as the interface with a ship or a submarine. The radiating (nonreflecting) boundary condition for exterior problems (such as a structure vibrating in an acoustic medium of infinite extent) is implemented as a special case of the impedance boundary condition (see "Acoustic and shock loads," Section 34.4.6). On any given part of the acoustic domain boundary only one boundary condition type should be applied, except for the combination of the impedance boundary condition and the acoustic-structural interface condition.

Boundary with a stationary rigid wall or a symmetry plane

The default boundary condition for an acoustic medium is a boundary with a stationary rigid wall or a symmetry plane. The "force" conjugate to pressure in the acoustics formulation in Abaqus is the normal pressure gradient at the surface divided by the mass density; dimensionally this is equal to a force per unit mass. In the absence of volumetric drag this force per unit mass is equal to the inward acceleration of the acoustic medium. The conjugate variable at a node on the surface is the inward volume acceleration, which is the integral of the inward acceleration of the acoustic medium evaluated over the surface area associated with the node. A "traction-free" surface (one with no boundary conditions, no applied loads, no surface impedance conditions, and no interface elements) is a surface normal to which the acoustic medium undergoes no motion and, thus, corresponds to a rigid, stationary surface adjacent to the fluid. A symmetry plane for the acoustic medium is another "traction-free" surface.

Prescribed pressure

The basic variable in the acoustic medium is pressure (degree of freedom 8). Therefore, this variable can be prescribed at any node in the acoustic model by applying a boundary condition ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). Setting the pressure to zero represents a "free surface," where the pressure does not vary because of the motion of the surface (to account for surface motion effects, see the discussion of impedance below). Prescribing a nonzero value for the pressure represents a sound source.

An amplitude variation can be used to specify the value of the pressure. In a steady-state analysis you can specify both the in-phase (real) part of the pressure (default) and the out-of-phase (imaginary) part of the pressure.

Input File Usage: Use either of the following options to define the real (in-phase) part of the boundary condition:

*BOUNDARY
*BOUNDARY, REAL

Use the following option to define the imaginary (out-of-phase) part of the boundary condition:

*BOUNDARY, IMAGINARY

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Acoustic pressure** for the **Types for Selected Step**: select regions: **Magnitude**: *real (in-phase) part* **+** *imaginary (out-of-phase) part* **i**

Boundary with a structure

If the acoustic medium is adjacent to a structure, there will be a transfer of momentum and energy between the media at the boundary. The pressure field modeled with acoustic elements creates a normal surface traction on the structure, and the acceleration field modeled with structural elements creates the natural forcing term at the fluid boundary (for details, see "Coupled acoustic-structural medium analysis," Section 2.9.1 of the Abaqus Theory Guide).

The surface-based coupling procedure and the user-defined acoustic interface elements differ slightly in their theoretical implementation. In essence, the interface elements computed internally by the surface-based procedure are discrete point elements computed at the nodes of the slave surface. A user-defined acoustic interface element, on the other hand, distributes coupling effects across all of its nodes. Generally, the results obtained using the two coupling methods will be very similar, but the difference in discretization at the coupling boundary may create small differences in results.

Defining acoustic-structural coupling with user-defined acoustic interface elements

In Abaqus/Standard, if the structural and acoustic meshes share nodes at the boundary, lining this boundary with acoustic-structural interface elements (see "Acoustic interface element library," Section 32.13.2) will enforce the required physical coupling condition. The interface element normals must point into the acoustic medium, which forces continuity of the normal accelerations of the acoustic medium and of the structure at the boundary and ensures that the pressure of the acoustic elements is applied to the structure. Displacements can also be prescribed at such a boundary.

Defining acoustic-structural coupling using a surface-based coupling procedure

Alternatively, a surface-based procedure can be used to enforce the coupling; in Abaqus/Explicit the surface-based procedure is the only available method. This method requires that the structural and acoustic meshes use separate nodes. You define surfaces on the structural and fluid meshes and define

the interaction between the two meshes using a surface-based tie constraint (see "Mesh tie constraints," Section 35.3.1). No additional element definitions are required.

The slave surface, the first of the two surfaces specified for the tie constraint, must be element-based; whereas the master surface can be either element- or node-based. A surface based on rigid element types (R3D4, etc.) or an analytical rigid surface cannot be used as a master surface; instead, use a deformable surface made rigid.

For surface-based tie constraints Abaqus automatically computes the region of influence for each internally generated acoustic-structural interface element. If the user-defined slave surface overhangs the master surface significantly, the regions of influence may include parts of the overhang. Consequently, the overhanging part of the slave surface may exhibit nonphysical coupled degrees of freedom: displacements if the slave surface is acoustic and acoustic pressures if the slave surface is solid or structural. These nonphysical results on the overhang do not affect the remainder of the solution, and it should be understood that they are not meaningful.

Input File Usage: Use the following option in an analysis with the fluid mesh surface as the slave:

```
*TIE, NAME=fluidslave
fluid_surf, struct_surface
```

Use the following option in an analysis with the solid mesh surface as the slave:

```
*TIE, NAME=solidslave
struct_surf, fluid_surf
```

Abaqus/CAE Usage: Interaction module: **Create Constraint**: **Tie**

Coupling surfaces to structures using acoustic infinite elements

Acoustic infinite elements may form surfaces that can be coupled to structures by using a tie constraint in two different ways. The acoustic infinite element surface may consist of the base (first) facets of the acoustic infinite elements; in this case this surface should be tied to a topologically similar structural surface. The acoustic infinite element edges may also be used to define surfaces (see "Mesh tie constraints," Section 35.3.1), which can be tied to solid elements. This approach couples the semi-infinite sides of acoustic infinite elements to solid elements.

Mesh refinement

Although the meshes may be nodally nonconforming at the tied surfaces, mesh refinement affects the accuracy of the coupled solution. In acoustic-solid problems the mesh refinement depends on the wave speeds in the two media. The mesh for the medium with the lower wave speed should generally be more refined and, therefore, should be the slave surface. If the details of the wave field in the vicinity of the fluid-solid interface are important, the meshes should be of equally high refinement, with the refinement corresponding to the lower wave speed medium. In this case the choice of the master surface is arbitrary. An exception is the case where the acoustic medium must be updated to follow the structure during a large-deformation analysis. In such a case the slave surface must be defined on the acoustic domain. Another exception is the case of fluids coupled to both sides of shell or beam elements (as described below).

Solving the structural system sequentially using the submodeling technique

In some applications the normal surface traction on the structure created by the acoustic fluid may be negligible compared to other forces in the structural system. For example, a metal motor housing may radiate sound into the surrounding air, but the reaction pressure of the air on the motor may be insignificant to the dynamics of the housing. In these cases the submodeling technique (see "Submodeling: overview," Section 10.2.1) can be used to solve the system sequentially; that is, the structural analysis (uncoupled from the fluid) is followed by the acoustic analysis (driven by the structure). Usually, this decoupling of the analysis reduces computational cost. The structural system plays the role of the "global" model, and the acoustic fluid is the submodel. The structural displacements on the boundary of the acoustic fluid must be saved to the results file in the global analysis. Since Abaqus interpolates the fields between the global and submodels, acoustic-structural interface elements can be used. They should be applied to the fluid boundary to be driven by the global structural model.

Input File Usage: Use the following options in the global (structural) analysis to be followed by a submodeling analysis:

```
*NSET, NSET=solid_boundary_driving_the_fluid
*NODE FILE, NSET=solid_boundary_driving_the_fluid
U
```

Use the following options in the subsequent submodeling (fluid) analysis that uses acoustic interface elements on the fluid boundary to be driven:

```
*NSET, NSET=fluid_boundary_to_be_driven
*SUBMODEL, EXTERIOR TOLERANCE=tolerance
fluid_boundary_to_be_driven
*BOUNDARY, SUBMODEL, STEP=1
fluid_boundary_to_be_driven, 1, 3,
```

Abaqus/CAE Usage: Use the following input in the submodeling (fluid) analysis that uses an acoustic interface on the fluid boundary to be driven:

Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select regions for *fluid_boundary_to _be_driven*: **Exterior tolerance**: **relative**: *tolerance*; **Degrees of freedom**: 1, 3; **Global step number**: 1

Defining acoustic-structural coupling on both sides of a beam or shell

In Abaqus/Standard there are two alternatives available for modeling a beam (in two dimensions) or shell interacting with fluid on both sides: a surface-based procedure and an element-based procedure. In Abaqus/Explicit the surface-based procedure must be used.

Use of the surface-based procedure is straightforward. Two surfaces must be defined on the beam or shell: one on the SPOS side and one on the SNEG side. Each surface is then coupled to the fluid using a tie constraint. At least one of the two surfaces on the beam or shell must be a master surface.

In Abaqus/Standard, if the same nodes are used for the fluid and the beam or shell, acoustic interface elements must be used in the following manner to define acoustic-structural coupling on both sides of a beam or shell element:

1. Define a second set of nodes coincident with the beam or shell nodes, and constrain the motions of the two sets of nodes together using a PIN-type MPC ("General multi-point constraints," Section 35.2.2).
2. Use the first set of nodes to line one side of the beam or shell elements with acoustic interface elements (with the normals of the acoustic interface elements pointing into the fluid).
3. Use the second set of nodes to line the other side of the beam or shell elements with acoustic interface elements (with the normals pointing into the fluid on the opposite side of the structure, as in Step 2).
4. The acoustic elements on the first side of the beam or shell elements should be defined using the first set of nodes, and the acoustic elements on the second side of the beam or shell elements should be defined using the second set of nodes.

Defining the virtual mass effect (fluid-structural coupling) for beam and pipe elements

In Abaqus virtual mass effects on submerged Timoshenko beam elements can be modeled by specifying additional inertia for the beam. The virtual mass effects are specified as part of the section definition of the beam.

1. Define the beam section ("Using a beam section integrated during the analysis to define the section behavior," Section 29.3.6, or "Using a general beam section to define the section behavior," Section 29.3.7), any additional internal inertia ("Adding inertia to the beam section behavior for Timoshenko beams" in "Beam section behavior," Section 29.3.5), and the beam material properties.
2. Define the virtual mass effect ("Additional inertia due to immersion in fluid" in "Beam section behavior," Section 29.3.5).
3. If the model is to be loaded using an incident wave ("Incident wave loading due to external sources" in "Acoustic and shock loads," Section 34.4.6), define a surface or surfaces on the beam elements.

Loads

The following types of loading can be prescribed in an acoustic analysis, as described in "Acoustic and shock loads," Section 34.4.6:

- Concentrated pressure-conjugate loading.
- An impedance condition that specifies the relationship between the pressure of the acoustic medium and the normal motion at the boundary (either element-based or surface-based). Such a condition is applied, for example, to include the effect of small-amplitude "sloshing" in a gravity field or to include the effect of a compressible, possibly dissipative, lining (such as a carpet) between the acoustic medium and a fixed, rigid wall or a structure. This type of condition can also be applied to edge facets of acoustic infinite elements.
- Nonreflecting radiation conditions on acoustic boundaries (either element-based or surface-based). An impedance can be defined to select the appropriate radiating boundary condition taking the radiating surface shape into consideration.

- Incident wave loading such as that caused by an underwater explosion or a sound field. Since this type of loading is usually applied in conjunction with semi-infinite acoustic regions, two alternative modeling formulations are available in Abaqus. A total pressure-based formulation is provided when the incident wave loading is applied to the exterior of a semi-infinite acoustic mesh. This formulation must be used to handle the incident wave loading when the acoustic medium is capable of cavitation, rendering the fluid material behavior nonlinear. The scattered pressure formulation is typically used when cavitation is not part of the fluid mechanical behavior and when the loads are applied to fluid-solid interfaces. Sound transmission loss and acoustic scattering problems usually fall into the latter category.

  For both formulations, when incident wave loading is applied to a given surface, a mathematical jump occurs between the pressures on both sides of the surface because the side from which the incident pressure arrives is implicitly a region of scattered pressure. This jump is handled automatically when the incident wave load is applied to a surface with a nonreflecting impedance condition and when the incident wave load is applied to a fluid-solid interface. However, if the incident wave load is applied to a surface lying between acoustic finite or infinite elements, the jump will not be modeled properly because pressures are continuous between acoustic elements. For this case, low-mass and low-stiffness membrane, shell, or surface elements should be interposed between the acoustic elements to permit the jump in pressure to exist.

  Incident wave loading can be applied in time-harmonic problems, using the direct solution steady-state dynamics and the subspace-based, steady-state dynamic procedures. You can define individual spherical or planar sources emitting waves, or you can use the deterministic diffuse field model in Abaqus. In the former case, usage is quite similar to transient analysis: the defined sources correspond to distinct sound sources. The latter case models the sound field incident on a surface exposed to a reverberant chamber: the field is assumed to be equivalent to a number of plane waves arriving from directions distributed on a hemisphere. Only the scattered wave formulation is supported when using incident wave loading in steady-state dynamics.

  See “Acoustic and shock loads,” Section 34.4.6, for several examples of incident wave loading.

- Loading due to an incident shock wave caused by an air explosion. Although this type of wave is highly nonlinear and complex, the pressure loading due to the shock wave can be calculated readily from empirical data provided by the CONWEP model available in Abaqus/Explicit. The main advantage of this model is that the loading is applied directly to the structure subject to the blast; there is no need to include the fluid medium in the computational domain. In the CONWEP model, empirical data for two types of waves are available: spherical waves for explosions in mid-air and hemispherical waves for explosions at ground level in which ground effects are included.

  The CONWEP model does not account for effects of shadowing by intervening objects. In addition, it does not account for effects due to confinement and, thereby, excludes incorporation of any reflecting surfaces in the analysis. The model does account for the angle of incident of the blast wave; see “Acoustic and shock loads,” Section 34.4.6, for incorporation of the incident angle in the pressure load calculation.

Predefined fields

The following predefined fields can be specified in an acoustic analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in acoustic elements, nodal temperatures can be specified. The specified temperature affects temperature-dependent material properties.
- The values of user-defined field variables can be specified. These values affect field-variable-dependent material properties.

Material options

Only the acoustic medium material model ("Acoustic medium," Section 26.3.1) is valid for use in an acoustic analysis. The structure in a coupled acoustic-structural analysis can be modeled using any material model. Since acoustic analyses are always performed using a dynamic procedure, the structure's density ("Density," Section 21.2.1) should usually be defined.

Porous materials are often modeled using an acoustic formulation when the dilatational waves in the porous medium dominate the shear effects. A large number of models exist for this category of phenomenon. In Abaqus, two categories of models are available for porous media modeled with acoustic elements: phenomenological models and general frequency-dependent models. Phenomenological models describe the dynamic characteristics using material data related to the porous structure, such as porosity itself, tortuosity, etc. Alternatively, you can specify the dynamic properties directly for the material; usually, this is done using a table of frequency-dependent data. See "Acoustic medium," Section 26.3.1, for details on specifying acoustic materials in Abaqus.

When the acoustic medium is capable of cavitation and the analysis includes incident wave loading, a total pressure-based formulation must be used. Either the default scattered wave formulation or the total wave formulation can be used if the cavitation is absent or the problem has no incident wave loading.

For beam elements using the virtual mass approximation, the relevant data are specified as part of the beam section definition.

Elements

Abaqus provides a set of elements for modeling an acoustic medium undergoing small pressure changes. In addition, Abaqus/Standard provides interface elements to couple these acoustic elements to a structural model (see "Choosing the appropriate element for an analysis type," Section 27.1.3). If interface elements are used, only direct-solution steady-state harmonic (linear) response analysis ("Direct-solution steady-state dynamic analysis," Section 6.3.4) and transient response analysis ("Implicit dynamic analysis using direct integration," Section 6.3.2) can be performed.

In Abaqus/Standard the second-order acoustic elements are generally considerably more accurate than first-order acoustic elements for a given number of degrees of freedom. The acoustic elements in Abaqus/Explicit are limited to first-order interpolations.

Acoustic elements cannot be used together with hydrostatic fluid elements.

With the CONWEP model provided in Abaqus/Explicit, the analysis must be three-dimensional. The loading surface must be comprised of solid, shell, or membrane elements only. In addition, CONWEP loading cannot be applied to acoustic elements.

Exterior problems

We often need to model an exterior problem, such as a structure vibrating in an acoustic medium of infinite extent. Impedance-type radiation boundary conditions can be used to model the motions of waves out of the mesh. Abaqus provides acoustic infinite elements for this class of problems. In addition, Abaqus/Standard provides perfectly matched layers to truncate the acoustic infinite domain in a direct-solution steady-state dynamic analysis.

Impedance-type radiation conditions

In this case acoustic elements are used to model the region between the structure and a simple geometric surface (located away from the structure), and a radiating (nonreflecting) boundary condition is applied at that surface. The radiating boundary conditions are approximate, so that the error in an exterior acoustic analysis is controlled not only by the usual finite element discretization error but also by the error in the approximate radiation condition. In Abaqus the radiation boundary conditions converge to the exact condition in the limit as they become infinitely distant from the radiating structure. In practice, these radiation conditions provide accurate results when the distance between the surface and the structure is at least one-half of the longest characteristic or responsive structural wavelength.

For details, see "Acoustic and shock loads," Section 34.4.6.

Acoustic infinite elements

Acoustic infinite elements are provided for modeling exterior problems ("Infinite elements," Section 28.3.1). These elements have surface topology: line and quadratic segments in two-dimensional and axisymmetric problems and triangles and quadrilaterals in three-dimensional problems. Generally, the acoustic infinite elements are defined on a terminating surface of a region of acoustic finite elements. The infinite element formulation is considerably more accurate than the impedance-type radiation boundary conditions in cases where the wave field impinging on the terminating surface has many complex features. The radiation boundary conditions are relatively simple, equivalent to a "zeroth-order" infinite element; the acoustic infinite elements implemented in Abaqus are of variable order, up to ninth.

Acoustic infinite elements can be coupled directly to structural surfaces by using a surface-based tie constraint: this may provide adequate accuracy in some applications. In general cases the acoustic infinite elements are defined on the terminating surface of the acoustic finite element mesh. The diameter of the acoustic finite element mesh can be considerably smaller, for a given solution accuracy, than is the case when using radiation boundary conditions.

The nodal connectivity on the acoustic infinite element defines the element's surface topology. To complete the element formulation, the surface topology must be mapped into the infinite domain. This mapping requires a reference point, given in the element section property definition. The reference point serves to define a characteristic length used in the coordinate mapping. In the ideal case of acoustic radiation from a spherical surface, the correct placement of the reference point is the center of the sphere.

In general, the acoustic infinite elements produce the most accurate results when the reference node is located near the center of the region enclosed by the infinite elements.

Nodal normal vectors are required for an accurate mapping of the infinite domain. The nodal normal vectors must point into the infinite domain and are used to define the portion of the infinite domain treated by a particular infinite element. To cover the infinite domain without overlap, each node attached to an infinite element must have a unique normal. The nodal normal vectors are specified or calculated as follows.

User-specified alternative nodal normals ("Normal definitions at nodes," Section 2.1.4) are ignored for acoustic infinite elements and, therefore, cannot be used to define normal directions for acoustic elements. Over the element's surface topology, the normal vectors must be divergent; that is, the area mapped (in two dimensions) or the volume mapped (in three dimensions) must increase with distance into the infinite domain. To ensure this criteria, the normal vectors at each acoustic infinite element node are defined to lie along the vector between that node and the reference point given in the element section property definition. See "Infinite elements," Section 28.3.1, for more information.

Perfectly matched layers

Perfectly matched layers are provided for modeling exterior acoustic problems during a direct-solution steady-state dynamic analysis. The perfectly matched layer is modeled using the acoustic elements that are available in Abaqus/Standard whose behavior is modified appropriately to act as an absorbing layer. The perfectly matched layer absorbs all the waves that are incident to it. For any given problem, it is recommended to have 4–7 layers of elements in the perfectly matched layer region. You specify the coefficients that are used to define the absorbing properties of the perfectly matched layer and the outer limits of the acoustic domain. These limits are used to compute the absorbing properties for the perfectly matched layer.

You cannot use axisymmetric acoustic elements to model the perfectly matched layer. The outer surface of the perfectly matched layer should have a zero pressure boundary condition. To avoid reflections from the perfectly matched layer region, the material properties used for the perfectly matched layer elements should match those of the acoustic elements.

Input File Usage: ∗PERFECTLY MATCHED LAYER, ELSET=*element set label*, MATERIAL=*material name*
∗PML COEFFICIENT, VARIATION=LINEAR

Abaqus/CAE Usage: You cannot define a perfectly matched layer property in Abaqus/CAE.

Mesh refinement

Inadequate mesh refinement is the most common source of difficulties in acoustic and vibration analysis. For reasonable accuracy, at least six representative internodal intervals of the acoustic mesh should fit into the shortest acoustic wavelength present in the analysis; accuracy improves substantially if ten or more internodal intervals are used at the shortest wavelength. In steady-state analyses the shortest wavelength will occur in the medium with the lowest speed of sound, at the highest frequency analyzed. In transient analyses the shortest wavelength present is more difficult to determine before an analysis: it

is reasonable to estimate this wavelength using the highest frequency present in the loads or prescribed boundary conditions.

An "internodal interval" is defined as the distance from a node to its nearest neighbor in an element; that is, the element size for a linear element or half of the element size for a quadratic element. At a fixed internodal interval, quadratic elements are more accurate than linear elements. The level of refinement chosen for the acoustic medium should be reflected in the solid medium as well: the solid mesh should be sufficiently refined to accurately model flexural, compressional, and shear waves.

The level of mesh refinement required depends on the application. Any finite element discretization of a domain in which waves propagate introduces a certain amount of error per wavelength. In meshes that are small in terms of wavelengths, relatively coarse (for example, six internodal intervals per wavelength) meshes may be adequate. For meshes that contain many wavelengths at the frequency of interest, the per-wavelength finite element discretization error accumulates, generally necessitating greater levels of refinement. In these larger meshes the accumulated per-wavelength error may be present throughout the mesh if refinement is inadequate.

The acoustic wavelength decreases with increasing frequency, so there is an upper frequency limit for a given mesh. Let L_{max} represent the maximum internodal interval of an element in a mesh, n_{min} the number of internodal intervals we desire per acoustic wavelength ($n_{min} \geq 10$ is recommended), f_{max} the cyclical frequency of excitation, and $c = \sqrt{\mathrm{K}_f/\rho_f}$ the speed of sound, where K_f is the bulk modulus of the acoustic medium and ρ_f is its density. The requirements are then expressed as

$$L_{max} < \frac{c}{n_{min} f_{max}} \qquad \text{or} \qquad f_{max} < \frac{c}{n_{min} L_{max}}.$$

The above expressions can be used to estimate the maximum allowable element length if the frequency is given or the maximum frequency for which a given mesh size is valid. For example, in air at room temperature, $c \approx 343$ meters per second. The following table gives some values for maximum internodal distances to model given maximum frequencies f_{max} accurately:

Maximum Frequency of Interest, f_{max}	**Maximum Internodal Interval,** L_{max}**,** $n_{min} \equiv 8$	**Maximum Internodal Interval,** L_{max}**,** $n_{min} \equiv 12$
100 Hz	< 430 mm	< 286 mm
500 Hz	< 86 mm	< 57 mm
1000 Hz	< 43 mm	<29mm
20 kHz	< 2.1 mm	< 1.4mm

For exterior problems the accuracy of an analysis also depends on the accuracy of the absorbing boundary condition. As mentioned above, the absorbing boundary impedance conditions implemented in Abaqus are used with a standoff thickness r_1 of acoustic finite elements between the acoustic sources and the radiating boundary. Since the approximate radiation conditions converge to the exact condition in the limit of infinite standoff, a greater standoff thickness improves the accuracy of the solution. The standoff thickness r_1 is expressed as m_{min} wavelengths at the minimum frequency to be analyzed:

$$r_1 > \frac{c\, m_{min}}{f_{min}} \qquad \text{or} \qquad f_{min} > \frac{c\, m_{min}}{r_1}.$$

Continuing the example using the properties of air, we can calculate the recommended minimum standoff thicknesses corresponding to a specified minimum frequency of interest, using $m_{min} = 1/3$:

Minimum Frequency of Interest, f_{min}	**Radiation Boundary Standoff,** r_1
100 Hz	> 1140 mm
500 Hz	> 230 mm
1000 Hz	> 114 mm
20 kHz	> 5.7 mm

The computational requirements for an exterior problem thus depend on both the radiation boundary standoff and the internodal distance. The number of nodes N in a model depends on the volume of the mesh, controlled by r_1 and the spatial dimension d, and the mesh density, controlled by L_{max}. The exact number of nodes depends on the details of the model, but the expression

$$N \sim \left(\frac{r_1}{L_{max}}\right)^d \sim \left(\frac{f_{max}}{f_{min}}\, m_{min}\, n_{min}\right)^d$$

indicates the size of the model with respect to the ratio of the maximum to minimum frequencies in a given analysis. Because the mesh size for an exterior problem exhibits such strong dependence on the bandwidth, $f_{max} - f_{min}$, you can control the size of an analysis by splitting the band. For example, if the overall frequency range of interest is 100 to 10000 Hz, a single spherical mesh covering this band in three dimensions has size

$$N \sim (100\, m_{min}\, n_{min})^3 \sim 1e6\, (m_{min}\, n_{min})^3 .$$

However, splitting the problem into two bands, $(100,\ 1000)$ and $(1000,\ 10000)$, and creating an exterior mesh for each band, results in two analyses of size

$$N \sim (10\, m_{min}\, n_{min})^3 \sim 1000\ (m_{min}\, n_{min})^3 .$$

In coupled acoustic-structural systems there usually exist different wave speeds for the fluid and solid media. In the region of the acoustic-structural interface, the wave phenomena in both media may exhibit length scales characteristic of the slower medium; that is, the length scale of the wave dynamics may be as short as the shorter wavelength, corresponding to the lower wave speed. This result follows from the fact that the two media are coupled at the boundary. The region near the acoustic-structural interface where these effects are important is usually no thicker than the shorter wavelength.

For example, in an analysis involving water interacting with rubber, the wave speed in the rubber may be much lower than that of water. A finite element mesh used to model this problem in detail would require refinement down to six (or more) nodes per shorter wavelength, on both sides of the interface.

On the water side (faster, longer wavelength) accuracy will probably not be compromised significantly if this region of high refinement extends no further into the water than one short wavelength. Of course, in some analyses the effects in the vicinity of the interface may be unimportant. Then, the two meshes can be refined only so far as to represent their own characteristic wavelengths accurately.

Output

Nodal output variable POR (pressure magnitude at the nodes of the acoustic elements) is available for an acoustic medium (in Abaqus/CAE this output variable is called PAC). When the scattered wave formulation (default) is used with incident wave loading, output variable POR represents only the scattered pressure response of the model and does not include the incident wave loading itself. When the total wave formulation is used, output variable POR represents the total dynamic acoustic pressure, which includes contributions from both incident and scattered waves as well as the dynamic effects of fluid cavitation. For either formulation output variable POR does not include the acoustic static pressure.

In Abaqus/Explicit an additional nodal output variable PABS (the absolute pressure, equal to the sum of POR and the acoustic static pressure) is available. When the dynamic effects of fluid cavitation are of interest, you can specify the acoustic static pressure in an acoustic analysis that uses the total wave formulation. If the acoustic static pressure is not specified in an acoustic region, it is assumed to be large; thus precluding cavitation in that region.

For general steps, including implicit and explicit dynamic steps, no energy quantities are computed for acoustic elements. Consequently, these elements will not contribute to the total energy balance.

Steady-state dynamic output

For steady-state dynamic analysis POR is complex and can be displayed in several forms in the Visualization module of Abaqus/CAE. The phase angle (PPOR) is available as output to the data (`.dat`) and results (`.fil`) files.

Several additional secondary quantities are available for multidimensional acoustic finite elements in direct-solution steady-state dynamic or subspace-based steady-state dynamic analysis. The "sound pressure level" is defined as:

$$SPL \equiv 20 \log_{10} p_{RMS} - 20 \log_{10} p_{REF},$$

where p_{REF} is defined as a physical constant in the model (see "Defining the reference pressure" below), and the p_{RMS} is computed from the complex-valued acoustic pressure $\tilde{p}$ at any point using the formula:

$$2p_{RMS}^2 = \Re(\tilde{p})^2 + \Im(\tilde{p})^2.$$

The acoustic particle velocity at any material point is

$$\tilde{\mathbf{v}} = \frac{i}{\tilde{\rho}\omega} \frac{\partial \tilde{p}}{\partial \mathbf{x}}.$$

The acoustic intensity vector, a measure of the rate of flow of energy at a material point, is

$$\tilde{\mathbf{I}} = -\frac{1}{2}\sigma\hat{\tilde{\mathbf{v}}}.$$

In an acoustic medium the stress tensor is simply the acoustic pressure times the identity tensor, so this expression simplifies to

$$\tilde{\mathbf{I}} = \frac{-1}{2i\hat{\tilde{\rho}}\omega}\left[\tilde{p}\frac{\partial\hat{\tilde{p}}}{\partial\mathbf{x}}\right].$$

The hats denote complex conjugation. The real part of the intensity is referred to as the "active intensity," and the imaginary part is the "reactive intensity." The acoustic pressure gradient is also available for acoustic finite elements in steady-state dynamic analysis.

In steady-state dynamic analysis, additional nodal output quantities are available for acoustic infinite elements.

PINF denotes the complex pressure coefficients of the infinite element shape functions. These coefficients can be used to visualize the exterior acoustic field (i.e., within the volume of the acoustic infinite elements) using scripting in the Visualization module of Abaqus/CAE; see "Using infinite elements to compute and view the results of an acoustic far-field analysis," Section 9.10.11 of the Abaqus Scripting User's Guide. INFN is the normal vector used by the acoustic infinite element to define the element volume. INFR denotes the radius used for the element at that node, and INFC denotes the element cosine; that is, the minimum dot product between the nodal normal vector and the acoustic infinite element facet normal vectors attached to that node. See "Acoustic infinite elements," Section 3.3.2 of the Abaqus Theory Guide, for more complete descriptions of these quantities. INFN, INFR, INFC are useful in debugging a model using acoustic infinite elements; consequently, it is sometimes valuable to perform a steady-state dynamics, direct analysis on a model to visualize this information.

For steady-state dynamic steps, energy quantities are available for acoustic elements. These elements contribute to the total energy balance in steady-state dynamics.

Defining the reference pressure

You must define the reference pressure, p_{REF}, used to compute the sound pressure level; there is no default value for the reference pressure.

Input File Usage: *PHYSICAL CONSTANTS, SPL REFERENCE PRESSURE=p_{REF}

Abaqus/CAE Usage: You cannot define a reference pressure in Abaqus/CAE.

Input file template

The following is an example of the step definition for a direct-solution steady-state dynamic acoustic analysis that looks for the response of a model at six frequencies ranging linearly from $f = 10$ to $f = 100$ cycles/time. The pressure at node set **INPUT** (nodes at the boundary) is prescribed to have an in-phase component of 3.0 and an out-of-phase component of −4.0 (i.e., a complex value of $3.0 - 4.0i$). An in-phase inward volume acceleration of 40.0 is specified at node 10.

On the surface **LINER1** an impedance is defined based on the impedance property named **CARPET1**. On the second face of all of the elements in element set **PAD**, another surface impedance based on **CARPET1** is defined. On the fourth face of all of the elements in element set **END**, the default plane wave boundary condition is specified.

Printed output of pressure magnitude and phase is requested for node set **OUTPUT**. Acoustic pressure and displacement are written to the output database. All output is written once for each of the six excitation frequencies.

```
*HEADING
…
*SURFACE, NAME=LINER1
 10, S3
*IMPEDANCE PROPERTY, NAME=CARPET1
Data describing impedance properties as a function of frequency
**
*STEP
*STEADY STATE DYNAMICS, DIRECT
 10, 100, 6
*SIMPEDANCE, PROPERTY=CARPET1
 LINER1,
**
*IMPEDANCE, PROPERTY=CARPET1
 PAD, I2
 *IMPEDANCE
 END, I4
** Apply complex pressure at node set INPUT
*BOUNDARY, REAL
 INPUT, 8, 8, 3.
*BOUNDARY, IMAGINARY
 INPUT, 8, 8, -4.
** Apply an in-phase inward volume acceleration at node 10
*CLOAD
 10, 8, 40.
** Output requests
*NODE PRINT, NSET=OUTPUT, TOTALS=YES
 POR, PPOR
*OUTPUT, FIELD
*NODE OUTPUT
U, PU, POR
*END STEP
```

The following is a template of the step definition for an Abaqus/Explicit acoustic analysis. On the surface **SURF** an impedance is defined based on the impedance property named **IPROP**. In addition, impedance is defined on elements or element sets.

```
*HEADING
…
*ELEMENT, TYPE=AC2D4R
…
**
*SURFACE, NAME=SURF
Data line to define surface
*IMPEDANCE PROPERTY, NAME=IPROP
Data describing impedance properties
**
*STEP
*DYNAMIC, EXPLICIT or *DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
Data line to define incrementation
*SIMPEDANCE, PROPERTY=IPROP
 SURF,
**
*IMPEDANCE
Data lines to define impedance on elements or element sets
*CLOAD
Data line to define acoustic loads
*FIELD
Data line to define field variable values
*END STEP
```

The following template is representative of a coupled acoustic-structural shock problem using the preferred interface for applying incident wave loading (see “Incident wave loading due to external sources” in “Acoustic and shock loads,” Section 34.4.6):

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define solid elements
*ELEMENT, TYPE=…, ELSET=BEAM
Data lines to define beam elements
*BEAM SECTION,ELSET=BEAM,MATERIAL=...
Data lines to define the beam stiffness section properties
*BEAM FLUID INERTIA
Data line to define the beam virtual mass property
```

***SURFACE, NAME=IW_LOAD_ACOUSTIC**
Data lines to define the acoustic surface loaded by the incident wave
***SURFACE, NAME=IW_LOAD_SOLID**
Data lines to define the solid surface loaded by the incident wave
***SURFACE, NAME=IW_LOAD_BEAM**
Data lines to define the beam surface loaded by the incident wave
***SURFACE, NAME=TIE_ACOUSTIC**
Data lines to define the acoustic surface interface with the solid mesh
***SURFACE, NAME=TIE_SOLID**
Data lines to define the solid surface interface with the acoustic mesh
***INCIDENT WAVE INTERACTION PROPERTY, NAME=IWPROP, TYPE=SPHERE**
Data lines to define a spherical incident wave field
***UNDEX CHARGE PROPERTY**
Data lines to define the underwater explosion parameters
**** Tie the acoustic mesh to the solid mesh**
***TIE, NAME=COUPLING**
TIE_ACOUSTIC, TIE_SOLID
***STEP**
***DYNAMIC, EXPLICIT or *DYNAMIC**
**** Load the acoustic surface**
***INCIDENT WAVE INTERACTION, PROPERTY=IWPROP**
IW_LOAD_ACOUSTIC, *source node*, *standoff node*, *reference magnitude*
**** Load the solid surface**
***INCIDENT WAVE INTERACTION, PROPERTY=IWPROP**
IW_LOAD_SOLID, *source node*, *standoff node*, *reference magnitude*
**** Load the beam surface**
***INCIDENT WAVE INTERACTION, PROPERTY=IWPROP**
IW_LOAD_BEAM, *source node*, *standoff node*, *reference magnitude*
***END STEP**

The following template is representative of a coupled acoustic-structural shock problem using the alternative interface for applying incident wave loading:

***HEADING**
…
***ELEMENT, TYPE=…, ELSET=ACOUSTIC**
Data lines to define acoustic elements
***ELEMENT, TYPE=…, ELSET=SOLID**
Data lines to define solid elements
***ELEMENT, TYPE=…, ELSET=BEAM**
Data lines to define beam elements
***BEAM SECTION,ELSET=BEAM,MATERIAL=...**
Data lines to define the beam stiffness section properties

```
*BEAM FLUID INERTIA
```
Data line to define the beam virtual mass property
```
*SURFACE, NAME=IW_LOAD_ACOUSTIC
```
Data lines to define the acoustic surface loaded by the incident wave
```
*SURFACE, NAME=IW_LOAD_SOLID
```
Data lines to define the solid surface loaded by the incident wave
```
*SURFACE, NAME=IW_LOAD_BEAM
```
Data lines to define the beam surface loaded by the incident wave
```
*SURFACE, NAME=TIE_ACOUSTIC
```
Data lines to define the acoustic surface interface with the solid mesh
```
*SURFACE, NAME=TIE_SOLID
```
Data lines to define the solid surface interface with the acoustic mesh
```
*INCIDENT WAVE PROPERTY, NAME=IWPROP, TYPE=SPHERE
```
Data lines to define a spherical incident wave field
```
*INCIDENT WAVE FLUID PROPERTY
```
Data lines to define the fluid properties for the incident wave field
```
*AMPLITUDE, DEFINITION=BUBBLE, NAME=PRESSUREVTIME
```
Data lines to define the underwater explosion parameters
```
** Tie the acoustic mesh to the solid mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
*STEP
*DYNAMIC or *DYNAMIC, EXPLICIT
** Load the acoustic surface
*INCIDENT WAVE, PRESSURE AMPLITUDE=PRESSUREVTIME,
PROPERTY=IWPROP
IW_LOAD_ACOUSTIC, {amplitude}
** Load the solid surface and the beam surface
*INCIDENT WAVE, PRESSURE AMPLITUDE=PRESSUREVTIME,
PROPERTY=IWPROP
IW_LOAD_SOLID, {amplitude}
IW_LOAD_BEAM, {amplitude}
*END STEP
```

The following template is representative of a coupled acoustic-structural sound transmission problem using the preferred interface for applying incident wave loading (see "Incident wave loading due to external sources" in "Acoustic and shock loads," Section 34.4.6):

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
```
Data lines to define acoustic elements
```
*ELEMENT, TYPE=…, ELSET=SOLID
```

Data lines to define solid elements
***SURFACE, NAME=IW_LOAD_ACOUSTIC**
Data lines to define the acoustic surface loaded by the incident wave
***SURFACE, NAME=IW_LOAD_SOLID**
Data lines to define the solid surface loaded by the incident wave
***SURFACE, NAME=TIE_ACOUSTIC**
Data lines to define the acoustic surface interface with the solid mesh
***SURFACE, NAME=TIE_SOLID**
Data lines to define the solid surface interface with the acoustic mesh
***INCIDENT WAVE INTERACTION PROPERTY, NAME=FIRST, TYPE=SPHERE**
Data lines to define a spherical incident wave field
***INCIDENT WAVE INTERACTION PROPERTY, NAME=SECOND, TYPE=PLANE**
Data lines to define a planar incident wave field
**** Tie the acoustic mesh to the solid mesh**
***TIE, NAME=COUPLING**
TIE_ACOUSTIC, TIE_SOLID
***STEP**
***STEADY STATE DYNAMICS, DIRECT or SUBSPACE PROJECTION**
**** Define the load on the acoustic and solid surfaces due to**
**** the first loading case:**
***LOAD CASE, NAME=FIRST_SOURCE**
**** Load the acoustic surface: define the real part at the**
**** standoff point**
***INCIDENT WAVE INTERACTION, PROPERTY=FIRST, REAL**
IW_LOAD_ACOUSTIC, *first source node*, *first standoff node*, *reference magnitude*
**** Load the acoustic surface: define the imaginary part at the**
**** standoff point**
***INCIDENT WAVE INTERACTION, PROPERTY=FIRST, IMAGINARY**
IW_LOAD_ACOUSTIC, *first source node*, *first standoff node*, *reference magnitude*
**** Load the solid surface: define the real part at the**
**** standoff point**
***INCIDENT WAVE INTERACTION, PROPERTY=FIRST, REAL**
IW_LOAD_SOLID, *first source node*, *first standoff node*, *reference magnitude*
**** Load the solid surface: define the imaginary part at the**
**** standoff point**
***INCIDENT WAVE INTERACTION, PROPERTY=FIRST, IMAGINARY**
IW_LOAD_SOLID, *first source node*, *first standoff node*, *reference magnitude*
***END LOAD CASE**
**** Define the load on the acoustic and solid surfaces due to**
**** the next loading case:**
***LOAD CASE, NAME=SECOND_SOURCE**
**** Load the acoustic surface: define the real part at the**

```
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, REAL
IW_LOAD_ACOUSTIC, second source node, second standoff node, reference magnitude
** Load the acoustic surface: define the imaginary part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, IMAGINARY
IW_LOAD_ACOUSTIC, second source node, second standoff node, reference magnitude
** Load the solid surface: define the real part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, REAL
IW_LOAD_SOLID, second source node, second standoff node, reference magnitude
** Load the solid surface: define the imaginary part at the
** standoff point
*INCIDENT WAVE INTERACTION, PROPERTY=SECOND, IMAGINARY
IW_LOAD_SOLID, second source node, second standoff node, reference magnitude
*END LOAD CASE
*END STEP
```

6.11 Abaqus/Aqua analysis

- “Abaqus/Aqua analysis,” Section 6.11.1

6.11.1 Abaqus/AQUA ANALYSIS

Product: Abaqus/Aqua

References

- "UWAVE," Section 1.1.59 of the Abaqus User Subroutines Reference Guide
- "Defining an analysis," Section 6.1.2
- *AQUA
- *CLOAD
- *C ADDED MASS
- *DLOAD
- *D ADDED MASS
- *SURFACE SECTION
- *WAVE
- *WIND

Overview

An Abaqus/Aqua analysis:

- is used to apply steady current, wave, and wind loading to submerged or partially submerged structures in problems such as the modeling of offshore piping installations or the analysis of marine risers;
- can be performed using the static ("Static stress analysis," Section 6.2.2), direct-integration dynamic ("Implicit dynamic analysis using direct integration," Section 6.3.2), explicit dynamics ("Explicit dynamic analysis," Section 6.3.3), or eigenfrequency extraction ("Natural frequency extraction," Section 6.3.5) procedures;
- will calculate drag, buoyancy, and inertia loading only for beam, pipe, elbow, truss, and certain rigid elements;
- can include elements that model spud cans for jack-up foundation analysis in Abaqus/Standard; and
- can be linear or nonlinear.

Procedures available for Aqua analysis

Aqua loading can be applied in static steps ("Static stress analysis," Section 6.2.2), direct-integration dynamic steps ("Implicit dynamic analysis using direct integration," Section 6.3.2), and explicit dynamic steps ("Explicit dynamic analysis," Section 6.3.3). During these steps fluid particle velocity is assumed to consist of two superposed effects: steady currents, which can vary with elevation and location, and gravity waves. Fluid particle accelerations are associated with gravity waves only.

The fluid particle velocities and accelerations are used to calculate drag and inertia loading on the immersed body. Abaqus/Aqua also computes the fluid surface elevation and allows for partial immersion; drag and buoyancy loadings are omitted for those parts of the structure that are above the fluid surface or below the seabed level.

An eigenfrequency extraction step ("Natural frequency extraction," Section 6.3.5) can be used to extract the natural frequencies of a structure prestressed by the Aqua loading in a static or direct-integration dynamic step (if that step included the effects of nonlinear geometry). The added-mass effect due to fluid inertia loads can be included in an eigenfrequency extraction step.

Defining an Abaqus/Aqua problem

Aqua loads are applied in the following manner:

1. The fluid properties and steady current velocity are defined for the model.
2. Gravity waves and wind velocity are defined for the model.
3. Drag, buoyancy, and fluid inertia loads are applied to elements and nodes of the structure using distributed or concentrated load definitions within the static or direct-integration dynamic step definition. The magnitudes of the loads applied are determined by the fluid properties, steady current, wave, and wind definitions.
4. In an eigenfrequency extraction step concentrated and distributed added mass definitions are used (instead of concentrated and distributed loads) to include the effects of fluid inertia.

The load-stiffness terms from Abaqus/Aqua loads, which are important in geometrically nonlinear analysis, are fundamentally unsymmetric. Therefore, the unsymmetric matrix solution and storage scheme should be used for the step when nonlinear geometric effects are included ("Defining an analysis," Section 6.1.2). It is essential to use the unsymmetric solver when the structure being analyzed is flexible (see, for example, "Slender pipe subject to drag: the "reed in the wind"," Section 1.13.3 of the Abaqus Benchmarks Guide).

On the other hand, if a relatively stiff structure is subject to Aqua loads or if a dynamic step uses small time increments, the unsymmetric load-stiffness terms may not be dominant and you may be able to obtain a convergent solution with the symmetric solver (see, for example, "Riser dynamics," Section 12.1.2 of the Abaqus Example Problems Guide).

Coordinate system

The z-coordinate axis must point vertically for three-dimensional cases, and the y-coordinate axis must point vertically for two-dimensional cases. For the three-dimensional case the still fluid surface (when there is no wave motion) lies in a plane that is parallel to the x–y plane. For the two-dimensional case it lies parallel to the x-axis. The position of the still fluid surface is specified as part of the fluid property data.

Defining the fluid properties

Aqua loadings require the definition of fluid density, seabed and free surface elevation, and the gravitational constant.

Input File Usage: *AQUA
seabed elevation, free surface elevation, gravitational constant, fluid density

The *AQUA option must be included in the model data portion of the input file.

Defining a steady current

Steady currents are defined by giving steady fluid velocity as a function of elevation and location. Elevation is defined in the positive z-direction for three-dimensional models and in the positive y-direction for two-dimensional models. For two-dimensional cases the z-component of the steady current velocity is ignored. See "Input syntax rules," Section 1.2.1, for an explanation of how to define one property (in this case steady current velocity) as a function of multiple independent variables.

If the fluid velocity is not a function of elevation or location (for example, when modeling a problem in a coordinate system that moves uniformly through the still fluid, such as a tow-out analysis), only one fluid velocity need be specified.

The steady current velocities can be scaled by referring to an amplitude curve ("Amplitude curves," Section 34.1.2) from the concentrated or distributed load definitions used to apply drag loads, as described later.

Input File Usage: *AQUA
fluid properties on first data line (described above)
X-velocity$_{fluid}$, Y-velocity$_{fluid}$, Z-velocity$_{fluid}$, elevation, X-coord, Y-coord
...

Defining gravity waves

Gravity waves are defined by specifying a wave theory. The wave theory determines fluid acceleration, velocity, and pressure field fluctuations. The fluid acceleration and velocity field fluctuations contribute to the drag loads. The fluid pressure field fluctuations contribute to the buoyancy loads.

Choosing the type of wave theory to be used

Using Abaqus/Aqua in an Abaqus/Standard analysis, you can choose Airy linear wave theory, Stokes fifth-order wave theory, wave data read from a gridded mesh, or fluid kinematics defined in user subroutine **UWAVE**. For Airy and Stokes waves the fluid surface elevation and the fluid particle velocities and accelerations will be calculated as functions of time and location based on the wave definition. If wave data are provided in the form of a gridded mesh, you must specify these quantities. If user subroutine **UWAVE** is used, the fluid kinematics must be defined in that routine.

Similarly, using Abaqus/Aqua in an Abaqus/Explicit analysis, you can choose Airy linear wave theory, Stokes fifth-order wave theory, or fluid kinematics defined in user subroutine **VWAVE**.

All of the built-in wave theories assume a series of waves in the horizontal plane (the plane of the fluid surface) that are unaffected by any fluid-structural interaction. The Airy and Stokes theories are based on irrotational flow of an inviscid, incompressible fluid, where the wave height H is small compared to the still water depth d. The bottom of the fluid is assumed to be flat (the still water depth is constant).

The Ursell parameter,

$$\frac{H}{\lambda}\left(\frac{\lambda}{d}\right)^3,$$

where λ is the wavelength, should be much less than 1.0 for Airy wave theory to be applicable and should be less than 10.0 for Stokes theory to be applicable. For ratios of H/λ greater than 0.142, the crest of the wave is predicted to break. The assumed boundary conditions on the free surface are then no longer valid in either theory, which limits the maximum wave amplitude for either theory.

Airy wave theory

Linear Airy wave theory is generally used when the ratio of wave height to water depth, H/d, is less than 0.03, provided that the water is deep (ratio of water depth to wavelength, d/λ, is greater than 20). Convective acceleration terms are neglected in the Airy theory as part of the linearization. The Airy wave theory is described in detail in "Airy wave theory," Section 6.2.2 of the Abaqus Theory Guide.

Since the Airy wave theory is linear, any number of wave trains traveling in different directions across the water can be defined; the fluid particle velocities and accelerations sum by linear superposition. The direction of each wave component is given by specifying the direction cosines of a vector, d_N, lying in the plane defined by the still fluid surface.

By default, Airy waves are defined in terms of wavelength, λ_N. Alternatively, you can define the waves in terms of wave period, τ_N. For Airy wave theory the wavelength and period of each component are related by

$$\frac{2\pi}{\tau_N^2} = g\frac{1}{\lambda_N}\tanh\frac{2\pi h}{\lambda_N},$$

where

- τ_N is the period of this component,
- g is the gravitational acceleration,
- λ_N is the wavelength, and
- h is the undisturbed (still) water depth.

Input File Usage: Use the following option to define an Airy wave in terms of wavelength:

∗WAVE, TYPE=AIRY
amplitude, wavelength, phase angle, x-direction cosine, y-direction cosine

Use the following option to define an Airy wave in terms of wave period:

∗WAVE, TYPE=AIRY, WAVE PERIOD
amplitude, wave period, phase angle, x-direction cosine, y-direction cosine

In either case repeat the data line to define multiple wave trains.

Stokes fifth-order wave theory

The Stokes fifth-order wave theory is a deep-water wave theory that is valid for relatively large wavelengths. Convective terms are included in the fluid particle acceleration calculations for Stokes fifth-order theory and can be significant for larger H/λ ratios. The Stokes wave theory is described in detail in "Stokes wave theory," Section 6.2.3 of the Abaqus Theory Guide.

Because the Stokes fifth-order wave theory is nonlinear, only one wave train is allowed in an analysis. The relationship between wavelength and period of the waves in Stokes fifth-order theory is not as simple as that for the Airy theory, although the formula given above is a first-order approximation. Stokes waves can be defined only in terms of the wave period, τ.

Input File Usage: *WAVE, TYPE=STOKES
wave height, wave period, phase angle, direction of travel cosines

Gridded wave data

You can choose to provide wave surface elevations, particle velocities and accelerations, and the dynamic pressure at points in a user-defined grid through a binary data file. The binary file contains information about the wave definition, the location of the grid points where wave information is specified, and the wave kinematics at user-defined times. At spatial locations within the user-defined grid, Abaqus/Aqua will interpolate the wave kinematics from the nearest grid points, using either linear or quadratic interpolation. When a point on the structure is above the user-defined grid, Abaqus/Aqua assumes that the point is above the free surface elevation. Hence, no fluid loads are applied. If a point on the structure falls outside the user-defined spatial grid without being above the grid, Abaqus/Aqua finds the wave kinematics at the nearest point within the grid and uses those values at the point on the structure.

Input File Usage: *WAVE, TYPE=GRIDDED, DATA FILE=*file_name*

Binary data file requirements for gridded wave data

The data file must contain the following unformatted (binary) records (see "Aqua load cases," Section 3.11.1 of the Abaqus Verification Guide). The data for the FORTRAN WRITE statement are given for each record:

First record:

```
NCOMP, DTG, NWGX, NWGY, NWGZ, IPDYN
```

where

NCOMP is the number of wave components to be read in the data file;

DTG is the time increment at which wave data are given on the grid;

NWGX is the number of grid points in the grid's x-direction;

NWGY is the number of grid points in the grid's y-direction—if this number is one, Abaqus/Aqua assumes that the wave data are constant with respect to the local y-direction;

NWGZ is the number of grid points in the grid's z-direction—if this number is zero or one, the analysis is two-dimensional and the y-direction is vertical; and

IPDYN is an integer flag indicating whether dynamic pressure information is stored (**IPDYN**=1) or not stored (**IPDYN**=0) in the gridded wave file.

Second record:

```
(AMP(K1), WXL(K1), PHI(K1), K1=1,NCOMP)
```

where

NCOMP is read on the first record, above;

AMP contains the wave component amplitude, a_N;

WXL contains the wavelength of this component, λ_N; and

PHI contains the phase angle of this component, ϕ_N (in degrees).

The second record of this file contains the wave component data used to generate the gridded wave data; it is not used by Abaqus/Aqua. This record is provided only for information in user subroutine **UEL** by using the **GETWAVE** interface (see "Obtaining wave kinematic data in an Abaqus/Aqua analysis," Section 2.1.13 of the Abaqus User Subroutines Reference Guide). The meaning of the arrays **AMP** and **WXL** is left to you; however, **PHI** is converted to radians.

Third record:

```
(WGX(K1),K1=1,NWGX), (WGY(K1),K1=1,NWGY), (WGZ(K1),K1=1,NWGZ)
```

where

NWG*i* are read on the first record, above;

WGX contains the local x-coordinates of the grid points;

WGY contains the local y-coordinates of the grid points; and

WGZ contains the local z-coordinates of the grid points (not included in the gridded wave file for two-dimensional analyses).

Remaining records if **IPDYN=0**:

For three dimensions:

```
(((WGVX(K1,K2,K3), WGVY(K1,K2,K3), WGVZ(K1,K2,K3),
WGAX(K1,K2,K3), WGAY(K1,K2,K3), WGAZ(K1,K2,K3), K3=1,NWGZ),
WZCRST(K1,K2), NCRST(K1,K2), K1=1,NWGX), K2=1,NWGY)
```

For two dimensions:

```
((WGVX(K1,K2), WGVY(K1,K2), WGAX(K1,K2), WGAY(K1,K2),
K2=1,NWGY), WZCRST(K1), NCRST(K1), K1=1,NWGX)
```

Remaining records if **IPDYN=1**:

For three dimensions:

```
(((WGVX(K1,K2,K3), WGVY(K1,K2,K3), WGVZ(K1,K2,K3),
```

```
WGAX(K1,K2,K3), WGAY(K1,K2,K3), WGAZ(K1,K2,K3),
P(K1,K2,K3), DPDZ(K1,K2,K3), K3=1,NWGZ),
WZCRST(K1,K2), NCRST(K1,K2), K1=1,NWGX), K2=1,NWGY)
```

For two dimensions:

```
((WGVX(K1,K2), WGVY(K1,K2), WGAX(K1,K2), WGAY(K1,K2),
P(K1,K2), DPDZ(K1,K2), K2=1,NWGY),
WZCRST(K1), NCRST(K1), K1=1,NWGX)
```

where

WGVX	contains the local x-components of the wave particle velocity,
WGVY	contains the local y-components of the wave particle velocity,
WGVZ	contains the local z-components of the wave particle velocity,
WGAX	contains the local x-components of the wave particle acceleration,
WGAY	contains the local y-components of the wave particle acceleration,
WGAZ	contains the local z-components of the wave particle acceleration,
WZCRST	contains the wave surface elevation,
NCRST	contains the index for the vertical grid level just above the instantaneous water surface,
P	contains the dynamic pressure, and
DPDZ	contains the gradient of the dynamic pressure in the vertical direction.

User-defined wave theory in Abaqus/Standard

A user-defined wave theory can be coded in user subroutine **UWAVE** in an Abaqus/Aqua analysis in Abaqus/Standard. You can define the fluid particle velocity, acceleration, free surface elevation, and fluid pressure field in the user subroutine.

For stochastic analysis, you can specify a random number seed, r, and define frequency/amplitude pairs that define the wave spectrum. During the analysis Abaqus/Aqua stores an intermediate configuration that can be used in the user subroutine to compute the stochastic description of the waves. The intermediate configuration is initialized as the reference configuration and is replaced by the current configuration only when requested by the user subroutine. In this way the stochastic description of the wave field can be stored in an external database and recalculated only when necessary.

Input File Usage: Use the following option to specify the wave kinematics in user subroutine **UWAVE**:

∗WAVE, TYPE=USER

Use the following option for stochastic analysis to make the intermediate configuration available in user subroutine **UWAVE**:

∗WAVE, TYPE=USER, STOCHASTIC=*r*
frequency, amplitude
...

User-defined wave theory in Abaqus/Explicit

A user-defined wave theory can be coded in user subroutine **VWAVE** in an Abaqus/Aqua analysis in Abaqus/Explicit. You can define the fluid particle velocity, acceleration, free surface elevation, and fluid pressure field in the user subroutine.

The quantities required to define the wave kinematics can be specified as properties and passed into the user subroutine. For example, in the case of stochastic wave kinematics, any required seed variable and/or frequency-amplitude data pairs can be specified as properties.

You can also declare and use state variables for user-defined wave calculations, which will be provided at the nodes and initialized to zero at the beginning of the step. You have to update the state variables within the user subroutine. For example, the state variables can be used to store any intermediate configuration of the structure that is used to describe a stochastic wave field.

Input File Usage: Use the following option to specify the wave kinematics in user subroutine **VWAVE**:

∗WAVE, TYPE=USER

Use the following option to specify properties available as a real-array argument **PROPS** of size **NPROPS** in user subroutine **VWAVE**:

∗WAVE, TYPE=USER, PROPERTIES=*nprops*
prop_1, prop_2, ..., prop_8
..., prop_nprops

Use the following option to specify state variables available as a real-array argument **STATEVAR** of size **NSTATEVAR** in user subroutine **VWAVE**:

∗WAVE, TYPE=USER, DEPVAR=*nstatevar*

Wave position as a function of time

For Airy and Stokes waves the position of the wave at time $t = 0$ can be chosen by specifying the phase angle ϕ of the wave (or wave components for Airy waves). By default, the waves are chosen such that they have a trough (vertical displacement of the fluid surface is a minimum) at the origin of the horizontal axes at time $t = 0$. You can change this trough by introducing a phase angle ϕ for the waves. A positive phase angle shifts the waves backward in their travel direction (see Figure 6.11.1–1).

The time t used in the wave theory is the total time in the analysis. Therefore, if the direct-integration dynamic steps in which Airy or Stokes waves are applied are preceded by any steps other than direct-integration dynamic steps (such as static steps), it is usually convenient to make the time period in these steps very small compared to the period of the wave.

Because total time is used, the phase of the wave will be continuous from the end of one dynamic step to the beginning of the next dynamic step.

Defining a minimum wave trough elevation

For computational efficiency Abaqus/Aqua uses a minimum wave trough elevation below which the structure is assumed to be immersed. Below this elevation no calculation of the fluid surface need be done

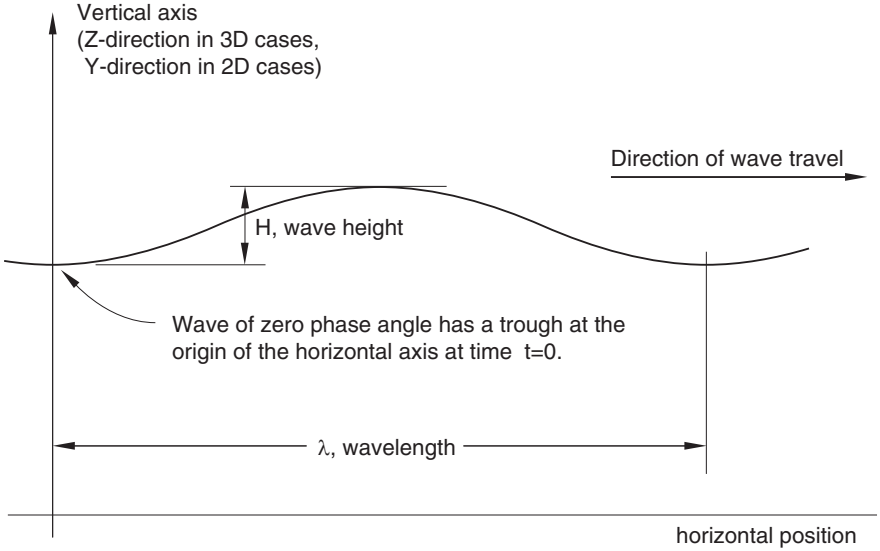


Figure 6.11.1–1 Wave of zero phase angle.

to determine if the point of interest is above the instantaneous free surface. Similarly, a maximum wave elevation is used: any point above the maximum wave elevation is assumed to have no fluid loading.

For Airy and Stokes waves the minimum and maximum wave elevations are calculated from the wave theory.

For gridded waves Abaqus/Aqua allows the definition of a minimum wave trough elevation: z_{min} in three-dimensional analysis or y_{min} in two-dimensional analysis. The structure is always assumed to be immersed below this elevation. The maximum wave elevation is calculated as the still water elevation plus the difference between this elevation and the minimum wave trough elevation. If the minimum wave trough elevation is not specified for gridded waves, Abaqus/Aqua will compare the elevation of every point on the structure with the instantaneous fluid surface as defined by the gridded data. When defining this elevation, make sure that no wave trough ever drops below the minimum wave trough elevation specified.

Input File Usage: ∗WAVE, TYPE=GRIDDED, DATA FILE=*file_name*, MINIMUM=*elevation*

Wave kinematics, dynamic pressure, and extrapolation for Airy waves

A spatial (Eulerian) description of the wave field is used for all wave types; therefore, a structural point's coordinates are used to evaluate the wave kinematics. In geometrically nonlinear analysis the structural point's coordinates are its current coordinates. In geometrically linear analysis the wave kinematics are evaluated using the structural point's reference coordinates.

In both geometrically linear and nonlinear analysis for both static and direct-integration dynamic procedures, submergence is calculated to the instantaneous water level at the current value of total time

for the analysis. Fluid loading is applied only to those points on the structure below the instantaneous water level.

When buoyancy loading is applied in conjunction with a gravity wave, the dynamic pressure due to the disturbance of the still surface is added to the hydrostatic pressure (measured to the still water level) to obtain the total buoyancy loading, except when the buoyancy loading described by a distributed or concentrated load definition overrides the fluid properties given for the Abaqus/Aqua analysis. Dynamic pressure is included for both static and dynamic procedures for Airy, Stokes, and gridded wave types; however, with gridded wave data you can choose to suppress this effect. See "Airy wave theory," Section 6.2.2 of the Abaqus Theory Guide, and "Stokes wave theory," Section 6.2.3 of the Abaqus Theory Guide, for a definition of dynamic pressure.

Although the linearized Airy wave theory assumes that the fluid displacements are small with respect to the wavelength and the fluid depth, these displacements may not be small with respect to the dimensions of the structure immersed in the fluid. As a result of the linearizing approximations special treatment is necessary to calculate the wave kinematics for points below the instantaneous water level but above the still water line. Abaqus/Aqua uses extrapolation with Airy wave theory: the wave velocity, acceleration, and dynamic pressure for points above the still water level but below the instantaneous free surface are taken to be the values evaluated from the wave theory at the still water level. See "Airy wave theory," Section 6.2.2 of the Abaqus Theory Guide, for more details.

Reading the data that define gravity waves from an alternate file

The data for the gravity wave can be contained in an alternate file. See "Input syntax rules," Section 1.2.1, for the syntax of the file name.

Input File Usage: ∗WAVE, INPUT=*file_name*

Visualization of gravity waves

In a three-dimensional analysis you can visualize gravity waves by meshing the free surface of the water with surface elements (see "General surface element library," Section 32.7.2) and identifying elements as aqua visualization elements through the surface section definition.

Aqua visualization elements are used for postprocessing only and do not affect the solution. The following must be true for proper use of these elements:

1. Aqua visualization elements can be connected to other visualization elements only through shared nodes. They cannot be connected in any way to any element in the model that is used during the analysis. This includes connections through shared nodes, kinematic constraints, or surface interactions. Abaqus issues an error message during input file preprocessing if these conditions are not met. For example, if you are doing an Abaqus/Aqua analysis of an offshore oil platform, the visualization elements cannot be connected to any element used to model the platform.
2. Any boundary conditions or loads that are applied on the visualization elements are ignored.
3. Density cannot be assigned to the visualization elements.
4. Reinforcement layers cannot be defined for the visualization elements.
5. To visualize the displacements, you must request displacement field output on the output database (`.odb`) file. During the analysis Abaqus computes the z-displacements of the elements using

whatever wave definitions you include in the model, including user subroutines. Only displacement output can be requested for these elements.

6. The initial *z*-coordinates of the elements should be defined at the still water height; if they are not, Abaqus automatically adjusts them to the still water height during input file preprocessing.

Input File Usage: *SURFACE SECTION, ELSET =*elset_name*, AQUAVISUALIZATION=YES

Defining a wind velocity profile

You can define a wind velocity profile. Wind loading is applied only to elements above the still water surface elevation (defined in the fluid properties). If an element is above the still water depth but is submerged due to a wave, the wind loading will still be applied.

The wind profile is assumed to vary with height (the positive *z*-direction in three-dimensional models, the positive *y*-direction in two-dimensional models) according to the power law wind profile and has no variation in the horizontal plane. The power law wind velocity profile is given by

$$\mathbf{v} = \mathbf{v}^0 \left(\frac{z}{z_0} \right)^{\alpha},$$

where

- $\mathbf{v}(z,t) = v_x(z,t)\hat{\mathbf{i}} + v_y(z,t)\hat{\mathbf{j}}$ is the local wind velocity ($\hat{\mathbf{i}}$ is a unit vector along the local *x*-axis of the wind field, and $\hat{\mathbf{j}}$ is a unit vector along the local *y*-axis of the wind field);
- $\mathbf{v}^0(t)$ is the time-varying wind velocity at the reference height, z_0, as described below;
- α is a user-defined constant (default value 1/7);
- z is the distance above the still water surface (i.e., $z = 0.0$ is the still water surface); and
- z_0 is the reference distance above the still water surface where the time variation of the wind velocity is given.

The wind local system is defined by giving the direction cosines of the unit vector $\hat{\mathbf{i}}$.

Input File Usage: *WIND
air density, z_0, c_x, c_y, *x-direction cosine for* $\hat{\mathbf{i}}$, *y-direction cosine for* $\hat{\mathbf{i}}$, α

Prescribing the time variation of wind velocity at the reference height

The variation in time of the wind profile is defined by $\mathbf{v}^0(t)$, the wind velocity vector time history at a reference height $z = z_0$:

$$\mathbf{v}^0(t) = v_x^0(t)\hat{\mathbf{i}} + v_y^0(t)\hat{\mathbf{j}}.$$

The wind velocity component time histories $v_x^0(t)$ and $v_y^0(t)$ are given by

$$v_x^0(t) = c_x A_x(t); \quad v_y^0(t) = c_y A_y(t),$$

where c_x and c_y are user-defined as described above (with default values of 1.0) and $A_x(t)$ and $A_y(t)$ are time-dependent functions defined by referring to amplitude curves from the concentrated or distributed

load definitions used to apply the wind loading to the model. If no amplitude curve is referenced, the wind velocity components are the constant values $v_x^0(t) = c_x$ and $v_y^0(t) = c_y$.

Geometrically linear versus geometrically nonlinear analysis

In geometrically linear analysis wind velocities are calculated based on the original coordinates of the structure. In geometrically nonlinear analysis the current coordinates of a point on the structure are used to calculate the wind velocity at that point.

Initial conditions

Initial conditions can be applied to the structure in an Abaqus/Aqua analysis in the same way as in static and dynamic analyses without Aqua loads. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Boundary conditions

Boundary conditions can be applied to the structure in an Abaqus/Aqua analysis in the same way as in static and dynamic analyses without Aqua loads. See "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Defining contact at the seabed

Aqua loads are applied only above the seabed. To model the bottom of the sea using a contact plane, the elevation of the contact plane must be slightly higher than the seabed level to avoid ambiguity between the contact condition and applied loading. If the contact plane is at the same level as the seabed, there is a risk that round-off problems will cause Aqua loads not to be applied to nodes in contact with the seabed.

Loads

Steady current, wave, and wind loads are applied to nodes or elements of the structure using concentrated and/or distributed load definitions. Wind loads are applied only if the point is currently above the still fluid surface; fluid loads are applied only if the point is currently below the instantaneous fluid surface and above the seabed. Distributed loads are applied to partly immersed elements.

Concentrated and distributed load definitions cannot be used in eigenfrequency extraction steps, so the loads described below can be applied only in static and direct-integration dynamic steps.

Controlling the time variation and magnitude of Aqua loading

You have three ways to control the magnitude of an Aqua load as a function of time:

1. You can reference a user-defined amplitude curve ("Amplitude curves," Section 34.1.2) from the concentrated or distributed load definition to scale the entire load.
2. You can specify a magnitude factor, M, for the concentrated or distributed load definition, which is used to scale all the load. This magnitude factor allows normalized amplitude curves to be defined and used for multiple loads. The default magnitude factor is always $M = 1.0$.

3. You can reference individual user-defined amplitude curves to scale different components of the loading separately. For example, steady current velocity and wave velocity can be scaled separately by referencing different amplitude curves.

All of these scaling factors are cumulative.

Buoyancy loads

The calculated buoyancy of a structure depends on the orientation of the exposed surface area with respect to the vertical direction. This surface area is calculated automatically by Abaqus/Aqua for distributed buoyancy loading; however, you must specify the exposed area and direction cosines of the outward normal at a node for concentrated buoyancy loading.

Abaqus/Aqua uses a closed-end loading condition while computing the distributed buoyancy forces on all line elements. To obtain an open-end loading condition, concentrated buoyancy loading can be used to counteract the buoyancy load applied to the ends of the elements.

The buoyancy loads require the definition of fluid density, seabed and free surface elevation, and the gravitational constant. The default external fluid properties are defined for the model as described in "Defining the fluid properties." You can override some of these properties by specifying them directly in the distributed or concentrated load definition. This provides for modeling situations where different parts of the structure are subjected to different buoyancy loads, such as a pipe inside another pipe where the static fluid surrounding the inner pipe is different from the fluid surrounding the outer pipe. Gravity waves ("Wave kinematics, dynamic pressure, and extrapolation for Airy waves") do not affect the buoyancy loading when any external fluid property is overridden.

Specifying distributed buoyancy loads

To apply distributed buoyancy loads to elements immersed in a fluid, the effective outer diameter of beam, truss, and one-dimensional rigid elements must be specified. Provide the external fluid density, free surface elevation, and additional pressure to override the default fluid properties to model the situations described above. For situations where it is necessary to model the fluid inside an element, the effective inner diameter of the element must also be given, along with the density and free surface elevation of the fluid inside the element.

Distributed buoyancy loading can be applied to rigid surface elements. However, the effects of waves are ignored for these elements; the buoyancy loading is calculated to the still water level only. For proper application of a positive buoyancy force, the positive normal of R3D3 and R3D4 elements must point into the fluid.

Input File Usage: *DLOAD
element number or set, PB, *M*, *effective outer diameter*, *internal fluid density, effective inner diameter, internal free surface elevation, external fluid density, external free surface elevation, additional pressure*

Specifying concentrated buoyancy loads

For concentrated buoyancy loads applied to nodes immersed in a fluid, the load is calculated based on the sum of the hydrostatic pressure (measured to the still water level) and the dynamic pressure due to

wave action. The total pressure is multiplied by the exposed area associated with the node. The loading is automatically considered to be a follower force in geometrically nonlinear analysis (for elements that have rotational degrees of freedom); therefore, it is not necessary to specify that the load is a follower force. Provide the external fluid density, free surface elevation, and additional pressure to override the default fluid properties to model the situations described above.

Input File Usage: ∗CLOAD
node number or set, TSB, ***M***, *exposed area*, *local coordinate system data, external fluid density, external free surface elevation, additional pressure*

Drag loads

Both waves and wind can cause drag loading on a structure. Fluid drag refers to drag caused by the structural member being immersed in the fluid defined by the fluid properties and the gravity waves and, thus, subject to steady current and wave loading. Fluid drag loading is provided by Morison's equation. Fluid drag loads must be specified in terms of a normal (transverse) load and a tangential load.

Wind drag is generated on the portions of a structure that are above the still fluid surface defined by the fluid properties because these portions are exposed to the user-defined wind velocity profile.

Specifying distributed transverse fluid or wind drag loads

Distributed transverse drag is defined as follows (see "Drag, inertia, and buoyancy loading," Section 6.2.1 of the Abaqus Theory Guide, for more details):

$$\mathbf{F}_D = A\frac{1}{2}\rho\, C_D D \Delta\mathbf{v}_{fn}|\Delta\mathbf{v}_{fn}|,$$

where

- $\mathbf{F}_D$ is the force per unit length, transverse to the member;
- $A(t)$ is the current value of the amplitude curve referred to by the distributed load definition, multiplied by the user-defined magnitude factor, ***M***;
- ρ is the mass density of the fluid (given in the fluid properties) for fluid distributed drag or is the mass density of the air (given in the wind velocity profile) for wind distributed drag;
- C_D is the drag coefficient; and
- ***D*** is the effective outer diameter of the member.

The relative fluid particle velocity in the normal direction, $\Delta\mathbf{v}_{fn}$, is given by

$$\Delta\mathbf{v}_{fn} = \Delta\mathbf{v} - \Delta\mathbf{v}\cdot\mathbf{t}\,\mathbf{t},$$
$$\Delta\mathbf{v} = \mathbf{v}_f - \alpha_R\mathbf{v}_p,$$

where

- $\mathbf{v}_f$ is the fluid particle velocity (see the discussion below);
- $\mathbf{v}_p$ is the velocity of this point on the structure (zero during static steps);

α_R is the structural velocity factor; and

$\mathbf{t}$ is the unit vector along the axis of the element.

The effective outer diameter of the element, ***D***; the drag coefficient, C_D; and the structural velocity factor, α_R, must be defined in the distributed load definition together with the distributed load type (fluid distributed drag or wind distributed drag).

The velocities due to steady current and waves can be scaled individually for fluid distributed drag by referring to different amplitude curves. Thus, the fluid particle velocity, $\mathbf{v}_f$, at any time is

$$\mathbf{v}_f = A_c\mathbf{v}_c + A_w\mathbf{v}_w,$$

where

$A_c(t)$ is the current value of the first amplitude curve listed in the load definition or 1.0 if the amplitude reference is omitted,

$\mathbf{v}_c$ is the steady current velocity defined in the fluid properties,

$A_w(t)$ is the current value of the second amplitude curve listed in the load definition or 1.0 if the amplitude reference is omitted, and

$\mathbf{v}_w$ is the user-defined wave velocity.

The wind velocity is defined in components relative to the local axes $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ defined for the wind velocity profile. Each velocity component can be scaled independently by referring to different amplitude curves. The total wind velocity at any time, $\mathbf{v}_f$, is

$$\mathbf{v}_f = (c_x A_x \hat{\mathbf{i}} + c_y A_y \hat{\mathbf{j}}) \left(\frac{z}{z_0}\right)^{\alpha},$$

where $A_x(t)$ and $A_y(t)$ are the amplitude references provided in the load definition for the velocity components in the local ***x***- and ***y***-directions, respectively. The values of c_x, c_y, z_0, and α are defined by the wind velocity profile; and ***z*** is the distance above the still fluid surface.

Input File Usage: Use the following option to define fluid distributed drag:

*DLOAD
element number or set, FDD, ***M***, ***D***, C_D, α_R, $A_c(t)$, $A_w(t)$

Use the following option to define wind distributed drag:

*DLOAD
element number or set, WDD, ***M***, ***D***, C_D, α_R, $A_x(t)$, $A_y(t)$

Specifying distributed tangential fluid drag loads

Distributed tangential fluid loading is a load in the tangential direction of an element due to skin friction. This type of loading is defined as follows (see "Drag, inertia, and buoyancy loading," Section 6.2.1 of the Abaqus Theory Guide, for more details):

$$\mathbf{F}_t = A\frac{1}{2}\rho_w C_t \pi D \Delta\mathbf{v}_{ft}|\Delta\mathbf{v}_{ft}|^{h-1},$$

where

$\mathbf{F}_t$ is the force per unit length, tangent to the member;

$A(t)$ is the amplitude curve referred to by the distributed load definition, multiplied by the user-defined magnitude factor, ***M***;

ρ_w is the mass density of the fluid (given in the fluid properties);

C_t is the tangential drag coefficient;

D is the effective outer diameter of the member; and

h is a constant (by default, $h = 2$, for quadratic dependence of force on velocity).

The relative fluid particle velocity in the tangential direction, $\Delta\mathbf{v}_{ft}$, is given by

$$\Delta\mathbf{v}_{ft} = (\mathbf{v}_f - \alpha_R\mathbf{v}_p)\cdot\mathbf{tt},$$

where

$\mathbf{v}_f$ is the fluid particle velocity (as defined above for distributed transverse fluid drag loading),

$\mathbf{v}_p$ is the velocity of this point on the structure (zero during static steps),

α_R is the structural velocity factor, and

$\mathbf{t}$ is the unit vector along the axis of the element.

The effective outer diameter of the element, ***D***; the drag coefficient, C_t; the structural velocity factor, α_R; and the exponent, ***h***, must be defined in the distributed load definition together with the distributed load type (fluid drag tangential).

As with distributed transverse fluid loading, the velocities due to steady current and waves (A_c and A_w) can be scaled individually by referring to different amplitude curves.

Input File Usage: Use the following option to define fluid drag tangential:

∗DLOAD
element number or set, FDT, ***M***, ***D***, C_t, α_R, ***h***, $A_c(t)$, $A_w(t)$

Specifying concentrated fluid or wind drag loads using a concentrated load definition

Concentrated fluid or wind drag loading applies a load normal to the end of an element. Such loading is automatically considered to be a follower force in geometrically nonlinear analysis (for elements that have rotational degrees of freedom).

The drag theory uses Morison's equation (see "Drag, inertia, and buoyancy loading," Section 6.2.1 of the Abaqus Theory Guide). The drag force is nonzero when the net flow is in the opposite direction of the outward normal to the exposed area and is zero when the net flow is in the direction of the normal:

$$\mathbf{F}_{\text{drag}} = \begin{cases} -A\frac{1}{2}\rho C_n \Delta A(\Delta v_{ft})^2\mathbf{t} & \text{for} \quad \Delta v_{ft} \le 0, \\ 0 & \text{for} \quad \Delta v_{ft} > 0, \end{cases}$$

where

$A(t)$ is the amplitude curve referenced by the concentrated load definition multiplied by the user-defined magnitude factor, ***M***;

ρ is the mass density of the fluid (given in the fluid properties) for transition section fluid drag or is the mass density of the air (given in the wind velocity profile) for transition section wind drag;

C_n is the drag coefficient;

ΔA is the exposed area; and

Δv_{ft} is the relative velocity between the structural member and the fluid particle along $\mathbf{t}$ and is given by $\mathbf{t} \cdot \Delta \mathbf{v}_{ft}$, where $\Delta \mathbf{v}_{ft} = (\mathbf{v}_f - \alpha_R \mathbf{v}_p) \cdot \mathbf{tt}$, as defined above for distributed tangential fluid drag loading.

The exposed area, ΔA; the drag coefficient, C_n; and the structural velocity factor, α_R, must be defined in the concentrated load definition together with the concentrated load type (transition section fluid drag or transition section wind drag).

As with distributed transverse fluid loading, the velocities due to steady current and waves (A_c and A_w) and the velocity components of the wind in the $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ directions (A_x and A_y) can be scaled individually by referring to different amplitude curves.

Input File Usage: Use the following option to define transition section fluid drag:

∗CLOAD
node number or set, TFD, ***M***, ΔA, C_n, α_R, $A_c(t)$, $A_w(t)$

Use the following option to define transition section wind drag:

∗CLOAD
node number or set, TWD, ***M***, ΔA, C_n, α_R, $A_x(t)$, $A_y(t)$

Specifying concentrated fluid or wind drag loads using a distributed load definition

You can apply concentrated fluid or wind drag loading on the ends of elements. These loads have the same effect as specifying a concentrated load at a node using a concentrated load definition with concentrated load type transition section fluid drag or transition section wind drag, except that the normal to the exposed area cannot be specified when a distributed load definition is used; the normal to the end of the element is defined by the tangent to the element.

The load can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element. These loads are nonzero only when the net flow is in the opposite direction of the outward normal to the exposed area.

The loading is exactly the same as that described for the concentrated fluid or wind drag loading applied with a concentrated load definition. The “distributed” form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid drag on the first end of the element:

∗DLOAD
element number or set, FD1, ***M***, ΔA, ***C***, α_R, $A_c(t)$, $A_w(t)$

Use the following option to define fluid drag on the second end of the element:

∗DLOAD
element number or set, FD2, ***M***, ΔA, ***C***, α_R, $A_c(t)$, $A_w(t)$

Use the following option to define wind drag on the first end of the element:

∗DLOAD
element number or set, WD1, ***M***, ΔA, ***C***, α_R, $A_x(t)$, $A_y(t)$

Use the following option to define wind drag on the second end of the element:

∗DLOAD
element number or set, WD2, ***M***, ΔA, ***C***, α_R, $A_x(t)$, $A_y(t)$

Neglecting the wave's contribution to drag and inertia loading during a step

If the wave's contribution to the drag and inertia loading should not be applied during a step, the concentrated or distributed load component definition must explicitly refer to an amplitude curve with a value of zero. This is the only way to prevent waves from contributing to the fluid velocities and accelerations used in the calculation of these concentrated or distributed load types.

Fluid inertia loads (added-mass effects)

Fluid inertia loading causes a structure to have increased inertial resistance to acceleration. This fluid "added-mass" effect is included automatically in a direct-integration dynamic step when fluid inertia loading is applied. Concentrated or distributed added mass must be defined to include the added-mass effect in an eigenfrequency extraction step.

Specifying distributed fluid inertia loads in a direct-integration dynamic step

Distributed fluid inertia loading is defined as follows (see "Drag, inertia, and buoyancy loading," Section 6.2.1 of the Abaqus Theory Guide, for a more detailed description):

$$\mathbf{F}_I = A\rho_w \frac{\pi D^2}{4} \left[C_M \, \mathbf{a}_{fn} - C_A \, \mathbf{a}_{pn} \right],$$

where

- $\mathbf{F}_I$ is the force per unit length, transverse to the member, caused by fluid inertia;
- $A(t)$ is the amplitude curve referred to by the distributed load definition multiplied by the user-defined magnitude factor, ***M***;
- ρ_w is the mass density of the fluid (given in the fluid properties);
- ***D*** is the effective outer diameter of the member;
- C_M is the transverse fluid inertia coefficient;
- C_A is the transverse added-mass coefficient;
- $\mathbf{a}_{fn}$ is the transverse component of the fluid acceleration; and
- $\mathbf{a}_{pn}$ is the transverse component of the beam acceleration (zero during static steps).

The effective outer diameter, D; transverse fluid inertia coefficient, C_M; and transverse added-mass coefficient, C_A, must be defined in the distributed load definition together with the distributed load type (distributed fluid inertia).

The fluid acceleration, $\mathbf{a}_{fn}$, is calculated according to the user-defined gravity wave and is further scaled by the amplitude curve, A_w, referred to by the distributed load definition.

Input File Usage: Use the following option to define distributed fluid inertia in a dynamic step:

∗DLOAD
element number or set, FI, M, D, C_M, C_A, $A_w(t)$

Specifying distributed fluid inertia loads in an eigenfrequency extraction step

The added mass contribution due to distributed fluid inertia loading is

$$\rho_w \frac{\pi D^2}{4} C_A$$

per unit length of the member in the directions transverse to the axis of the member only, where

- ρ_w is the mass density of the fluid (given in the fluid properties),
- D is the effective outer diameter of the member, and
- C_A is the transverse added-mass coefficient.

Input File Usage: ∗D ADDED MASS
element number or set, FI, D, C_A

Specifying concentrated fluid inertia loads in a direct-integration dynamic step using a concentrated load definition

Concentrated fluid inertia loading is automatically considered to be a follower force (for elements that have rotational degrees of freedom).

The inertia term is calculated as a force in the current direction of the outward normal to the exposed surface area:

$$\mathbf{F}_I = A\rho_w \left(K_{ts} F_{1s} \mathbf{a}_{ft} - L_{ts} F_{2s} \mathbf{a}_{pt}\right),$$

where

- $\mathbf{F}_I$ is the point force caused by fluid inertia;
- $A(t)$ is the amplitude curve referenced by the concentrated load definition multiplied by the user-defined magnitude factor, M;
- ρ_w is the mass density of the fluid (given in the fluid properties);
- K_{ts} is the tangential inertia coefficient;
- F_{1s} is the fluid acceleration shape factor (of dimension L^3);
- L_{ts} is the tangential added-mass coefficient;
- F_{2s} is the structural acceleration shape factor (of dimension L^3);

$\mathbf{a}_{ft}$ is the fluid acceleration in the direction of the outward normal to the exposed surface; and

$\mathbf{a}_{pt}$ is the structural acceleration in the direction of the outward normal to the exposed surface (zero during static steps).

The tangential inertia coefficient, K_{ts}; the fluid acceleration shape factor, F_{1s}; the tangential added-mass coefficient, L_{ts}; and the structural acceleration shape factor, F_{2s}, are given in the concentrated load definition together with the concentrated load type (transition section inertia).

The fluid acceleration, $\mathbf{a}_{ft}$, is calculated according to the user-defined gravity wave and is further scaled by the amplitude curve, A_w, referred to by the concentrated load definition.

Input File Usage: Use the following option to define transition section inertia in a dynamic step:

∗CLOAD
node number or set, TSI, $\boldsymbol{M}$, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w(t)$

Specifying concentrated fluid inertia loads in a direct-integration dynamic step using a distributed load definition

You can apply concentrated fluid inertia loading at the ends of elements. These loads have the same effect as specifying a concentrated fluid added-inertia loading using a concentrated load definition with concentrated load type transition section inertia, except that the normal to the exposed area cannot be specified when a distributed load definition is used; the normal to the end of the element is defined by the tangent to the element.

The inertia loading can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element.

The loading is exactly the same as that described for the concentrated fluid inertia loading applied with a concentrated load definition. The “distributed” form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid inertia on the first end of the element in a dynamic step:

∗DLOAD
element number or set, FI1, $\boldsymbol{M}$, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w(t)$

Use the following option to define fluid inertia on the second end of the element in a dynamic step:

∗DLOAD
element number or set, FI2, $\boldsymbol{M}$, K_{ts}, F_{1s}, L_{ts}, F_{2s}, $A_w(t)$

Specifying concentrated fluid inertia effects in an eigenfrequency extraction step using a concentrated added mass definition

The added mass contribution due to concentrated fluid inertia loading in an eigenfrequency extraction step is

$$\rho_w L_{ts} F_{2s}$$

in the direction normal to the transition section area, where

ρ_w is the mass density of the fluid (given in the fluid properties),

L_{ts} is the tangential added-mass coefficient, and

F_{2s} is the structural acceleration shape factor (of dimension L^3).

Input File Usage: ∗C ADDED MASS
node number or set, TSI, L_{ts}, F_{2s}
direction cosines defining the outward normal of the exposed area

Specifying concentrated fluid inertia effects in an eigenfrequency extraction step using a distributed added mass definition

You can apply concentrated fluid inertia effects at the ends of elements. These loads have the same effect as specifying concentrated fluid inertia effects using a concentrated added mass definition with concentrated load type transition section inertia, but in this case the normal to the exposed area cannot be specified; the normal to the end of the element is defined by the tangent to the element.

The added mass can be applied to the first end (node) of the element or to the second end (node 2 or 3, as appropriate) of the element.

The effect is exactly the same as that described for the concentrated fluid inertia effects applied with a concentrated added mass definition. The "distributed" form of the loading is provided for convenience.

Input File Usage: Use the following option to define fluid inertia on the first end of the element in an eigenfrequency extraction step:

∗D ADDED MASS
element number or set, FI1, L_{ts}, F_{2s}

Use the following option to define fluid inertia on the second end of the element in an eigenfrequency extraction step:

∗D ADDED MASS
element number or set, FI2, L_{ts}, F_{2s}

Applying non-Aqua loads to the structure

Concentrated and distributed load definitions can also be used to apply concentrated and distributed forces that are not associated with wind, waves, or steady current to the structure. See "Concentrated loads," Section 34.4.2, and "Distributed loads," Section 34.4.3.

Predefined fields

The following predefined fields can be specified for the structure (not the fluid) in an Abaqus/Aqua analysis, as described in "Predefined fields," Section 34.6.1:

- Temperatures of nodes in the structure can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.

- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Material options

Any of the mechanical constitutive models in Abaqus can be used for modeling the structure in an Abaqus/Aqua analysis (see Part V, "Materials," for details on the material models available in Abaqus/Standard).

Elements

The fluid loads in an Abaqus/Aqua analysis cannot be applied to all element types. Only the beam, pipe, elbow, truss, and rigid beam elements in Abaqus/Standard and linear beam and pipe elements in Abaqus/Explicit can be used to subject a structure to general Abaqus/Aqua loading. The only load that can be applied to two-dimensional rigid surfaces (R3D3 and R3D4 elements) is hydrostatic buoyancy; and this loading can be applied only in Abaqus/Standard. Current, wave, and wind loading have no effect on rigid surfaces.

Jack-up foundation analysis

Abaqus/Standard provides element types JOINT2D and JOINT3D, which can be used to model elastic-plastic interaction between spud cans and the sea floor (see "Elastic-plastic joints," Section 32.10.1).

Output

In addition to the usual output variables available in Abaqus/Standard (see "Abaqus/Standard output variable identifiers," Section 4.2.1) and in Abaqus/Explicit (see "Abaqus/Explicit output variable identifiers," Section 4.2.2), element section output variable ESF1 can be used to request output of the effective axial force in a beam subjected to pressure loading (see "Beam element library," Section 29.3.8). The velocities and accelerations of the fluid cannot be output.

Input file template

```
*HEADING
…
*SURFACE SECTION,ELSET=aquaviz,AQUAVISUALIZATION=YES
*NSET,NSET=naquaviz,ELSET=aquaviz
*AQUA
Data lines defining the fluid properties and steady current velocity
*WAVE, TYPE=wave theory
Data lines defining gravity waves
**
*STEP (, NLGEOM)
Use the NLGEOM parameter to include nonlinear geometric effects
*DYNAMIC (or *STATIC or *DYNAMIC, EXPLICIT)
```

…
***CLOAD**
Data lines defining concentrated buoyancy, fluid/wind drag, and fluid inertia loads
***DLOAD**
Data lines defining distributed buoyancy, fluid/wind drag, and fluid inertia loads
OUTPUT, FIELD, TIME INTERVAL=interval for field output*
NODE OUTPUT,NSET=naquaviz*
U
***END STEP**
******
***STEP**
The NLGEOM parameter must have been included in the previous step to obtain the natural frequencies of the prestressed structure
***FREQUENCY**
…
***C ADDED MASS**
Data lines to define concentrated added-mass effects
***D ADDED MASS**
Data lines to define distributed added-mass effects
OUTPUT, FIELD, TIME INTERVAL=interval for field output*
NODE OUTPUT,NSET=naquaviz*
U
***END STEP**

6.12 Annealing

- “Annealing procedure,” Section 6.12.1

6.12.1 ANNEALING PROCEDURE

Products: Abaqus/Explicit Abaqus/CAE

References

- *ANNEAL
- "Configuring an annealing procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The anneal procedure:

- is used to anneal a structure by setting all appropriate state variables and velocities to zero; and
- is intended only for metal plasticity and user-defined material models; it has no effect on other material models.

The annealing process

The anneal procedure is intended to simulate the relaxation of stresses and plastic strains that occurs as metals are heated to high temperatures. Physically, annealing is the process of heating a metal part to a high temperature to allow the microstructure to recrystallize, removing dislocations caused by cold working of the material. During the anneal procedure Abaqus/Explicit sets all appropriate state variables to zero. These variables include stresses, backstresses, plastic strains, and velocities. In the case of metal porous plasticity, the void volume fraction is also set to zero, such that the material becomes fully dense.

There is no time scale in an annealing step; therefore, time does not advance. The annealing process occurs instantaneously. No data are required for the anneal procedure.

Input File Usage: *ANNEAL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Anneal**

Temperatures

Thermal strains are set to zero, and the temperature at all nodes in the model will be set to a uniform temperature or will be maintained at the current temperature during the anneal procedure. By default, the temperature at all nodes is maintained at the current temperature. You can specify a different final temperature, θ.

Input File Usage: *ANNEAL, TEMPERATURE=θ

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Anneal**: **Post-anneal reference temperature: Value** θ

Initial conditions

The initial state for the anneal step is the state of the model at the end of the last explicit dynamic analysis step.

Boundary conditions

It is not appropriate to specify new boundary conditions or to modify boundary conditions in an anneal procedure; all boundary conditions in effect prior to this procedure will remain fixed.

Loads

It is not meaningful to specify loads in an anneal procedure.

Predefined fields

It is not meaningful to specify predefined fields in an anneal procedure.

Material options

The annealing procedure is intended only for metal plasticity models ("Classical metal plasticity," Section 23.2.1) and user-defined materials modeled with user subroutines **VFABRIC** and **VUMAT**. The metal plasticity models in Abaqus/Explicit include Mises, Johnson-Cook, Hill, and metal porous plasticity. Abaqus/Explicit also allows annealing of elastic materials ("Linear elastic behavior," Section 22.2.1), including isotropic, orthotropic, and anisotropic elasticity. The annealing procedure has no effect on other material models.

Elements

All of the elements that are available in Abaqus/Explicit can be used in an anneal procedure. The elements are listed in Part VI, "Elements."

Output

There is no output associated with an anneal step.

Input file template

```
*HEADING
 …
**
*STEP
*DYNAMIC, EXPLICIT (,ADIABATIC) or
*DYNAMIC TEMPERATURE-DISPLACEMENT, EXPLICIT
```
Data line to specify the time period of the step

BOUNDARY, AMPLITUDE=name*
Data lines to describe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD**
Data lines to specify loads
***TEMPERATURE and/or *FIELD**
Data lines to specify values of predefined fields
***END STEP**
******
***STEP**
***ANNEAL (,TEMPERATURE=**θ**)**
***END STEP**
******
***STEP**
***DYNAMIC, EXPLICIT (,ADIABATIC)**
Data line to specify the time period of the step
BOUNDARY, AMPLITUDE=name*
Data lines to describe zero-valued or nonzero boundary conditions
***CLOAD and/or *DLOAD and/or *DSLOAD**
Data lines to specify loads
***TEMPERATURE and/or *FIELD**
Data lines to specify values of predefined fields
***END STEP**

7. Analysis Solution and Control

Solving nonlinear problems 7.1
Analysis convergence controls 7.2

7.1 Solving nonlinear problems

- "Solving nonlinear problems," Section 7.1.1

7.1.1 SOLVING NONLINEAR PROBLEMS

Products: Abaqus/Standard Abaqus/CAE

References

- "Convergence and time integration criteria: overview," Section 7.2.1
- "Commonly used control parameters," Section 7.2.2
- "Convergence criteria for nonlinear problems," Section 7.2.3
- "Time integration accuracy in transient problems," Section 7.2.4
- "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Solving nonlinear problems in Abaqus/Standard involves:

- a combination of incremental and iterative procedures;
- using the Newton method to solve the nonlinear equations;
- determining convergence;
- defining loads as a function of time; and
- choosing suitable time increments automatically.

Some static problems may become unstable because of severe nonlinearity. Abaqus/Standard offers a set of automatic stabilization mechanisms to handle such problems.

The solution of nonlinear problems

The nonlinear load-displacement curve for a structure is shown in Figure 7.1.1–1.

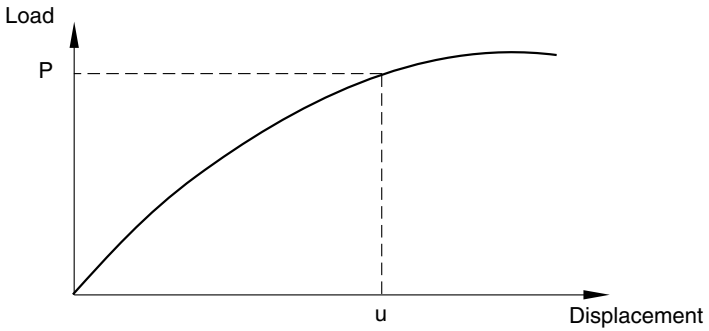


Figure 7.1.1–1 Nonlinear load-displacement curve.

The objective of the analysis is to determine this response. In a nonlinear analysis the solution cannot be calculated by solving a single system of linear equations, as would be done in a linear problem. Instead, the solution is found by specifying the loading as a function of time and incrementing time to obtain the nonlinear response. Therefore, Abaqus/Standard breaks the simulation into a number of *time increments* and finds the approximate equilibrium configuration at the end of each time increment. Using the Newton method, it often takes Abaqus/Standard several iterations to determine an acceptable solution to each time increment.

Steps, increments, and iterations

- The time history for a simulation consists of one or more *steps*. You define the steps, which generally consist of an analysis procedure, loading, and output requests. Different loads, boundary conditions, analysis procedures, and output requests can be used in each step. For example:
  Step 1: Hold a plate between rigid jaws.
  Step 2: Add loads to deform the plate.
  Step 3: Find the natural frequencies of the deformed plate.
- An *increment* is part of a step. In nonlinear analyses each step is broken into increments so that the nonlinear solution path can be followed. You suggest the size of the first increment, and Abaqus/Standard automatically chooses the size of the subsequent increments. At the end of each increment the structure is in (approximate) equilibrium and results are available for writing to the restart, data, results, or output database files.
- An *iteration* is an attempt at finding an equilibrium solution in an increment. If the model is not in equilibrium at the end of the iteration, Abaqus/Standard tries another iteration. With every iteration the solution that Abaqus/Standard obtains should be closer to equilibrium; however, sometimes the iteration process may diverge—subsequent iterations may move away from the equilibrium state. In that case Abaqus/Standard may terminate the iteration process and attempt to find a solution with a smaller increment size.

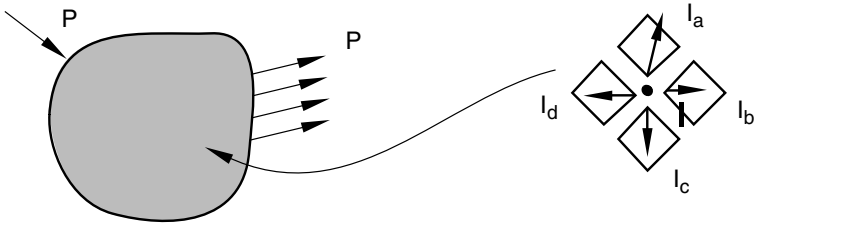
Convergence

Consider the external forces, P, and the internal (nodal) forces, I, acting on a body (see Figure 7.1.1–2(a) and Figure 7.1.1–2(b), respectively). The internal loads acting on a node are caused by the stresses in the elements that are attached to that node.

For the body to be in equilibrium, the net force acting at every node must be zero. Therefore, the basic statement of equilibrium is that the internal forces, I, and the external forces, P, must balance each other:

$$P - I = 0.$$

The nonlinear response of a structure to a small load increment, ΔP, is shown in Figure 7.1.1–3. Abaqus/Standard uses the structure's tangent stiffness, K_0, which is based on its configuration at u_0, and ΔP to calculate a *displacement correction*, c_a, for the structure. Using c_a, the structure's configuration is updated to u_a.



(a) External loads in a simulation. (b) Internal forces acting at a node.

Figure 7.1.1–2 Internal and external loads on a body.

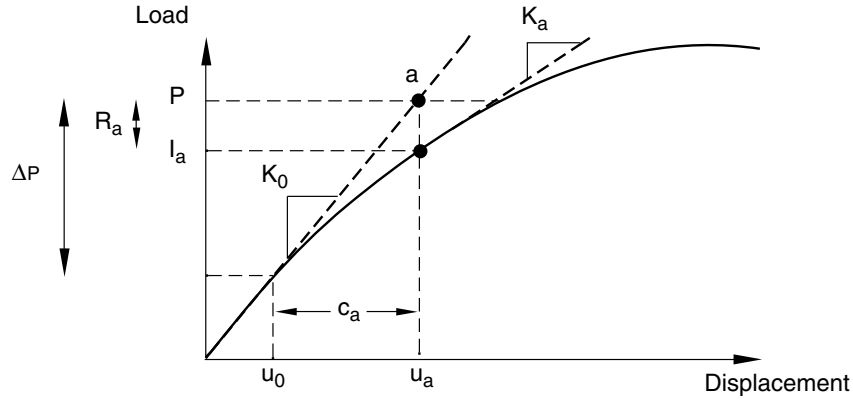


Figure 7.1.1–3 First iteration in an increment.

Abaqus/Standard then calculates the structure's internal forces, I_a, in this updated configuration. The difference between the total applied load, $\boldsymbol{P}$, and I_a can now be calculated as

$$R_a = P - I_a,$$

where R_a is the *force residual* for the iteration.

If R_a is zero at every degree of freedom in the model, point $\boldsymbol{a}$ in Figure 7.1.1–3 would lie on the load-deflection curve and the structure would be in equilibrium. In a nonlinear problem R_a will never be exactly zero, so Abaqus/Standard compares it to a tolerance value. If R_a is less than this force residual tolerance at all nodes, Abaqus/Standard accepts the solution as being in equilibrium. By default, this tolerance value is set to 0.5% of an average force in the structure, averaged over time. Abaqus/Standard automatically calculates this spatially and time-averaged force throughout the simulation. You can change this, and all other such tolerances, by specifying solution controls (see "Convergence criteria for nonlinear problems," Section 7.2.3).

If R_a is less than the current tolerance value, $\boldsymbol{P}$ and I_a are considered to be in equilibrium and u_a is a valid equilibrium configuration for the structure under the applied load. However, before Abaqus/Standard accepts the solution, it also checks that the last displacement correction, c_a, is small relative to the total incremental displacement, $\Delta u_a = u_a - u_0$. If c_a is greater than a fraction (1% by default) of the incremental displacement, Abaqus/Standard performs another iteration. Both convergence checks must be satisfied before a solution is said to have *converged* for that time increment.

If the solution from an iteration is not converged, Abaqus/Standard performs another iteration to try to bring the internal and external forces into balance. First, Abaqus/Standard forms the new stiffness, K_a, for the structure based on the updated configuration, u_a. This stiffness, together with the residual R_a, determines another displacement correction, c_b, that brings the system closer to equilibrium (point *b* in Figure 7.1.1–4).

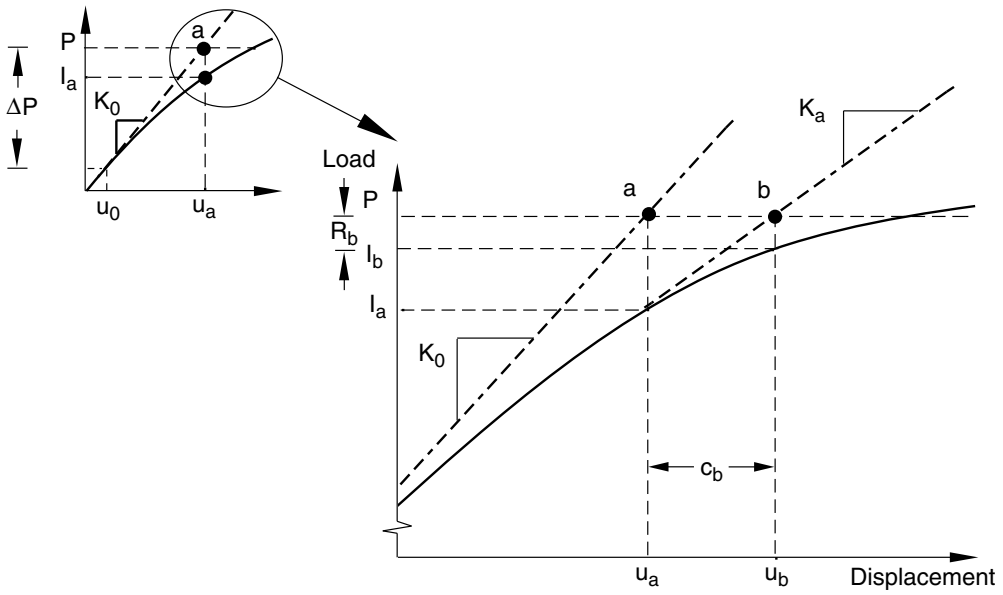


Figure 7.1.1–4 Second iteration.

Abaqus/Standard calculates a new force residual, R_b, using the internal forces from the structure's new configuration, u_b. Again, the largest force residual at any degree of freedom, R_b, is compared against the force residual tolerance, and the displacement correction for the second iteration, c_b, is compared to the increment of displacement, Δu_b. If necessary, Abaqus/Standard performs further iterations. For more details on convergence in Abaqus/Standard, see "Convergence criteria for nonlinear problems," Section 7.2.3.

For each iteration in a nonlinear analysis Abaqus/Standard forms the model's stiffness matrix and solves a system of equations. Therefore, the computational cost of each iteration is close to the cost of conducting a complete linear analysis, making the computational expense of a nonlinear

analysis potentially many times greater than the cost of a linear analysis. Since it is possible with Abaqus/Standard to save results at each converged increment, the amount of output data available from a nonlinear simulation can also be much greater than that available from a linear analysis of the same geometry.

Automatic incrementation control

By default, Abaqus/Standard automatically adjusts the size of the time increments to solve nonlinear problems efficiently. You need to suggest only the size of the first increment in each step of the simulation, after which Abaqus/Standard automatically adjusts the size of the increments. If you do not provide a suggested initial increment size, Abaqus/Standard will attempt to apply all of the loads defined in the step in a single increment. For highly nonlinear problems Abaqus/Standard will have to reduce the increment size repeatedly to obtain a solution, resulting in wasted CPU time. It is advantageous to provide a reasonable initial increment size because only in mildly nonlinear problems can all of the loads in a step be applied in a single increment.

The number of iterations needed to find a converged solution for a time increment will vary depending on the degree of nonlinearity in the system. With the default incrementation control, the procedure works as follows. If the solution has not converged within 16 iterations or if the solution appears to diverge, Abaqus/Standard abandons the increment and starts again with the increment size set to 25% of its previous value. It then attempts to find a converged solution with this smaller time increment. If the solution still fails to converge, Abaqus/Standard reduces the increment size again. This process is continued until a solution is found. If the time increment becomes smaller than the minimum you defined or more than 5 attempts are needed, Abaqus/Standard stops the analysis.

If the increment converges in fewer than 5 iterations, this indicates that the solution is being found fairly easily. Therefore, Abaqus/Standard automatically increases the increment size by 50% if 2 consecutive increments require fewer than 5 iterations to obtain a converged solution.

While the default automatic incrementation control is suitable for most analyses, you can change all the defaults when necessary by specifying solution controls; see "Commonly used control parameters," Section 7.2.2, and "Time integration accuracy in transient problems," Section 7.2.4.

Automatic stabilization of unstable problems

Nonlinear static problems can be unstable. Such instabilities may be of a geometrical nature, such as buckling, or of a material nature, such as material softening. If the instability manifests itself in a global load-displacement response with a negative stiffness, the problem can be treated as a buckling or collapse problem as described in "Unstable collapse and postbuckling analysis," Section 6.2.4. However, if the instability is localized, there will be a local transfer of strain energy from one part of the model to neighboring parts, and global solution methods may not work. This class of problems has to be solved either dynamically or with the aid of (artificial) damping; for example, by using dashpots.

Abaqus/Standard provides an automatic mechanism for stabilizing unstable quasi-static problems through the automatic addition of volume-proportional damping to the model. The applied damping factors can be constant over the duration of a step, or they can vary with time to account for changes over the course of a step. The latter, adaptive approach is typically preferred.

Automatic stabilization of static problems with a constant damping factor

Automatic stabilization with a constant damping factor is triggered by including automatic stabilization in any nonlinear quasi-static procedure. Viscous forces of the form

$$F_v = c\mathbf{M}^* v$$

are added to the global equilibrium equations

$$P - I - F_v = 0,$$

where $\mathbf{M}^*$ is an artificial mass matrix calculated with unity density, c is a damping factor, $v = \Delta u/\Delta t$ is the vector of nodal velocities, and Δt is the increment of time (which may or may not have a physical meaning in the context of the problem being solved).

For the case of static stabilization the mass matrix for Timoshenko beams is always calculated assuming isotropic rotary inertia, regardless of the type of rotary inertia specified for the beam section definition ("Rotary inertia for Timoshenko beams" in "Beam section behavior," Section 29.3.5).

Automatic stabilization does not carry over automatically to subsequent steps. It needs to be declared for any step in which you want it to be active. Abaqus/Standard recalculates new values for the damping factor, based on the declared damping intensity and on the solution of the first increment of the step. Therefore, unless you specify the same damping factor directly (see "Directly specifying the damping factor" below), an analysis with an unstable step may produce slightly different results from the same analysis with the original step split into two steps. Moreover, if the instabilities in the model have not subsided by the end of a step, viscous forces may be terminated abruptly or modified at the beginning of subsequent steps, potentially causing convergence difficulties if automatic stabilization is not used in the subsequent step. If such a situation arises, it is recommended that the problem be restarted with the damping factor set equal to the value chosen by Abaqus/Standard (or to the value you specified) in the previous step. This value is printed in the message (`.msg`) file for the previous step. If it is necessary to have an accurate static equilibrium solution after an instability has occurred (and the model's behavior has returned to a stable regime), the step with automatic stabilization can be followed by a step without such stabilization.

Calculating the damping factor based on the dissipated energy fraction

It is assumed that the problem is stable at the beginning of the step and that instabilities may develop in the course of the step. While the model is stable, viscous forces and, therefore, the viscous energy dissipated are very small. Thus, the additional artificial damping has no effect. If a local region goes unstable, the local velocities increase and, consequently, part of the strain energy then released is dissipated by the applied damping. Abaqus/Standard can, if necessary, reduce the time increment to permit the process to occur without the unstable response causing very large displacements. Abaqus/Standard calculates and prints to the message file the damping factor, c, based on the solution of the first increment of a step. In most applications the first increment of the step is stable without the need to apply damping. The damping factor is then determined in such a way that the dissipated energy for a given increment with characteristics similar to the first increment is a small fraction of the extrapolated strain energy. The

fraction is called the *dissipated energy fraction* and has a default value of 2.0×10^{-4}. If the default value for the dissipated energy fraction is used, the adaptive automatic stabilization scheme discussed in the next section will be activated automatically by default in the step.

Alternatively, you can specify the non-default dissipated energy fraction for automatic stabilization directly.

Input File Usage: Use any of the following options to specify a nondefault dissipated energy fraction:

∗COUPLED TEMPERATURE-DISPLACEMENT,
STABILIZE=*dissipated energy fraction*
∗SOILS, STABILIZE=*dissipated energy fraction*
∗STATIC, STABILIZE=*dissipated energy fraction*
∗STEADY STATE TRANSPORT, STABILIZE=*dissipated energy fraction*
∗VISCO, STABILIZE=*dissipated energy fraction*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: *any valid step type*: **Basic**: select **Specify dissipated energy fraction** from the **Automatic stabilization** field

Considerations when the first increment is unstable or singular

There are cases where the first increment is either unstable or singular (due to a rigid body mode). In such cases it is not possible to obtain a solution to the first increment without applying some damping. Therefore, some damping is already applied during the first increment. The damping factor used for the initial increment is chosen such that the average element damping matrix component, divided by the step time, is equal to the average element stiffness matrix component multiplied by the dissipated energy fraction. If the calculated strain energy change in this increment indicates that the solution without damping is stable, the damping factor is recalculated based upon the energy method described previously. However, if the strain energy change indicates that the solution is unstable or singular, the initially calculated damping factor is maintained, and a warning message is issued indicating that the amount of damping applied may not be appropriate. In many cases the amount of damping may actually be rather large, which can affect the solution in ways that are not desirable. Therefore, if the above mentioned warning message is issued, check the viscous forces (VF) and compare them with the expected nodal forces to make sure that the viscous forces do not dominate the solution. If necessary, follow the stabilized step with another step in which stabilization is not used or with a step in which a much smaller damping factor is used.

Directly specifying the damping factor

You can also specify the damping factor directly. Unfortunately, it is generally quite difficult to make a reasonable estimate for the damping factor unless a value is known from the output of previous runs; the damping factor depends not only on the amount of damping but also on mesh size and material behavior.

Input File Usage: Use any of the following options to specify the damping factor directly:

∗COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE,
FACTOR=*damping factor*
∗SOILS, STABILIZE, FACTOR=*damping factor*

∗STATIC, STABILIZE, FACTOR=*damping factor*
∗STEADY STATE TRANSPORT, STABILIZE, FACTOR=*damping factor*
∗VISCO, STABILIZE, FACTOR=*damping factor*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select **Specify damping factor** from the **Automatic stabilization** field

Adaptive automatic stabilization scheme

As discussed above, the automatic stabilization scheme with a constant damping factor typically works well to subside instabilities and to eliminate rigid body modes without having a major effect on the solution. However, there is no guarantee that the value of the damping factor is optimal or even suitable in some cases. This is particularly true for thin shell models, in which the damping factor may be too high when a poor estimation of the extrapolated strain energy is made during the first increment. For such models you may have to increase the damping factor if the convergence behavior is problematic or to decrease the damping factor if it distorts the solution. The former case would require you to rerun the analysis with a larger damping factor, while the latter case would require you to perform postanalysis comparison of the energy dissipated by viscous damping (ALLSD) to the total strain energy (ALLIE). Therefore, obtaining an optimal value for the damping factor is a manual process requiring trial and error until a converged solution is obtained and the dissipated stabilization energy is sufficiently small.

The adaptive automatic stabilization scheme, in which the damping factor can vary spatially and with time, provides an effective alternative approach. In this case the damping factor is controlled by the convergence history and the ratio of the energy dissipated by viscous damping to the total strain energy. If the convergence behavior is problematic because of instabilities or rigid body modes, Abaqus/Standard automatically increases the damping factor. For example, the damping factor may increase if an analysis takes extra severe discontinuity or equilibrium iterations per increment or requires time increment cutbacks. On the other hand, Abaqus/Standard may reduce the damping factor automatically if instabilities and rigid body modes subside.

The ratio of the energy dissipated by viscous damping to the total strain energy is limited by an accuracy tolerance that you specify. Such an accuracy tolerance is imposed on the global level for the whole model. If the ratio of the energy dissipated by viscous damping to the total strain energy for the whole model exceeds the accuracy tolerance, the damping factor at each individual element is adjusted to ensure that the ratio of the stabilization energy to the strain energy is less than the accuracy tolerance on both the global and local element level. The stabilization energy always increases, while the strain energy may decrease. Therefore, Abaqus/Standard restricts the ratio of the incremental value of the stabilization energy to the incremental value of the strain energy for each increment to ensure that this value has not exceeded the accuracy tolerance if the ratio of the total stabilization energy to the total strain energy exceeds the accuracy tolerance. The accuracy tolerance is a targeted value and can be exceeded in some situations, such as when there is rigid body motion or when significant non-local instability occurs.

The default accuracy tolerance used by the adaptive automatic stabilization scheme is 0.05. The default tolerance is suitable for most applications, but you have the option of specifying a nondefault accuracy tolerance if necessary. If the accuracy tolerance is set equal to zero, the adaptive automatic

stabilization scheme is not activated and the automatic stabilization scheme with a constant damping factor will be used in the step.

If the accuracy tolerance is not specified but the dissipated energy fraction with the default value of 2.0×10^{-4} is used, the adaptive automatic damping algorithm will be activated automatically with an accuracy tolerance of 0.05.

Input File Usage: Use any of the following options to activate adaptive automatic stabilization with the default stabilization energy tolerance:

*COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE
*SOILS, STABILIZE
*STATIC, STABILIZE
*STEADY STATE TRANSPORT, STABILIZE
*VISCO, STABILIZE

Use any of the following options to activate adaptive automatic stabilization with a nondefault stabilization energy tolerance:

*COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE, ALLSDTOL=*accuracy tolerance*
*SOILS, STABILIZE, ALLSDTOL=*accuracy tolerance*
*STATIC, STABILIZE, ALLSDTOL=*accuracy tolerance*
*STEADY STATE TRANSPORT, STABILIZE, ALLSDTOL=*accuracy tolerance*
*VISCO, STABILIZE, ALLSDTOL=*accuracy tolerance*

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select an **Automatic stabilization** method: toggle on **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *accuracy tolerance*

Default value of the initial damping factor

By default, the initial value of the damping factor is typically equal to the value that would be used for automatic stabilization with a constant damping factor (see "Calculating the damping factor based on the dissipated energy fraction" above). In some cases additional factors that are considered with adaptive automatic stabilization cause some differences in the initial damping factor.

Specifying the initial damping factor directly

Alternatively, you can specify the initial damping factor directly. The damping factor is adjusted based on the convergence history and the accuracy tolerance through the step.

Input File Usage: Use any of the following options to specify the initial damping factor directly with the default stabilization energy tolerance:

*COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
*SOILS, STABILIZE, FACTOR=*damping factor*, ALLSDTOL

*STATIC, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
*STEADY STATE TRANSPORT, STABILIZE, FACTOR=*damping factor*, ALLSDTOL
*VISCO, STABILIZE, FACTOR=*damping factor*, ALLSDTOL

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: from the **Automatic stabilization** field, select **Specify damping factor:** *damping factor*: toggle on **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *maximum ratio*

Propagating the damping factors from the immediately preceding general step into the current step

Adaptive automatic stabilization provides an option to propagate the damping factors from the immediately preceding general step to the subsequent steps. The default is to not propagate the damping factors from the results of the preceding general step. In this case Abaqus recalculates the initial damping factors based on the declared dissipated energy faction and on the solution of the first increment of the step, or you can specify the initial damping factors directly.

Input File Usage: Use any of the following options to indicate that the damping factors in the current step are propagated from the immediately preceding general step:

*COUPLED TEMPERATURE-DISPLACEMENT, STABILIZE, ALLSDTOL, CONTINUE=YES
*SOILS, STABILIZE, ALLSDTOL, CONTINUE=YES
*STATIC, STABILIZE, ALLSDTOL, CONTINUE=YES
*STEADY STATE TRANSPORT, STABILIZE, ALLSDTOL, CONTINUE=YES
*VISCO, STABILIZE, ALLSDTOL, CONTINUE=YES

Abaqus/CAE Usage: Step module: **Create Step**: **General**: **Coupled temp-displacement**, **Soils**, **Static, General**, or **Visco**: **Basic**: select **Use damping factors from previous general step** from the **Automatic stabilization** field: **Use adaptive stabilization with max. ratio of stabilization to strain energy:** *accuracy tolerance*

Ensuring that an accurate solution is obtained with automatic stabilization

Whenever automatic stabilization is applied to a problem, check the following to ensure that accurate solutions are obtained:

- For a damping factor calculated using the dissipated energy fraction, check the factor printed to the message (`.msg`) file at the end of the first increment to ensure that a reasonable amount of damping is applied. Unfortunately, the damping factor is problem dependent; therefore, you must rely on experience from previous runs.
- Compare the viscous forces (VF) with the overall forces in the analysis, and ensure that the viscous forces are relatively small compared with the overall forces in the model.

- Compare the viscous damping energy (ALLSD) with the total strain energy (ALLIE), and ensure that the ratio does not exceed the dissipated energy fraction or any reasonable amount. The viscous damping energy may be large if the structure undergoes a large amount of motion.

The automated procedure of computing damping factors works well for many applications. However, there are cases where the computed damping factor is either too small, thus not controlling the instability, or too high, thus leading to inaccurate results. These problems are more likely to occur when using a constant damping factor—the damping factor is computed in the first increment, which may not be representative of behavior in the rest of the step. For example, consider a sequentially coupled thermal-stress analysis in which a mechanical analysis reads temperatures from a previous transient thermal analysis. Typically the thermal analysis exhibits a diffusive process, where rapid changes in temperature occurs early in the analysis and minor changes in temperature occur once steady state is reached. In such a case Abaqus will compute the extrapolated strain energy based on the temperatures corresponding to the time of the first increment (in this case there may be a significant change in temperature for the first increment), thus yielding a larger then expected extrapolated strain energy. This in turn leads to a damping factor that is too large, resulting in inaccurate results.

If one of the automatic stabilization methods is not working appropriately, you can try using the other automatic stabilization method; the adaptive stabilization scheme is generally preferred. Alternatively, you can try directly specifying the damping factor.

7.2 Analysis convergence controls

- "Convergence and time integration criteria: overview," Section 7.2.1
- "Commonly used control parameters," Section 7.2.2
- "Convergence criteria for nonlinear problems," Section 7.2.3
- "Time integration accuracy in transient problems," Section 7.2.4

7.2.1 CONVERGENCE AND TIME INTEGRATION CRITERIA: OVERVIEW

Numerous control parameters are associated with the convergence and integration accuracy algorithms in Abaqus/Standard. These parameters are assigned default values that are chosen to optimize the accuracy and efficiency of the solution for a wide spectrum of nonlinear problems. You can change the solution control parameters, as described in the following sections:

- A brief synopsis of the more important solution control parameters, together with a description of the circumstances in which they can be used effectively, is provided in "Commonly used control parameters," Section 7.2.2. This section is likely to be the most useful for the general user and should be read first.
- Abaqus/Standard incorporates an empirical algorithm designed to solve the equilibrium equations of nonlinear systems accurately and economically. The criteria used to establish convergence of nonlinear increments and the automatic adjustment of increment size based on the convergence rate are described in "Convergence criteria for nonlinear problems," Section 7.2.3.
- Abaqus/Standard allows you to choose "time integration accuracy parameters" in problems that have a physical time scale. The algorithms that use these parameters for automatically controlling time increment sizes are described in "Time integration accuracy in transient problems," Section 7.2.4.
- Abaqus/CFD allows you to choose the control parameters used in an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation to alleviate instability and mesh distortion during the analysis.

Modifying the default solution controls

The default values for the solution control parameters need not be adjusted for most cases. You can reset them, however, within a step definition.

Values given for the solution control parameters remain in effect for the remainder of the analysis or until they are reset.

Input File Usage: *CONTROLS

The *CONTROLS option can be repeated, if necessary, with different parameters.

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**

Resetting all default solution controls

You can restore all solution control parameters to their default values.

Input File Usage: *CONTROLS, RESET

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Reset all parameters to their system-defined defaults**

7.2.2 COMMONLY USED CONTROL PARAMETERS

Products: Abaqus/Standard Abaqus/CFD Abaqus/CAE

References

- "Convergence and time integration criteria: overview," Section 7.2.1
- *CONTROLS
- "Customizing general solution controls," Section 14.15.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Solution control parameters can be used to control:

- nonlinear equation solution accuracy;
- time increment adjustment; and
- FSI stabilization and mesh distortion in an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation.

These solution control parameters need not be changed for most analyses. In difficult cases, however, the solution procedure may not converge with the default controls or may use an excessive number of increments and iterations. After it has been established that such problems are not due to modeling errors, it may be useful to change certain control parameters.

This section presents a brief synopsis of the more important solution control parameters, together with a description of the circumstances in which they can be used effectively.

Values given for the solution control parameters remain in effect for the remainder of the analysis or until they are reset. You can restore all solution control parameters to their default values (see "Convergence and time integration criteria: overview," Section 7.2.1).

Terminology

In this section the word "flux" means the variable whose discretized equilibrium is being sought and for which the equilibrium equations may be nonlinear: force, moment, heat flux, concentration volumetric flux, or pore liquid volumetric flux. The word "field" refers to the basic variables of the system, such as the components of the displacement in a continuum stress analysis or temperature in a heat transfer analysis. The superscript α refers to one such type of equation. The fields and corresponding fluxes available in Abaqus/Standard are listed in "Convergence criteria for nonlinear problems," Section 7.2.3.

Defining tolerances for field equations

Solution control parameters can be used to define tolerances for field equations. You can select the type of equation for which the solution control parameters are being defined, as shown in Table 7.2.2–1. The default tolerances can be reset if the analysis does not require high accuracy in the convergence criteria.

Table 7.2.2–1 Selecting the field equation. The input file parameters are followed by the corresponding Abaqus/CAE conditions in parentheses.

Equilibrium equation	Input file (Abaqus/CAE)	DOF
All active fields	FIELD=GLOBAL **(Apply to all applicable fields)**	all
Force and bimoment	FIELD=DISPLACEMENT **(Displacement)**	1, 2, 3, 7
Moment	FIELD=ROTATION **(Rotation)**	4, 5, 6
Heat transfer	FIELD=TEMPERATURE **(Temperature)**	11, 12, 13, ...
Hydrostatic fluid	FIELD=HYDROSTATIC FLUID PRESSURE **(Hydrostatic Fluid Pressure)**	8
Pore fluid pressure	FIELD=PORE FLUID PRESSURE **(Pore Fluid Pressure)**	8
Mass diffusion	FIELD=CONCENTRATION **(Concentration)**	11
Electrical conduction	FIELD=ELECTRICAL POTENTIAL **(Electrical Potential)**	9
Mechanism analysis (connector elements with material flow degree of freedom)	FIELD=MATERIAL FLOW **(Unsupported)**	10

Equilibrium equation	Input file (Abaqus/CAE)	DOF
Analysis containing C3D4H elements (all materials, except compressible hyperelastic elastomers and elastomeric foams).	FIELD=PRESSURE LAGRANGE MULTIPLIER **(Unsupported)**	N/A
Analysis containing C3D4H elements with compressible hyperelastic or hyperfoam materials.	FIELD=VOLUMETRIC LAGRANGE MULTIPLIER **(Unsupported)**	N/A

The most significant solution control parameters for field equation tolerances—R_n^α, C_n^α, $\tilde{q}_0^\alpha$, and $\tilde{q}_u^\alpha$—may have to be modified in cases where the residuals are large relative to the fluxes or in cases where the incremental solution is essentially zero.

Input File Usage: *CONTROLS, PARAMETERS=FIELD, FIELD=*field*

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Field Equations**: **Apply to all applicable fields** or **Specify individual fields**: *field*

Modifying the residual control

R_n^α is the convergence criterion for the ratio of the largest residual to the corresponding average flux norm, $\tilde{q}^\alpha$, for convergence. $\tilde{q}^\alpha$ is defined in "Convergence criteria for nonlinear problems," Section 7.2.3. The default value is $R_n^\alpha = 5 \times 10^{-3}$, which is rather strict by engineering standards but in all but exceptional cases will guarantee an accurate solution to complex nonlinear problems. The value for this ratio can be increased to a larger number if some accuracy can be sacrificed for computational speed.

Modifying the solution correction control

C_n^α is the convergence criterion for the ratio of the largest solution correction to the largest corresponding incremental solution value. The default value is $C_n^\alpha = 10^{-2}$. In addition to sufficiently small residuals, Abaqus/Standard requires that the largest correction to the solution value be small in comparison to the largest corresponding incremental solution value. Some analyses may not require such accuracy, thus permitting this ratio to be increased. To avoid testing the magnitude of the solution correction, you can set C_n^α to 1.0.

Specifying the average flux

$\tilde{q}^\alpha$ is the value of average flux used by Abaqus/Standard for checking residuals. The default value is the time average flux calculated by Abaqus/Standard, as defined in "Convergence criteria for nonlinear problems," Section 7.2.3. You may, however, define a constant value, $\tilde{q}_u^\alpha$, for the average flux, in which case $\tilde{q}^\alpha = \tilde{q}_u^\alpha$ throughout the step.

You may wish to use absolute tolerances for your residual checks. The absolute tolerance value is then equal to the product of the average flux, $\tilde{q}_u^\alpha$, and the ratio R_n^α.

Modifying the initial time average flux

$\tilde{q}_0^\alpha$ is the initial value of the time average flux for the current step. The default value is the time average flux from the previous step or 10^{-2} if this is Step 1. Redefining $\tilde{q}_0^\alpha$ is sometimes helpful when a coupled problem is analyzed and some of the fields in the problem are not active in the first step; for example, if a static step is carried out before a fully coupled thermal-stress step.

Redefinition of $\tilde{q}_0^\alpha$ can also be useful if the first step is essentially a null step; for example, in a contact problem before any contact occurs, the initial fluxes (forces) generated are zero. In such cases $\tilde{q}_0^\alpha$ should be given as a typical flux magnitude that will occur when field α first becomes active.

The initial value of $\tilde{q}^\alpha$ is retained until an iteration is completed for which $\bar{q}^\alpha > \epsilon^\alpha \tilde{q}^\alpha$, at which time we redefine $\tilde{q}^\alpha = \bar{q}^\alpha$. This new $\tilde{q}^\alpha$ can become less than $\tilde{q}_0^\alpha$. The criterion for zero flux compared to $\tilde{q}^\alpha$ is $\bar{q}^\alpha \le \tilde{q}^\alpha \cdot \epsilon^\alpha$ (see "Convergence criteria for nonlinear problems," Section 7.2.3).

If you specify the average flux, $\tilde{q}_u^\alpha$, directly, the value given for $\tilde{q}_0^\alpha$ is ignored.

Abaqus/Standard output

The controls in effect for an analysis are listed in the data (**.dat**) and message (**.msg**) files. Nondefault controls are marked by ***. For example, specifying the following controls:

Field Equation	R_n^α	C_n^α	$\tilde{q}_0^\alpha$	$\tilde{q}_u^\alpha$	R_P^α	ϵ^α
Displacement	0.01	1.0	10.0	–	–	1.E−4
Rotation	0.02	2.0	20.0	2.E3	–	–

would result in the following output:

```
CONVERGENCE TOLERANCE PARAMETERS FOR FORCE
*** CRIT. FOR RESIDUAL FORCE FOR A NONLINEAR PROBLEM       1.000E-02
*** CRITERION FOR DISP. CORRECTION IN A NONLINEAR PROBLEM        1.00
*** INITIAL VALUE OF TIME AVERAGE FORCE                         10.0
    AVERAGE FORCE IS TIME AVERAGE FORCE
    ALT. CRIT. FOR RESIDUAL FORCE FOR A NONLINEAR PROBLEM  2.000E-02
*** CRIT. FOR ZERO FORCE RELATIVE TO TIME AVRG. FORCE       1.000E-04
    CRIT. FOR DISP. CORRECTION WHEN THERE IS ZERO FLUX      1.000E-03
    CRIT. FOR RESIDUAL FORCE WHEN THERE IS ZERO FLUX        1.000E-08
    FIELD CONVERSION RATIO                                       1.00
CONVERGENCE TOLERANCE PARAMETERS FOR MOMENT
 *** CRIT. FOR RESIDUAL MOMENT FOR A NONLINEAR PROBLEM     2.000E-02
 *** CRIT. FOR ROTATION CORRECTION IN A NONLINEAR PROBLEM        2.00
 *** INITIAL VALUE OF TIME AVERAGE MOMENT                        20.0
 *** USER DEFINED VALUE OF AVERAGE MOMENT NORM              2.000E+03
```

```
ALT. CRIT. FOR RESID. MOMENT FOR A NONLINEAR PROBLEM  2.000E-02
CRIT. FOR ZERO MOMENT RELATIVE TO TIME AVRG. MOMENT   1.000E-05
CRIT. FOR ROTATION CORRECTION WHEN ZERO FLUX          1.000E-03
CRIT. FOR RESIDUAL MOMENT WHEN ZERO FLUX              1.000E-08
FIELD CONVERSION RATIO                                     1.00
```

Controlling the time incrementation scheme

Solution control parameters can be used to alter both the convergence control algorithm and the time incrementation scheme. The time incrementation parameters I_0 and I_R are the most significant since they have a direct effect on convergence. They may have to be modified if convergence is (initially) nonmonotonic or if convergence is nonquadratic.

Nonmonotonic convergence may occur if various nonlinearities interact; for example, the combination of friction, nonlinear material behavior, and geometric nonlinearity may lead to nonmonotonically decreasing residuals.

Nonquadratic convergence will occur if the Jacobian is not exact, which may occur for complex material models. It may also occur if the Jacobian is nonsymmetric but the symmetric equation solver is used. In that case the unsymmetric equation solver should be specified for the step (see "Defining an analysis," Section 6.1.2).

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**

Specifying the equilibrium iteration for a residual check

I_0 is the number of equilibrium iterations after which the check is made that the residuals are not increasing in two consecutive iterations. The default value is $I_0 = 4$. If the initial convergence is nonmonotonic, it may be necessary to increase this value.

Specifying the equilibrium iteration for a logarithmic rate of convergence check

I_R is the number of equilibrium iterations after which the logarithmic rate of convergence check begins. The default value is $I_R = 8$. In cases where convergence is nonquadratic and this cannot be corrected by using the unsymmetric equation solver for the step, the logarithmic convergence check should be eliminated by setting this parameter to a high value.

Avoiding premature cutbacks in difficult analyses

Sometimes it is useful to increase both I_0 and I_R. For example, in a difficult analysis involving both friction and the concrete material model, it may be helpful to set $I_0 = 8$ and $I_R = 10$ to avoid premature cutbacks of the time increment. These two parameters can be raised to more appropriate values for severely discontinuous problems by increasing them individually.

Automatically setting the time incrementation parameters

You can automatically set the parameters described above to the values $I_0 = 8$ and $I_R = 10$. In this case any values that you specified previously for I_0 and I_R are overridden. However, if I_0 and I_R are specified multiple times in a step with different solution control settings, the last definition will be used.

Input File Usage: *CONTROLS, ANALYSIS=DISCONTINUOUS

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**: **Discontinuous analysis**

Improving solution efficiency in a problem that involves a high coefficient of friction

The solution efficiency can sometimes be improved in an analysis that involves a high coefficient of friction by automatically setting the time incrementation parameters and using the unsymmetric equation solver.

Abaqus/Standard output

The controls in effect for an analysis are listed in the data (**.dat**) and message (**.msg**) files. Nondefault controls are marked by **. For example, specifying the time incrementation parameters I_0=7 and I_R=10 would result in the following output:

```
TIME INCREMENTATION CONTROL PARAMETERS:
 *** FIRST EQUIL. ITERATION FOR CONSECUTIVE DIVERGENCE CHECK        7
 *** EQUIL. ITER. AT WHICH LOG. CONVERGENCE RATE CHECK BEGINS      10
     EQUIL. ITER. AFTER WHICH ALTERNATE RESIDUAL IS USED            9
     MAXIMUM EQUILIBRIUM ITERATIONS ALLOWED                        16
     EQUIL. ITERATION COUNT FOR CUT-BACK IN NEXT INCREMENT         10
     MAX EQUIL. ITERS IN TWO INCREMENTS FOR TIME INC. INCREASE      4
     MAXIMUM ITERATIONS FOR SEVERE DISCONTINUITIES                 12
     MAXIMUM CUT-BACKS ALLOWED IN AN INCREMENT                      5
     MAX DISCON. ITERS IN TWO INCS FOR TIME INC. INCREASE           6
     CUT-BACK FACTOR AFTER DIVERGENCE                           0.250
     CUT-BACK FACTOR FOR TOO SLOW CONVERGENCE                   0.500
     CUT-BACK FACTOR AFTER TOO MANY EQUILIBRIUM ITERATIONS      0.750
```

Activating the “line search” algorithm

In strongly nonlinear problems the Newton algorithms used in Abaqus/Standard may sometimes diverge during equilibrium iteration. The line search algorithm (discussed in “Improving the efficiency of the solution by using the line search algorithm” in “Convergence criteria for nonlinear problems,” Section 7.2.3) detects these situations automatically and applies a scale factor to the computed solution correction, which helps to prevent divergence. The line search algorithm is particularly useful when the quasi-Newton method (see “Solution method” in “Convergence criteria for nonlinear problems,” Section 7.2.3) is used.

By default, the line search algorithm is enabled only during steps where the quasi-Newton method is used. Set the maximum number of line search iterations, N^{ls}, to a reasonable value (such as 5) to activate the line search procedure or to zero to forcibly deactivate the line search.

Input File Usage: *CONTROLS, PARAMETERS=LINE SEARCH
N^{ls}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Line Search Control**: N^{ls}

Defining tolerances for constraint equations

Solution control parameters can be used to set tolerances for constraint equations. You can set strain compatibility tolerances for hybrid elements, displacement and rotation compatibility tolerances for distributing coupling constraints (specified as surface-based constraints or using DCOUP2D/DCOUP3D elements), and compatibility tolerances for softened contact. See "Convergence criteria for nonlinear problems," Section 7.2.3, for details.

Controlling the solution accuracy in direct cyclic analysis

Solution control parameters can be used in direct cyclic analysis to specify when to impose the periodicity conditions and to set tolerances for stabilized state and plastic ratchetting detections.

Input File Usage: *CONTROLS, TYPE=DIRECT CYCLIC
I_{PI}, CR_n^α, CU_n^α, CR_0^α, CU_0^α

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Direct Cyclic**: I_{PI}, CR_n^α, CU_n^α, CR_0^α, CU_0^α

Imposing the periodicity condition

You can specify the iteration number at which the periodicity condition is first imposed, I_{PI}. The default value is I_{PI} = 1, in which case the periodicity condition is imposed for all iterations from the beginning of an analysis. This solution control parameter rarely needs to be reset from its default value.

Defining tolerances for stabilized state and plastic ratchetting detections

You can specify the stabilized state detection criteria, CR_n^α and CU_n^α. CR_n^α is the maximum allowable ratio of the largest residual coefficient on any terms in the Fourier series to the corresponding average flux norm, and CU_n^α is the maximum allowable ratio of the largest correction to the displacement coefficient on any terms in the Fourier series to the largest displacement coefficient. The default values are CR_n^α = 5×10^{-3} and CU_n^α = 5×10^{-3}. The solution converges to a stabilized state if both these criteria are satisfied.

If plastic ratchetting occurs, the shape of the stress-strain curves remains unchanged but the mean value of the plastic strain over a cycle continues to shift from one iteration to the next. In that case it is desirable to use separate tolerances for the constant term in the Fourier series to detect the plastic ratchetting.

You can also specify the plastic ratchetting detection criteria, CR_0^α and CU_0^α. CR_0^α is the maximum allowable ratio of the largest residual coefficient on the constant term in the Fourier series to the corresponding average flux norm, and CU_0^α is the maximum allowable ratio of the largest correction to the displacement coefficient on the constant term in the Fourier series to the largest displacement coefficient. The default values are $CR_0^\alpha = 5 \times 10^{-3}$ and $CU_0^\alpha = 5 \times 10^{-3}$. Plastic ratchetting is expected if the residual coefficients and the corrections to the displacement coefficients on any of the periodic terms are within the tolerances set by CR_n^α and CU_n^α, respectively, but the maximum residual coefficient on the constant term and the maximum correction to the displacement coefficient on the constant term exceed the tolerances set by CR_0^α and CU_0^α, respectively.

Abaqus/Standard output

The controls in effect for an analysis are listed in the data (**.dat**) and message (**.msg**) files. Nondefault controls are marked by **. For example, specifying the following controls:

I_{PI}	CR_n^α	CU_n^α	CR_0^α	CU_0^α
5	1.0E−4	1.0E−4	1.0E−4	1.E−4

would result in the following output:

```
STABILIZED STATE AND PLASTIC RATCHETTING DETECTION
PARAMETERS FOR FORCE
** CRIT. FOR RESI. COEFF. ON ANY FOURIER TERMS            1.0E-04
** CRIT. FOR CORR. TO DISP. COEFF. ON ANY FOURIER TERMS   1.0E-04
** CRIT. FOR RESI. COEFF. ON CONSTANT FOURIER TERM        1.0E-04
** CRIT. FOR CORR. TO DISP. COEFF. ON CONST. FOURIER TERM 1.0E-04

PERIODICITY CONDITION CONTROL PARAMETER:
** ITERATION NUMBER AT WHICH PERIODICITY CONDITION
** STARTS TO IMPOSE                                          5
```

Controlling the solution accuracy and mesh quality in a deforming-mesh analysis with Abaqus/CFD

Solution control parameters can be used to control the mesh motion and to maintain the mesh quality in deforming-mesh problems involving moving boundaries or deforming geometries. They can also be used to control FSI stabilization when performing Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation.

Controlling mesh smoothing and FSI stabilization

When the implicit algorithm (default) for mesh smoothing is used, you can specify the number of iterations before performing a convergence check, the maximum number of iterations, the FSI penalty

scale factor, the solid/fluid density ratio, the linear convergence criterion, and the stiffness scale factor to control mesh motion and FSI stabilization.

The implicit algorithm uses the matrix-free, iteration method to solve the pseudo-elastic problem. The number of iterations and the linear convergence criterion control accuracy when solving the linear elasticity equations during the ALE process for FSI or deforming-mesh problems. Reducing the number of iterations or relaxing the linear convergence criterion can help reduce computational time. Similarly, increasing the number of iterations or the linear convergence criterion can help to ensure that the mesh quality remains good. The stiffness scale factor can be used to scale the elastic stiffness. Decreasing the elastic stiffness produces an ALE mesh with more local deformation.

When the explicit algorithm for mesh smoothing is used, you can specify the minimum number of mesh smoothing increments, the maximum number of mesh smoothing increments, the FSI penalty scale factor, the solid/fluid density ratio, and the stiffness scale factor to control the mesh motion and FSI stabilization.

The minimum and maximum number of mesh smoothing increments controls the number of mesh smoothing steps taken during the ALE process for FSI or deforming-mesh problems. Reducing the minimum and maximum number of mesh smoothing increments can help reduce computational time. Similarly, increasing the minimum/maximum number of smoothing increments helps to ensure that the mesh quality remains good and avoids potential element collapse during the evolution of a deforming-mesh problem.

The FSI penalty scale factor is used to control FSI stabilization and has a default value of 1.0. Increasing this parameter in increments of 0.1 may be necessary for extremely flexible structures in high-density fluids when the structural accelerations are high.

The solid/fluid density ratio is also used to control FSI stabilization. By default, the solid/fluid density ratio is ignored if its value is not specified. When multiple solid-fluid interfaces are present, you should choose the smallest solid/fluid density ratio.

Input File Usage: Use one of the following options to control the mesh smoothing or FSI stabilization:

∗CONTROLS, TYPE=FSI, MESH SMOOTHING=IMPLICIT
number of iterations before convergence check, maximum number of iterations, FSI penalty scale factor, solid/fluid density ratio, stiffness scale factor, linear convergence criterion

∗CONTROLS, TYPE=FSI, MESH SMOOTHING=EXPLICIT
minimum number of mesh smoothing increments, maximum number of mesh smoothing increments, FSI penalty scale factor, solid/fluid density ratio, stiffness scale factor

Abaqus/CAE Usage: Controlling FSI stabilization in an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation is not supported in Abaqus/CAE.

Controlling mesh distortion

Similar to the distortion control used in Abaqus/Explicit (see “Section controls,” Section 27.1.4, for details), Abaqus/CFD offers distortion control to prevent elements from inverting or distorting

excessively in fluid mesh movement when the explicit mesh smoothing algorithm is used. By default, distortion control is turned off during the co-simulation and ignored if the implicit mesh smoothing algorithm is used.

Input File Usage: Use the following option to deactivate distortion control (default) when the implicit mesh smoothing algorithm is used:

```
*CONTROLS, TYPE=FSI, MESH SMOOTHING=EXPLICIT,
DISTORTION CONTROL=OFF
```

Use the following option to activate distortion control:

```
*CONTROLS, TYPE=FSI, MESH SMOOTHING=EXPLICIT,
DISTORTION CONTROL=ON
```

Abaqus/CAE Usage: Controlling mesh distortion in an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation is not supported in Abaqus/CAE.

7.2.3 CONVERGENCE CRITERIA FOR NONLINEAR PROBLEMS

Products: Abaqus/Standard Abaqus/CAE

WARNING: The information in this section is provided for users who may wish to adjust the convergence criteria for the solution of nonlinear systems. In most cases these criteria need not be adjusted.

References

- "Convergence and time integration criteria: overview," Section 7.2.1
- *CONTROLS
- "Customizing general solution controls," Section 14.15.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

In nonlinear problems the governing balance equations must be solved iteratively. This section describes:

- the solution method for nonlinear problems (Newton's method);
- the field equations that can be solved by Abaqus/Standard;
- the criteria used to establish convergence of each iteration during the solution;
- "severe discontinuity" iterations; and
- the line search algorithm, which can be used to improve the robustness of the Newton method.

Solution method

Where possible, Abaqus/Standard uses Newton's method to solve nonlinear problems. In some cases it uses an exact implementation of Newton's method, in the sense that the Jacobian of the system is defined exactly, and quadratic convergence is obtained when the estimate of the solution is within the radius of convergence of the algorithm. In other cases the Jacobian is approximated so that the iterative method is not an exact Newton method. For example, some material and surface interface models (such as nonassociated flow plasticity models or Coulomb friction) create a nonsymmetric Jacobian matrix, but you may choose to approximate this matrix by its symmetric part.

Many problems exhibit discontinuous behavior. A common example is contact: at a particular point on a surface, the contact constraint is either present or absent. Another (usually less severe) example is strain reversal in plasticity at a point where the material is yielding.

Specifying the quasi-Newton method

You can choose to use the quasi-Newton technique for a particular step (described in "Quasi-Newton solution technique," Section 2.2.2 of the Abaqus Theory Guide) instead of the standard Newton method for solving nonlinear equations.

The quasi-Newton technique can save substantial computational cost in some cases by reducing the number of times the Jacobian matrix is factorized. Generally it is most successful when the system is large and many iterations are needed per increment or when the stiffness matrix is not changing much from iteration to iteration (such as in a dynamic analysis using implicit time integration or in a small-displacement analysis with local plasticity). It can be used only for symmetric systems of equations; therefore, it cannot be used when the unsymmetric solver is specified for a step (see "Defining an analysis," Section 6.1.2), nor can it be used for procedures that always produce an unsymmetric system of equations, such as "Fully coupled thermal-stress analysis," Section 6.5.3, and "Abaqus/Aqua analysis," Section 6.11.1. In addition, it cannot be used for a static Riks procedure (see "Unstable collapse and postbuckling analysis," Section 6.2.4).

The quasi-Newton method works well in combination with the line search method (see "Improving the efficiency of the solution by using the line search algorithm). Line searches help to prevent divergence of equilibrium iterations resulting from the inexact Jacobian produced by the quasi-Newton method. The line search method is activated by default for steps that use the quasi-Newton method. You can override this action by specifying line search controls.

You can specify the number of quasi-Newton iterations allowed before the kernel matrix is reformed. The default number of iterations is 8. Additional matrix reformations may occur automatically during the iteration process depending on the convergence behavior. Since quadratic convergence is not expected during quasi-Newton iterations, the logarithmic rate of convergence check is not applied during the time incrementation. Furthermore, the iteration count used in the time incrementation is a weighted sum of quasi-Newton iterations, with the weight factor depending on whether or not a kernel matrix has been reformed.

Input File Usage: ∗SOLUTION TECHNIQUE, TYPE=QUASI-NEWTON,
REFORM KERNEL=*n*

Abaqus/CAE Usage: Step module: step editor: **Other**: **Solution technique: Quasi-Newton**,
Number of iterations allowed before the kernel matrix is reformed: *n*

Specifying the separated method

Alternatively, you can choose to use the separated technique instead of the standard Newton method for solving nonlinear equations for fully coupled thermal-stress and coupled thermal-electrical procedures.

The separated technique (described in "Fully coupled thermal-stress analysis," Section 6.5.3, and "Coupled thermal-electrical analysis," Section 6.7.3) approximates the Jacobian by eliminating interfield coupling terms and can save substantial computational cost in cases where there is relatively weak coupling between the fields.

Input File Usage: ∗SOLUTION TECHNIQUE, TYPE=SEPARATED

Abaqus/CAE Usage: Step module: step editor: **Other**: **Solution technique: Separated**

Field equations

Field equations can be modeled separately or fully coupled. Some fields in Abaqus/Standard can only have linear response. Each field is discretized by using basic nodal variables (the degrees of freedom at

the nodes of the finite element model) such as the components of the displacement in a continuum stress analysis problem. Each field has a conjugate "flux."

Available fields and their conjugate fluxes

The fields and conjugate fluxes available in Abaqus/Standard are as follows:

Basic problem	Field	Conjugate flux
Stress analysis: force equilibrium	Displacement, $\mathbf{u}$	Force, $\mathbf{F}$
Stress analysis: moment equilibrium	Rotation, ϕ	Moment, $\mathbf{M}$
Stress analysis: analysis containing beams with warping	Warping, w	Bimoment, W
Heat transfer analysis	Temperature, θ	Heat flux, q
Acoustic analysis (linear only)	Acoustic pressure, u	Rate of change of fluid volumetric flux
Pore liquid flow analysis	Pore liquid pressure, u	Pore liquid volumetric flux, q
Hydrostatic fluid modeling	Fluid pressure, p	Fluid volume, V
Mass diffusion analysis	Normalized concentration, ϕ	Mass concentration volumetric flux, Q
Piezoelectric analysis	Electrical potential, φ	Electrical charge, q
Electric conduction analysis	Electrical potential, φ	Electrical current, J
Mechanism analysis (connector elements with material flow degree of freedom)	Material flow	Material flux
Analysis containing C3D4H elements (all materials, except compressible hyperelastic elastomers and elastomeric foams).	Pressure Lagrange multiplier	Volumetric flux
Analysis containing C3D4H elements with compressible hyperelastic or hyperfoam materials.	Volumetric Lagrange multiplier	Pressure flux

Constraint equations

In some cases the problem also involves constraint equations. In Abaqus/Standard the following constraints are included by using Lagrange multipliers:

Problem	Constraint variable	Constraint
Hybrid solid (except C3D4H elements)	Pressure stress	Volumetric strain compatibility
Hybrid beam	Axial force	Axial strain compatibility
Hybrid beam	Transverse shear force	Transverse shear strain compatibility
Distributing coupling	Force	Coupling displacement compatibility
Distributing coupling	Moment	Coupling rotation compatibility
Contact	Normal pressure	Surface penetration
Contact with Lagrange friction	Shear stress	Relative shear sliding

If the penalty method is used, the contact Lagrange multipliers may not be present.

Solving coupled field equations

In a general problem several (possibly nonlinear) coupled field equations of types $\alpha = 1, 2, \ldots N$ must be solved and several different (possibly nonlinear) constraints of type $j = 1, 2, \ldots K$ must be satisfied simultaneously. For example, in a structural problem in which hybrid beam elements are used, $\alpha = 1$ might represent the displacement field and the equilibrium equations for the conjugate force and $\alpha = 2$ might represent the rotation field and the equilibrium equations for the conjugate moment, while $j = 1$ represents axial strain compatibility and $j = 2$ represents transverse shear strain compatibility.

Controlling the accuracy of the solution

The default solution control parameters defined in Abaqus/Standard are designed to provide reasonably optimal solution of complex problems involving combinations of nonlinearities as well as efficient solution of simpler nonlinear cases. However, the most important consideration in the choice of the control parameters is that any solution accepted as "converged" is a close approximation to the exact solution of the nonlinear equations. In this context "close approximation" is interpreted rather strictly by engineering standards when the default value is used, as described below.

You can reset many solution control parameters related to the tolerances used for field equations. If you define less strict convergence criteria, results may be accepted as converged when they are not sufficiently close to the exact solution of the system. Use caution when resetting solution control parameters. Lack of convergence is often due to modeling issues, which should be resolved before changing the accuracy controls.

You can select the type of equation for which the solution control parameters are being defined; for example, you can redefine the default controls for the displacement field and warping degree of freedom equilibrium equations only. By default, the solution control parameters will apply to all active fields in the model. See "Defining tolerances for field equations" in "Commonly used control parameters," Section 7.2.2, for details.

Input File Usage: *CONTROLS, PARAMETERS=FIELD, FIELD=*field*
R_n^α, C_n^α, $\tilde{q}_0^\alpha$, $\tilde{q}_u^\alpha$, R_P^α, ϵ^α, C_ϵ^α, R_l^α
C_f, ϵ_l^α, ϵ_d^α

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Field Equations**: **Apply to all applicable fields** or **Specify individual fields**: *field*

Terminology

Each field, α, that is active in the problem is tested for convergence of the field equations. The following measures are used in deciding if an increment has converged:

$r_{\max}^\alpha$ The largest residual in the balance equation for field α.

$\Delta u_{\max}^\alpha$ The largest change in a nodal variable of type α in the increment.

$c_{\max}^\alpha$ The largest correction to any nodal variable of type α provided by the current Newton iteration.

e^j The largest error in a constraint of type ***j***.

$\bar{q}^\alpha(t)$ The instantaneous magnitude of the flux for field α at time ***t***, averaged over the entire model (spatial average flux). This average is by default defined by the fluxes that the elements apply to their nodes and any externally defined fluxes:

$$\bar{q}^\alpha(t) \stackrel{\text{def}}{=} \frac{1}{\sum_{e=1}^{E}\sum_{n_e=1}^{N_e} N_{n_e}^\alpha + N_{ef}^\alpha} \left(\sum_{e=1}^{E}\sum_{n_e=1}^{N_E}\sum_{i=1}^{N_{n_e}^\alpha} |q|_{i,n_e}^\alpha + \sum_{i=1}^{N_{ef}^\alpha} |q|_i^{\alpha,ef} \right).$$

Here, ***E*** is the number of elements in the model, N_e is the number of nodes in element ***e***, $N_{n_e}^\alpha$ is the number of degrees of freedom of type α at node n_e of element ***e***, $q|_{i,n_e}^\alpha$ is the magnitude of the total flux component that element ***e*** applies at its *i*th degree of freedom of type α at its n_eth node at time ***t***, N_{ef}^α is the number of external fluxes for field α (depends on element type, loading type, and number of loads applied to an element), and $q|_i^{\alpha,ef}$ is the magnitude of the *i*th external flux for field α.

$\tilde{q}^\alpha(t)$ An overall time-averaged value of the typical flux for field α so far during this step including the current increment. Normally, $\tilde{q}^\alpha(t)$ is defined as $\bar{q}^\alpha$ averaged over all the increments in the step in which $\bar{q}^\alpha$ is nonzero. The $\bar{q}^\alpha$ for the current increment is recalculated after every iteration of the current increment.

$$\tilde{q}^\alpha(t) \stackrel{\text{def}}{=} \frac{1}{N_t}\sum_{i=1}^{N_t} \bar{q}^\alpha(t|_i),$$

where N_t is the total number of increments so far in the step, including the current increment, in which $\bar{q}^\alpha(t|_i) > \epsilon^\alpha \tilde{q}^\alpha(t|_i)$. Here $\bar{q}^\alpha(t|_i)$ is the value of $\bar{q}^\alpha$ at increment *i* and ϵ^α is a small number. The default for ϵ^α is 10^{-5}, but in rare cases, you can change this default.

Alternatively, you can define a value for the average flux in the step, $\tilde{q}_u^\alpha$. In this case, $\tilde{q}^\alpha(t) = \tilde{q}_u^\alpha$ throughout the step.

At the start of the step, $\tilde{q}^\alpha$ is normally the value from the previous step (except for Step 1, when $\tilde{q}^\alpha = 10^{-2}$ by default). Alternatively, you can define an initial value for the time average flux, $\tilde{q}_0^\alpha$, as described in "Modifying the initial time average flux" in "Commonly used control parameters," Section 7.2.2. $\tilde{q}^\alpha$ retains its initial value until an iteration is completed for which $\bar{q}^\alpha > \epsilon^\alpha \tilde{q}^\alpha$, at which time we redefine $\tilde{q}^\alpha = \bar{q}^\alpha$. (If $\tilde{q}_u^\alpha$ is defined, the value defined for $\tilde{q}_0^\alpha$ is ignored.)

$\tilde{q}_{\max}^\alpha$ The time-averaged value of the largest flux corresponding to the field α during this step, excluding the current increment.

$q_{\max}^\alpha$ The largest flux corresponding to the field α during the current iteration.

Average flux

The time-averaged value of the flux ($\tilde{q}^\alpha(t)$) is computed from the spatial average of the flux ($\bar{q}^\alpha(t)$) at various instants in time. In some situations where only a small part of the model is *active* (the fluxes over the rest of the model are zero or very small), the spatial average of a flux over the entire model can be very small when compared to the spatial average over the active part of the model. Over a period of time this can result in a small value for the time-averaged value of the flux and in turn may lead to a convergence criterion that is very strict by engineering standards. To avoid such an excessively strict convergence criterion, Abaqus/Standard uses an algorithm to determine the active parts of a model at any given instant.

During an iteration any flux $q_i^\alpha(t)| < \epsilon_l^\alpha \tilde{q}_{\max}^\alpha$ is treated as inactive, and the corresponding degree of freedom is also marked inactive. $\tilde{q}_{\max}^\alpha$ is the time-averaged value of the largest flux in the model during the current step. The default value of ϵ_l^α is 10^{-5}; you can redefine this parameter.

At the end of an iteration the largest flux in the model during the current iteration ($q_{\max}^\alpha$) is compared with the time-averaged value of the largest flux ($\tilde{q}_{\max}^\alpha$). If $q_{\max}^\alpha \geq 0.1\tilde{q}_{\max}^\alpha$, the spatial average is computed over only the active parts of the model; if $q_{\max}^\alpha < 0.1\tilde{q}_{\max}^\alpha$, all inactive parts of the model are reclassified as active and the spatial average is computed over the entire model. The appropriate spatial average of the flux obtained in this manner is then used to compute the time-averaged flux $\tilde{q}^\alpha(t)$ that is used in the convergence criterion. Setting $\epsilon_l^\alpha = 0$ forces the spatial averages of a flux to be always computed over the entire model.

If you specify a value for the average flux in the step, $\tilde{q}_u^\alpha$, $\tilde{q}^\alpha(t) = \tilde{q}_u^\alpha$ throughout the step.

Residuals

Most nonlinear engineering calculations will be sufficiently accurate if the error in the residuals is less than $\frac{1}{2}$%. Therefore, Abaqus/Standard normally uses

$$r_{\max}^\alpha \leq R_n^\alpha \tilde{q}^\alpha$$

as the residual check, where you can define R_n^α (it is 0.005 by default). If this inequality is satisfied, convergence is accepted if the largest correction to the solution, $c_{\max}^\alpha$, is also small compared to the largest incremental change in the corresponding solution variable, $\Delta u_{\max}^\alpha$,

$$c^{\alpha}_{\max} \le C^{\alpha}_{n} \Delta u^{\alpha}_{\max},$$

or if the magnitude of the largest correction to the solution that would occur with one more iteration, estimated as

$$c^{\alpha}_{est} = \frac{\left(r^{\alpha}_{\max}\right)^{i}}{\min\left(\left(r^{\alpha}_{\max}\right)^{i-1}, \left(r^{\alpha}_{\max}\right)^{i-2}\right)} c^{\alpha}_{\max},$$

satisfies the same criterion:

$$c^{\alpha}_{est} \le C^{\alpha}_{n} \Delta u^{\alpha}_{\max}.$$

You can define C^{α}_{n}; the default value is 10^{-2}.

The superscripts i, $i-1$, and $i-2$ refer to the iteration number, and $(r^{\alpha}_{\max})^{0}$ refers to the largest residual in field α at the start of the first iteration of the increment. See "Commonly used control parameters," Section 7.2.2, for more details on specifying C^{α}_{n}.

Zero flux

In some cases there may be zero flux in the equations of type α anywhere in the model during some increments. Zero flux is defined as $\bar{q}^{\alpha} \le \epsilon^{\alpha} \tilde{q}^{\alpha}$, where, as discussed earlier, ϵ^{α} has a default value of 10^{-5} and the solution for field α is accepted if $r^{\alpha}_{\max} \le \epsilon^{\alpha} \tilde{q}^{\alpha}$. If not, $c^{\alpha}_{\max}$ is compared to $\Delta u^{\alpha}_{\max}$, and convergence for field α is accepted when $c^{\alpha}_{\max} \le C^{\alpha}_{\epsilon} \Delta u_{\max}$. The default value of C^{α}_{ϵ} is 10^{-3}; you can redefine this parameter.

Negligible response in some fields

Cases may arise where more than one field is active in the model yet there is negligible response in some of the fields in some increments. If some type of physical conversion factor, f^{α}_{β}, exists between active fields α and β, $\tilde{q}^{\alpha}$ in the above paragraph can be replaced by $f^{\alpha}_{\beta} C_f \tilde{q}^{\beta}$ for those particular increments where $\tilde{q}^{\alpha}$ is deemed too small ($\bar{q}^{\alpha} \le \tilde{q}^{\alpha} < f^{\alpha}_{\beta} C_f \tilde{q}^{\beta}$) to be used realistically as part of the convergence criteria for field α. An example of f^{α}_{β} is a characteristic length to convert between force and moment.

Here, f^{α}_{β} is a factor calculated by Abaqus/Standard based on the problem definition and the fields involved and C_f is a field conversion ratio that you can define. The default value for C_f is 1.0. Currently, this concept is used only for converting between the fields associated with forces and moments, when f^{α}_{β} represents a characteristic element length.

Linear increments

Linear cases do not require more than one equilibrium iteration per increment. If

$$r^{\alpha}_{\max} \le R^{\alpha}_{l} \tilde{q}^{\alpha}$$

for all α, the increment is considered to be linear.

You can define R_l^α; it is intended to be very small. The default value of R_l^α is 10^{-8}. Any case that passes such a stringent comparison of the largest residual with the average flux magnitude in each field is considered linear and does not require further iteration. If this requirement is satisfied at some iteration after the first, the solution is accepted without any check on the size of the correction to the solution.

Nonquadratic convergence

In some cases quadratic convergence of the iterations is not possible because the Jacobian of the Newton scheme is approximated. If after I_P iterations the convergence rate is only linear, Abaqus/Standard uses a looser tolerance,

$$r_{\max}^\alpha \le R_p^\alpha \tilde{q}^\alpha,$$

as the residual check. This tolerance modification is not applied when the quasi-Newton method is used, since it is normal for this method to require a larger number of iterations to converge.

You can define R_p^α, which is 2×10^{-2} by default. You can also define I_P (by default, $I_P = 9$; see "Controlling iteration").

Convergence also requires that

$$c_{\max}^\alpha \le C_n^\alpha \Delta u_{\max}^\alpha.$$

Iteration continues until both criteria are satisfied for all active fields or the increment is abandoned.

When the active field is the displacement, the convergence criterion requiring the largest displacement correction to be small relative to the maximum displacement increment ($c_{\max}^\alpha \le C_n^\alpha \Delta u_{\max}^\alpha$) is ignored when the maximum displacement increment itself is very small, as defined by $\Delta u_{max}^\alpha < \epsilon_d^\alpha f_\beta^\alpha$, where f_β^α is the characteristic element length. The default value for ϵ_d^α is 10^{-8}; you can redefine this parameter.

Controlling iteration

Each increment of a nonlinear solution will usually be solved by multiple equilibrium iterations. The number of iterations may become excessive, in which case the increment size should be reduced and the increment attempted again. On the other hand, if successive increments are solved with a minimum number of iterations, the increment size may be increased. You can specify a number of time incrementation control parameters; some of them are described in this section, while the remainder are described in "Time integration accuracy in transient problems," Section 7.2.4.

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION
$I_0, I_R, I_P, I_C, I_L, I_G, I_S, I_A, I_J, I_T, I_S^c, I_J^c, I_A^c$
$D_f, D_C, D_B, D_A, D_S, D_H, D_D, W_G$
$D_G, D_M, D_M^{dyn}, D_M^{diff}, D_L, D_E, D_R, D_F$
D_T

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Reattempting an increment because of trouble with element or material calculations

Abaqus/Standard may have trouble with the element calculations because of excessive distortion in large-displacement problems or because of very large plastic strain increments. If this occurs and automatic time incrementation has been chosen, the increment will be attempted again with a time increment of D_H times the current time increment, where you can define D_H. By default, $D_H = 0.25$. If fixed time stepping has been chosen, the analysis will terminate with an error message.

Reattempting a diverging increment

Sometimes the increment is too large for the solution to converge at all—the initial state is outside the "radius of convergence" of the Newton method. This condition can be detected by observing the behavior of the largest residuals, $r^{\alpha}_{\max}$. In some cases these will not decrease from iteration to iteration throughout an iteration sequence that leads to convergence, but we assume that, if they fail to decrease over two consecutive iterations, the iterations should be abandoned. Thus, if

$$\min\left(\left(r^{\alpha}_{\max}\right)^{i}, \left(r^{\alpha}_{\max}\right)^{i-1}\right) > \left(r^{\alpha}_{\max}\right)^{i-2},$$

where *i* is the iteration counter, the iterations are abandoned. This check is first made after I_0 iterations following a solution discontinuity. You can define I_0; it must be at least 3. The default value of I_0 is 4. If fixed time stepping has been chosen, the analysis will terminate with an error message.

With automatic time stepping the increment is begun again, using a time increment of D_f times the previous attempt, where you can define D_f. By default, $D_f = 0.25$. This subdivision continues until a successful time increment is found or the minimum time increment allowed has failed, in which case the job ends with an error message. Using the line search algorithm with $N^{ls} = 4$ sometimes helps in such cases (see "Improving the efficiency of the solution by using the line search algorithm").

Reattempting an increment when too many equilibrium iterations are required

In case quadratic convergence cannot be obtained, the logarithmic rate of convergence,

$$\ln\left(\left(r^{\alpha}_{\max}\right)^{i} / \left(r^{\alpha}_{\max}\right)^{i-1}\right),$$

will often be maintained throughout the iteration process. This rate can be established during the early iterations. If convergence has not been achieved after I_R or more iterations following a solution discontinuity, if automatic time incrementation has been selected, and if the slowest convergence rate over all fields α suggests that more than I_C total iterations subsequent to the last solution discontinuity are expected to be required, the increment is begun again with a time increment of D_C times the one abandoned. If fixed time incrementation has been chosen, the iterations are continued; but if convergence is not achieved within I_C iterations after the last solution discontinuity in the increment, the analysis will terminate with an error message.

You can define the values of I_R, I_C, and D_C. By default, $I_R = 8$, $I_C = 16$, and D_C=0.5.

Increasing or reducing the size of the time increment for efficiency

When automatic time incrementation is chosen, the effectiveness of the nonlinear equation solution is used in the selection of the next time increment (in addition to the time integration accuracy criteria discussed in "Time integration accuracy in transient problems," Section 7.2.4). If no more than I_G iterations are required in two consecutive increments, the time increment may be increased by a factor of D_D. If an increment converges but takes more than I_L iterations, the next time increment is reduced to D_B times the current time increment. You can define the values of I_G, I_L, D_D, and D_B. By default, $I_G = 4$, $I_L = 10$, $D_D = 1.5$, and $D_B = 0.75$.

Extrapolation

At each increment after the first increment of a nonlinear analysis step Abaqus/Standard estimates the solution to the increment by extrapolating the solution from the previous increment (or increments). By default, 100% linear extrapolation is used (1% for the Riks method). Extrapolation is abandoned if

$$\Delta t_i \leq D_E \Delta t_{i-1},$$

where Δt_i is the proposed new time increment, and Δt_{i-1} is the last successful time increment. You can define the value of D_E; it is 0.1 by default.

You can turn this extrapolation scheme off for a particular step—see "Defining an analysis," Section 6.1.2.

Convergence of strain constraints in hybrid elements

Strain constraint convergence in "hybrid" elements is checked by comparing the largest error in each strain constraint, e^j, with an absolute tolerance for the corresponding error, T^j. The magnitudes of these errors are reported in the message (`.msg`) file after each iteration as "compatibility errors." For example, the volumetric compatibility error is a measure of the accuracy with which the incompressibility constraint is satisfied. Since nonlinearity in constraint equations is generally reflected in the field equations in the same problem, no attempt is made to estimate convergence rates in these constraint equations: we assume that the measures of convergence rate in the field equations are sufficient.

You can define the T^j (T^{vol}, T^{axial}, and T^{tshear}). By default, all of the $T^j = 10^{-5}$.

Input File Usage: *CONTROLS, PARAMETERS=CONSTRAINTS
T^{vol}, T^{axial}, T^{tshear}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Constraint Equations**

Severe discontinuity iterations

Abaqus/Standard distinguishes between regular, equilibrium iterations (in which the solution varies smoothly) and severe discontinuity iterations (SDIs) in which abrupt changes in stiffness occur. By default, Abaqus/Standard will continue to iterate until the severe discontinuities are sufficiently small (or no severe discontinuities occur) and the equilibrium (flux) tolerances are satisfied. For more information

on the criteria used for the severe discontinuity checks, see "Severe discontinuities in Abaqus/Standard" in "Defining an analysis," Section 6.1.2. Alternatively, Abaqus/Standard will continue to iterate until no severe discontinuities occur and the equilibrium (flux) tolerances are satisfied. This more traditional method can cause convergence difficulties if the contact conditions are only weakly determined and contact "chattering" occurs or if a large number of severe discontinuity iterations are required to settle the contact conditions.

You can define the contact and slip compatibility tolerance, the soft contact compatibility tolerance for low pressure, and the contact force error tolerance.

Input File Usage: *CONTROLS, PARAMETERS=CONSTRAINTS
, , , T^{cont}, T^{soft}, , , T^{cfe}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Constraint Equations**

Defining the contact force error tolerance is not supported in Abaqus/CAE.

Severe discontinuity iterations in implicit dynamic analysis

In implicit dynamic analysis, the average time of all contact changes in the increment is estimated and the time incrementation is interrupted to solve impact equations at that time. With augmented Lagrange or penalty constraint enforcement methods or with softened contact, no contact constraints are imposed when impact equations are solved. However, if the contact constraints are not satisfied within given tolerances, a severe discontinuity iteration is forced. See "Intermittent contact/impact," Section 2.4.2 of the Abaqus Theory Guide, for details on intermittent contact in dynamic problems.

Controlling the number of severe discontinuity iterations

By default, Abaqus applies sophisticated criteria involving changes in penetration, changes in the residual force, and the number of severe discontinuities from one iteration to the next to determine whether iteration should be continued or terminated. Hence, it is in principle not necessary to limit the number of severe discontinuity iterations. This makes it possible to run contact problems that require large numbers of contact changes without having to change the control parameters. It is still possible to set a limit, I_S^c, for the maximum number of severe discontinuity iterations; by default, $I_S^c = 50$, which in practice should always be more than the actual number of iterations in an increment.

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION
, , , , , , , , , , I_S^c

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Controlling the number of severe discontinuity iterations when severe discontinuities always force iterations

In this case a limit, I_S, is placed on the number of iterations caused by severe discontinuities in an increment. If more than I_S iterations are required for severe discontinuities, the increment is started over with a time increment size of D_S times the abandoned increment size (for automatic

time incrementation). If fixed time incrementation was chosen, the analysis terminates with an error message. You can define the values of I_S and D_S. By default, $I_S = 12$ and $D_S = 0.25$.

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION
, , , , , , I_S
, , , , D_S

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Improving the efficiency of the solution by using the line search algorithm

Abaqus/Standard provides the option of including a "line search" algorithm. The purpose of the line search is to improve the robustness of the Newton or quasi-Newton methods. By default, the line search is active only for steps that use the quasi-Newton method. During equilibrium iterations where residuals are large, the line search algorithm scales the correction to the solution by a line search scale factor, s^{ls}. An iterative process is used to find the value of s^{ls} that minimizes the component of the residual vector in the direction of the correction vector; this component is called y^j, where j is the line search iteration number. Each line search iteration requires one pass through the Abaqus/Standard element loop but does not require any operations using the global stiffness matrix.

It is usually sufficient to determine s^{ls} only to modest accuracy. There are several controls used to limit this accuracy. A maximum of $j = N^{ls}$ line search iterations are performed. There is a limit on the allowable range of s^{ls}:

$$s^{ls}_{min} \le s^{ls} \le s^{ls}_{max}.$$

The line search ceases when

$$|y^j| \le f^{ls}_s |y^0|,$$

where y^0 is evaluated before the first equilibrium iteration. The residual reduction factor at which the line search ceases, f^{ls}_s, is typically set to a rather loose tolerance. The line search algorithm will also cease when the change in s^{ls} provided by a line search iteration is less than η^{ls} times s^{ls}.

You can define the values of N^{ls}, s^{ls}_{max}, s^{ls}_{min}, f^{ls}_s, and η^{ls}. By default, $N^{ls} = 0$ with the Newton method, and N^{ls}=5 with the quasi-Newton method. Set N^{ls} to a nonzero value to activate the line search algorithm or to zero to forcibly deactivate line search. Default values for the additional line search parameters are s^{ls}_{max} = 1.0, s^{ls}_{min} = 0.0001, f^{ls}_s = 0.25, and η^{ls} = 0.10. These defaults are chosen to achieve modest accuracy for the line search scale factor, while minimizing the additional cost of line search iterations. More agressive line searching can be beneficial in some simulations, especially when many nonlinear iterations and/or cutbacks are needed to resolve sharp discontinuities in the solution. In these cases you could try allowing more line search iterations (N^{ls}=10) and requiring more accuracy in the line search scale factor (η^{ls}=0.01). This may result in more line search iterations but fewer nonlinear iterations and cutbacks and an overall reduction in solution cost.

Input File Usage: *CONTROLS, PARAMETERS=LINE SEARCH
N^{ls}, s^{ls}_{max}, s^{ls}_{min}, f^{ls}_s, η^{ls}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Line Search Control**

7.2.4 TIME INTEGRATION ACCURACY IN TRANSIENT PROBLEMS

Products: Abaqus/Standard Abaqus/CAE

References

- "Convergence and time integration criteria: overview," Section 7.2.1
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Uncoupled heat transfer analysis," Section 6.5.2
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1
- "Rate-dependent plasticity: creep and swelling," Section 23.2.4
- *CONTROLS
- "Customizing general solution controls," Section 14.15.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Abaqus/Standard usually uses automatic time stepping schemes for the solution of transient problems. Factors influencing the increment size for transient problems include convergence aspects related to the degree of geometric, material, and contact nonlinearity (which also influence non-transient problems and are discussed in "Convergence criteria for nonlinear problems," Section 7.2.3) and the ability of the time integration operator to accurately resolve variations in the accelerations, velocities, and displacements over an increment. This section discusses tolerance parameters and adjustments to the time increment size related to the latter aspect.

Time incrementation parameters and adjustment criteria

Table 7.2.4–1 lists tolerance parameters available for specific analysis procedures. Descriptions of time integrators for the transient procedure types and, in the case of implicit dynamics, discussion of additional factors influencing the time increment size related to accuracy of time integration are provided in the respective sections referenced in Table 7.2.4–1.

Table 7.2.4–1 Time integration accuracy measures for various procedures.

Procedure	**Accuracy measure** (S^J)	**Tolerance** (T^J)
Implicit dynamics ("Implicit dynamic analysis using direct integration," Section 6.3.2)	Half-increment residual	Half-increment residual tolerance

Transient heat transfer analysis (“Uncoupled heat transfer analysis,” Section 6.5.2)	Temperature increment, $\Delta\theta$	$\Delta\theta_{max}$
Consolidation analysis (“Coupled pore fluid diffusion and stress analysis,” Section 6.8.1)	Pore pressure increment, Δu_w	Δu_w^{max}
Creep and viscoelastic material behavior (“Rate-dependent plasticity: creep and swelling,” Section 23.2.4)	$(\dot{\bar{\varepsilon}}^{cr}\|_{t+\Delta t} - \dot{\bar{\varepsilon}}^{cr}\|_t)\Delta t$	Creep tolerance

In any transient analysis where automatic time incrementation is used, some of these tolerances, T^J, $J = 1, 2, \ldots$, will be active. Corresponding measures of the integration accuracy, S^J, will be calculated for each increment in the step. Abaqus/Standard will use these values to adjust the time incrementation using the criteria described in this section. The smallest time increment required by all criteria is used if more than one accuracy measure is active.

Reducing the time increment size

If $S^J > T^J$ for any control, ***J***, that is active in the step, the time increment Δt is too large to satisfy that time integration accuracy requirement. The increment is, therefore, begun again with a time increment of

$$D_A \left(\frac{T^J}{S^J} \right) \Delta t,$$

where you can define the value of D_A. By default, D_A = 0.85.

Input File Usage: ∗CONTROLS, PARAMETERS=TIME INCREMENTATION
first data line
, , , D_A

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Increasing the time increment size

If at the current time increment, Δt,

$$\Delta t \left(\frac{S^J}{\Delta t} \right)_i < W_G T^J$$

for all ***J*** in each of I_T consecutive increments, i, and if no cut-back has occurred within those increments because of nonlinearity, the next time increment will be increased to

$$\min(D_G \Delta t_p, D_M \Delta t).$$

You can define the values of I_T, W_G, and D_G. By default, I_T = 3, W_G = 0.75, and D_G = 0.8. Δt_p is the proposed new time increment, which is defined as

$$\Delta t_p = \left(\frac{T^J}{S^J/\Delta t} \right)$$

for transient heat transfer and transient mass diffusion problems and which is defined as

$$\Delta t_p = I_T \left(\frac{T^J}{\sum_{i=1}^{I_T} \left(S^J/\Delta t\right)_i} \right)$$

for other transient problems.

A limit, D_M, is placed on the time increment increase factor. The default value of D_M depends on the type of analysis:

- D_M^{dyn} = 1.25 for dynamic analysis
- D_M^{diff} = 2.0 for diffusion-dominated processes: creep, transient heat transfer, coupled temperature-displacement, soils consolidation, and transient mass diffusion
- D_M = 1.5 for all other cases

You can redefine D_M for each analysis type.

If the problem is nonlinear, the time increment may be restricted by the rate of convergence of the nonlinear equations. The time incrementation controls used with nonlinear problems are described in "Convergence criteria for nonlinear problems," Section 7.2.3.

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION
, , , , , , , , , I_T
, , , , , , , W_G
D_G, D_M, D_M^{dyn}, D_M^{diff}

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Avoiding small changes to the time increment size during implicit integration procedures

In linear transient problems when Abaqus/Standard uses implicit integration, the system of equations must be reformed and decomposed whenever the time increment changes even though the stiffness matrix does not change. Therefore, to reduce the number of increments at which the system matrix changes, Abaqus/Standard makes use of the factor D_L, where

$$D_L = min \left(\frac{\Delta t_p}{D_M \Delta t} \right).$$

The definition of D_L results in the following inequality between the proposed and the current time increments:

$$\Delta t_p \geq D_L D_M \Delta t.$$

Based on this inequality the time increment is allowed to increase only when its value computed by the criteria described earlier in this section, or computed using the value of **PNEWDT** specified in certain user subroutines (**UMAT**, for example), is greater than or equal to $D_L D_M \Delta t$. The default value of D_L is 1.0, but you can redefine it to be a smaller number. Reducing D_L to a value less than 1.0 allows the time increment to increase by a factor that is smaller than D_M, thereby forcing a time increment change, even if the change is small. Otherwise, the solution continues with the same Δt.

Input File Usage: *CONTROLS, PARAMETERS=TIME INCREMENTATION
first data line
second data line
, , , , D_L

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: toggle on **Specify**: **Time Incrementation**; click **More** to see additional data tables

Part IV: Analysis Techniques

- Chapter 8, “Analysis Techniques: Introduction”
- Chapter 9, “Analysis Continuation Techniques”
- Chapter 10, “Modeling Abstractions”
- Chapter 11, “Special-Purpose Techniques”
- Chapter 12, “Adaptivity Techniques”
- Chapter 13, “Optimization Techniques”
- Chapter 14, “Eulerian Analysis”
- Chapter 15, “Particle Methods”
- Chapter 16, “Sequentially Coupled Multiphysics Analyses”
- Chapter 17, “Co-simulation”
- Chapter 18, “Extending Abaqus Analysis Functionality”
- Chapter 19, “Design Sensitivity Analysis”
- Chapter 20, “Parametric Studies”

8. Analysis Techniques: Introduction

Introduction 8.1

8.1 Introduction

- “Analysis techniques: overview,” Section 8.1.1

8.1.1 ANALYSIS TECHNIQUES: OVERVIEW

Abaqus provides an extensive selection of analysis techniques. These techniques provide powerful tools for performing your analysis more efficiently and effectively.

Analysis continuation techniques

In many cases your analysis results represent a significant investment of computational effort. As a result, you will often want to reduce computation costs by utilizing results from an analysis that has already been performed. In other cases your overall analysis history will be comprised of distinct Abaqus jobs, each representing a portion of the response history of the model. Abaqus provides the following analysis continuation techniques:

- Abaqus allows you to restart an analysis, as long as you request that certain files containing model and state data be saved in the original analysis. See "Restarting an analysis," Section 9.1.1.
- You can perform part of an analysis with Abaqus/Standard or Abaqus/Explicit and continue the analysis with the other product. You can transfer results from Abaqus/Standard to Abaqus/Explicit, from Abaqus/Explicit to Abaqus/Standard, and from Abaqus/Standard to Abaqus/Standard. See "Transferring results between Abaqus analyses: overview," Section 9.2.1.

Modeling abstractions

All Abaqus models involve certain abstractions. In addition to the traditional abstractions associated with the finite element method, you can include techniques in your model to obtain more cost-effective solutions. Abaqus provides the following techniques for modeling abstractions:

- You can create substructures by grouping a number of elements together and retaining only the degrees of freedom needed to interface with adjacent structures. This technique is particularly useful when a substructure is to be reused in the same analysis, in different analyses, or by different analysts. See "Using substructures," Section 10.1.1.
- You can analyze local regions of a model in greater detail and interpolate the solution results from a larger coarser global model. See "Submodeling: overview," Section 10.2.1.
- You can allow for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the stiffness, mass, viscous damping, structural damping, and load vectors in a model. See "Generating matrices," Section 10.3.1.
- You can create a three-dimensional model in Abaqus/Standard by revolving various forms of axisymmetric and three-dimensional model sectors about an axis of symmetry (see "Symmetric model generation," Section 10.4.1). You can also transfer the solution obtained in an original axisymmetric model to the new model (see "Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh," Section 10.4.2). In addition, for models that exhibit cyclic symmetry you can extract eigenmodes and perform mode-based steady-state

dynamic analysis by modeling only a single repetitive sector of the model (see "Analysis of models that exhibit cyclic symmetry," Section 10.4.3).

- Using the periodic media analysis technique, you can effectively model systems that are repetitive in nature, such as manufacturing processes involving conveyor belts or continuous forming operations. See "Periodic media analysis," Section 10.5.1.
- You can define a complex beam cross-section, including multiple materials and complex geometry, and automatically generate beam element cross-section properties. See "Meshed beam cross-sections," Section 10.6.1.
- Using the extended finite element method, you can model discontinuities, such as cracks, as an enriched feature without creating a mesh to match the geometry of the discontinuity. See "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1.

Special-purpose techniques

Certain analysis techniques do not fall into a general classification and are grouped here as special-purpose techniques. Abaqus provides the following special-purpose techniques:

- You can use the inertia relief technique as an inexpensive alternative to performing a full dynamic analysis on a free or partially constrained body subjected to loads derived from rigid body accelerations. See "Inertia relief," Section 11.1.1.
- You can selectively remove, and later reintroduce, parts of a model. See "Element and contact pair removal and reactivation," Section 11.2.1.
- You can introduce small imperfections into a model, typically for postbuckling analysis. See "Introducing a geometric imperfection into a model," Section 11.3.1.
- You can evaluate fracture performance through contour integral evaluation, through crack propagation modeling techniques, or by using line spring elements in conjunction with shell elements. See "Fracture mechanics: overview," Section 11.4.1.
- You can model coupling between the deformation of a fluid-filled structure and the pressure exerted by a contained fluid (see "Surface-based fluid cavities: overview," Section 11.5.1).
- In Abaqus/Explicit you can use the mass scaling technique to control the stable time increment and increase computational efficiency. See "Mass scaling," Section 11.6.1.
- You can use selective subcycling to allow different time increments to be used for different groups of elements, which can reduce the run time for an analysis when a small region of elements in the model controls the stable time increment. See "Selective subcycling," Section 11.7.1.
- You can use steady-state detection to detect the time in a quasi-static uni-directional Abaqus/Explicit simulation when a steady-state condition has been reached and then terminate the simulation. See "Steady-state detection," Section 11.8.1.

Adaptivity techniques

Adaptivity techniques enable modification of your mesh to obtain a better solution. Abaqus provides the following adaptivity techniques:

- You can use ALE adaptive meshing to control mesh distortion or to model material loss. See "ALE adaptive meshing: overview," Section 12.2.1.
- You can use adaptive remeshing with Abaqus/Standard and Abaqus/CAE to iteratively improve your mesh to obtain a more accurate solution. See "Adaptive remeshing: overview," Section 12.3.1.
- You can use mesh-to-mesh solution mapping as part of a mesh replacement strategy for distortion control. See "Mesh-to-mesh solution mapping," Section 12.4.1.

See "Adaptivity techniques," Section 12.1.1, for a comparison of the adaptivity methods.

Optimization techniques

You can use structural optimization, an iterative process that helps you refine your designs, to perform topology and shape optimization. In Abaqus/CAE you create the model to be optimized and define, configure, and execute the structural optimization. See "Structural optimization: overview," Section 13.1.1.

Eulerian analysis

You can use Abaqus/Explicit to simulate extreme deformation, up to and including fluid flow, in an Eulerian analysis. Eulerian materials can be coupled to Lagrangian structures to analyze fluid-structure interactions. See "Eulerian analysis," Section 14.1.1.

Particle methods

Using the smoothed particle hydrodynamics technique, you can model violent free-surface fluid flow (such as wave impact) and extremely high deformation/obliteration of solid structures (such as ballistics). See "Smoothed particle hydrodynamics," Section 15.2.1.

Using the discrete element technique, you can model particulate media and perform analyses such as granular material mixing or segregation, transport, and deposition of particulate materials. See "Discrete element method," Section 15.1.1.

Sequentially coupled multiphysics analyses

In Abaqus/Standard you can perform sequentially coupled multiphysics analyses when the coupling between one or more of the physical fields in a model is only important in one direction. See "Sequentially coupled multiphysics analyses," Section 16.1.

Co-simulation

You can use the co-simulation technique for run-time coupling of two Abaqus analyses or of Abaqus with third-party analysis programs to perform multiphysics simulation. See "Co-simulation: overview," Section 17.1.1.

Extending Abaqus analysis functionality

You can use the flexibility of user subroutines to increase the functionality of Abaqus. See "User subroutines and utilities," Section 18.1.

Design sensitivity analysis

You can use design sensitivity analysis (DSA) techniques to determine sensitivities of responses with respect to specified design parameters. You can use these techniques for design studies within Abaqus/Standard or in conjunction with third-party design optimization tools. See "Design sensitivity analysis," Section 19.1.1.

Parametric studies

You can use parametric studies to perform multiple analyses in which you can systematically vary modeling parameters that you define. See "Scripting parametric studies," Section 20.1.1, and "Parametric studies: commands," Section 20.2.

Availability of analysis techniques

The availability of the analysis techniques provided in Abaqus is summarized in Table 8.1.1–1. In addition, adaptive remeshing and optimization techniques are available in Abaqus/CAE (see "Adaptive remeshing," Section 12.3 and "Structural optimization: overview," Section 13.1).

Table 8.1.1–1 Availability of analysis techniques in Abaqus.

Technique	Abaqus/Standard	Abaqus/Explicit	Abaqus/CFD
"Restarting an analysis," Section 9.1	√	√	√
"Importing and transferring results," Section 9.2	√	√	
"Substructuring," Section 10.1	√		
"Submodeling," Section 10.2	√	√	
"Generating matrices," Section 10.3	√		
"Symmetric model generation, results transfer, and analysis of cyclic symmetry models," Section 10.4	√		
"Periodic media analysis," Section 10.5		√	
"Meshed beam cross-sections," Section 10.6	√	√	

Technique	Abaqus/Standard	Abaqus/Explicit	Abaqus/CFD
"Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7	√		
"Inertia relief," Section 11.1	√		
"Mesh modification or replacement," Section 11.2	√		
"Geometric imperfections," Section 11.3	√	√	
"Fracture mechanics," Section 11.4	√		
"Surface-based fluid modeling," Section 11.5	√	√	
"Mass scaling," Section 11.6		√	
"Selective subcycling," Section 11.7		√	
"Steady-state detection," Section 11.8		√	
"ALE adaptive meshing," Section 12.2	√	√	
"Analysis continuation after mesh replacement," Section 12.4	√		
"Eulerian analysis," Section 14.1		√	
"Continuum particle analyses," Section 15.2		√	
"Discrete element method," Section 15.1		√	
"Sequentially coupled multiphysics analyses," Section 16.1	√		
"Co-simulation," Section 17.1	√	√	√
"User subroutines and utilities," Section 18.1	√	√	√
"Design sensitivity analysis," Section 19.1	√		
"Scripting parametric studies," Section 20.1	√	√	

9. Analysis Continuation Techniques

Restarting an analysis 9.1
Importing and transferring results 9.2

9.1 Restarting an analysis

- "Restarting an analysis," Section 9.1.1

9.1.1 RESTARTING AN ANALYSIS

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CFD Abaqus/CAE

References

- "Output," Section 4.1.1
- *RESTART
- "Restarting an analysis," Section 19.6 of the Abaqus/CAE User's Guide

Overview

When you run an analysis, you can write the model definition and state to the files required for restart. Scenarios for using the restart capability include:

- **Continuing an interrupted run:** If an analysis is interrupted by a computer malfunction, the Abaqus restart analysis capability allows the analysis to complete as originally defined.
- **Continuing with additional steps:** After viewing results from a successful analysis, you may decide to append steps to the load history.
- **Changing an analysis:** Sometimes, having viewed the results of the previous analysis, you may want to restart the analysis from an intermediate point and change the remaining load history data in some manner. In addition, you may want to add additional steps to the load history if the previous analysis completed successfully.

"Output," Section 4.1.1, describes the process of obtaining results output from an Abaqus/Standard restart file.

Writing restart files

If you want to be able to restart an analysis, you must request restart output. This output will be written to files that can be used to restart the analysis. If you do not request that restart data be written, restart files will not be created in Abaqus/Standard, while in Abaqus/Explicit and Abaqus/CFD a state file will be created with results at only the beginning and end of each step.

In Abaqus/Standard these files are the restart (*job-name*`.res`; file size limited to 16 gigabytes), analysis database (`.mdl` and `.stt`), part (`.prt`), output database (`.odb`), and linear dynamics and substructure database (`.sim`) files. In Abaqus/Explicit these files are the restart (*job-name*`.res`; file size limited to 16 gigabytes), analysis database (`.abq`, `.mdl`, `.pac`, and `.stt`), part (`.prt`), selected results (`.sel`), and output database (`.odb`) files. In Abaqus/CFD these files are the restart and analysis database (*job-name*`.sim`) and output database (`.odb`) files. These files, collectively referred to as the restart files, allow an analysis to be completed up to a certain point in a particular run and restarted and continued in a subsequent run. The output database file only needs to contain the model data; results data are not required and can be suppressed.

You can control the amount of data written to the restart files, as described below. The amount of data written to the restart file can be changed from step to step if you include the restart request in each step definition.

Restart information is not written during the following linear perturbation steps:

- "Static stress analysis," Section 6.2.2 (perturbation)
- "Eigenvalue buckling prediction," Section 6.2.3
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Complex eigenvalue extraction," Section 6.3.6
- "Transient modal dynamic analysis," Section 6.3.7
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Subspace-based steady-state dynamic analysis," Section 6.3.9
- "Response spectrum analysis," Section 6.3.10
- "Random response analysis," Section 6.3.11
- "Eddy current analysis," Section 6.7.5

Input File Usage: Use the following option to request that restart data be written for an analysis:

*RESTART, WRITE

The *RESTART, WRITE option can be used as either model data or history data.

Abaqus/CAE Usage: Step module: **Output→Restart Requests**

In Abaqus/CAE restart requests are always associated with a particular step; you cannot define a restart request for the entire analysis. Restart requests are created by default for every step; restart requests for Abaqus/Standard and Abaqus/CFD steps have a default frequency of 0, while restart requests for Abaqus/Explicit steps have a default number of intervals of 1.

Controlling the frequency of output to the restart files

You can specify the frequency at which data will be written to the Abaqus/Standard restart file and the Abaqus/Explicit and Abaqus/CFD state files. The variables to be written cannot be specified; a complete set of data is written each time. Therefore, the restart files can be quite large unless you control the frequency with which restart information is written. If restart information is requested for an Abaqus/Standard analysis at exact time intervals, Abaqus/Standard will obtain a solution each time data are written. In this case if the frequency of output to the restart file is high, the number of increments and, consequently, the computational cost of the analysis may increase considerably.

Specifying the frequency of output to the Abaqus/Standard restart file in increments

By default, Abaqus/Standard will write data to the restart file after each increment at which the increment number is exactly divisible by a user-specified frequency value, N, and at the end of each step of the analysis (regardless of the increment number at that time). In a direct cyclic or a low-cycle fatigue analysis Abaqus/Standard will write data to the restart file only at the end of a loading cycle; therefore,

Abaqus/Standard will write data to the restart file after each iteration (or cycle in a low-cycle fatigue analysis) at which the iteration number (or cycle number in a low-cycle fatigue analysis) is exactly divisible by N and at the end of each step of the analysis.

Input File Usage: *RESTART, WRITE, FREQUENCY=N

By default, N=1.

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter N in the **Frequency** column for each step

By default, N=0 (no restart information is written).

Specifying the frequency of output to the Abaqus/Standard restart file in time intervals

Abaqus/Standard can divide the step into a user-specified number of time intervals, n, and write the results at the end of each interval, for a total of n points for the step. If n is specified, by default data will be written to the results file at the exact times calculated by dividing the step into n equal intervals. Alternatively, you can choose to write the information at the increment ending immediately after the time dictated by each interval.

You can specify the frequency of restart output in time intervals only for the procedures listed in Table 9.1.1–1. In addition, this capability is not supported for linear perturbation analyses.

Input File Usage: Use the following option to request results at the exact time intervals:

*RESTART, WRITE, NUMBER INTERVAL=n, TIME MARKS=YES

Use the following option to request results at the increments ending immediately after each time interval:

*RESTART, WRITE, NUMBER INTERVAL=n, TIME MARKS=NO

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter n in the **Intervals** column; toggle on the **Time Marks** column for each step if you want the results written at the exact time intervals

Table 9.1.1–1 List of Abaqus/Standard procedures that support restart at time intervals.

Procedure	Time incrementation	Restart at exact time intervals	Restart at approximate time intervals
"Static stress analysis," Section 6.2.2 (except if the Riks method is used)	Automatic	√	√
	Fixed	—	√
"Implicit dynamic analysis using direct integration," Section 6.3.2	Automatic	√	√
	Fixed	—	√

Procedure	Time incrementation	Restart at exact time intervals	Restart at approximate time intervals
"Uncoupled heat transfer analysis," Section 6.5.2 (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
"Mass diffusion analysis," Section 6.9.1 (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
"Coupled pore fluid diffusion and stress analysis," Section 6.8.1 (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
"Fully coupled thermal-stress analysis," Section 6.5.3	Automatic	√	√
	Fixed	—	√
"Fully coupled thermal-electrical-structural analysis," Section 6.7.4	Automatic	√	√
	Fixed	—	√
"Coupled thermal-electrical analysis," Section 6.7.3 (except if you specify that the analysis end when steady state is reached)	Automatic	√	√
	Fixed	—	√
"Steady-state transport analysis," Section 6.4.1	Automatic	√	√
	Fixed	—	√
"Subspace-based steady-state dynamic analysis," Section 6.3.9	Fixed	—	√
"Quasi-static analysis," Section 6.2.5	Automatic	√	√
	Fixed	—	√

Time incrementation

If the output frequency is specified in terms of the number of intervals, Abaqus/Standard will adjust the time increments to ensure that data are written at the exact time points specified. In some cases Abaqus may use a time increment smaller than the minimum time increment allowed in the step in the increment directly before a time point. However, Abaqus will not violate the minimum time increment allowed for consolidation, transient mass diffusion, transient heat transfer, transient couple thermal-electrical,

transient coupled temperature-displacement, and transient coupled thermal-electrical-structural analyses. For these procedures if a time increment smaller than the minimum time increment is required, Abaqus will use the minimum time increment allowed in the step and will write restart data at the first increment after the time point.

When the output frequency is specified in terms of the number of intervals, the number of increments necessary to complete the analysis might increase, which might adversely affect performance.

Specifying the frequency of output to the Abaqus/Explicit state file

Abaqus/Explicit will divide the step into a user-specified number of time intervals, n, and write the results at the beginning of the step and at the end of each interval, for a total of n+1 points for the step, with the last point matching the end of the step. By default, the results will be written to the state file at the increment ending immediately after the time dictated by each interval. Alternatively, you can choose to write the results at the exact times calculated by dividing the step into n equal intervals.

If a problem precludes the analysis from continuing to completion, such as if an element becomes excessively distorted, Abaqus/Explicit will attempt to save the last completed increment in the state file.

Alternatively, you can specify the number of intervals as zero to turn off all restart frame output. This setting can be used to reduce the disk usage if there is no continuation after the current analysis.

Input File Usage: Use the following option to request results at the increments ending immediately after each time interval:

*RESTART, WRITE, NUMBER INTERVAL=n, TIME MARKS=NO

Use the following option to request results at the exact time intervals:

*RESTART, WRITE, NUMBER INTERVAL=n, TIME MARKS=YES

By default, n=1.

Use the following option to turn off all restart frame output:

*RESTART, WRITE, NUMBER INTERVAL=0

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter n in the **Intervals** column; toggle on the **Time Marks** column for each step if you want the results written at the exact time intervals

By default, n=1.

Turning off all restart frame output is not supported in Abaqus/CAE.

Specifying the frequency of output to the Abaqus/CFD state file in increments

Abaqus/CFD will write data to the restart file after each increment at which the increment number is exactly divisible by a user-specified frequency value, N, and at the end of each step of the analysis (regardless of the increment number at that time).

Input File Usage: *RESTART, WRITE, FREQUENCY=N

By default, N=1.

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter *N* in the **Frequency** column for each step

By default, *N*=0 (no restart information is written).

Specifying the frequency of output to the Abaqus/CFD state file in time intervals

Abaqus/CFD will divide the step into a user-specified number of time intervals, *n*, and write the results at the beginning of the step and at the end of each interval, for a total of *n*+1 points for the step. By default, the results will be written to the state file at the increment ending immediately after the time dictated by each interval.

If a problem precludes the analysis from continuing to completion, such as if the solution does not converge, Abaqus/CFD will attempt to save the last completed increment in the state file.

Input File Usage: *RESTART, WRITE, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter *n* in the **Intervals** column

By default, *n*=0.

Synchronizing restart information written in a co-simulation

Restart output must be synchronized between co-simulation analyses for a co-simulation restart to be successful. To achieve this synchronization, it is recommended that you request that restart data are written at a specified number of time intervals, *n*. In this case Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD will write restart information at the co-simulation target time immediately after the time dictated by each interval. If you specify the frequency of output for restart data in increments, it is very difficult to synchronize the writing of restart information, and the restart analysis may start from two different time points, possibly leading to an imbalance.

Input File Usage: Use the following option to synchronize restart information written in a co-simulation:

*RESTART, WRITE, NUMBER INTERVAL=*n*

When using the NUMBER INTERVAL parameter for a co-simulation, the TIME MARKS parameter on the *RESTART option is always set to NO.

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter *n* in the **Intervals** column

Controlling the precision of output to the Abaqus/Explicit state file

By default, Abaqus/Explicit writes to the state file in double precision when the analysis is run in double precision. Alternatively, you can choose to write data to the state file in single precision if you want to reduce the size of the state file. This option may cause noisy results between step boundaries or for the first step of a restart analysis. If Abaqus/Explicit is run in single precision, this control parameter is ignored and single precision is always used.

Input File Usage: *RESTART, WRITE, SINGLE

Abaqus/CAE Usage: Single precision state file output is not supported by Abaqus/CAE.

Overlaying results in the restart files

For an Abaqus/Standard or Abaqus/Explicit analysis, you can specify that only one increment (or one iteration in the case of a direct cyclic analysis) per step should be retained in the Abaqus/Standard restart file or Abaqus/Explicit state file, thus minimizing the size of the files. As the data are written, they overlay the data from the previous increment (or iteration), if any, written for the same step. You can specify whether or not the data should be overlaid for each step individually. Since in Abaqus/Explicit the results are written by default only at the end of the step, it is recommended to overlay the data in conjunction with specifying a number of time intervals at which data are written; in this way the data in the restart file are advanced as dictated by the number of intervals used.

To protect you from losing data if your system crashes, when Abaqus/Standard writes a frame from a given increment, it does not strictly overwrite the frame from the last saved increment. Instead, it always keeps a reserve frame and only frees a given saved frame for overwriting when the next frame is secured on the file. This reserve frame is not deleted unless the space is required for later increments. This process produces a bonus frame in the last step of an analysis if overlaying is occurring in that step and if the analysis completes successfully; users will observe that the penultimate restart frame is also retained for the last step, even though overlay is being used.

The advantage of overlaying the restart data is that it minimizes the space required to store the restart files.

Input File Usage: Use the following option in Abaqus/Standard:

*RESTART, WRITE, OVERLAY

Use the following option in Abaqus/Explicit:

*RESTART, WRITE, OVERLAY, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: click to check the **Overlay** column for each step

Restarting an analysis

You restart (continue) an analysis by specifying that the restart or state, analysis database, and part files created by the original analysis be read into the new analysis. The restart files must be saved upon completion of the first job. In Abaqus/Explicit the package (`.pac`) file and the selected results (`.sel`) file are also used for restarting an analysis and must be saved upon completion of the first job. Since restart files can be very large, sufficient disk space must be provided (in Abaqus/Standard the analysis input file processor estimates the space that is required for the restart file).

You can specify the point at which the analysis is continued in the new run, as discussed below.

An analysis cannot be restarted from the linear perturbation steps listed in “Writing restart files.”

In addition, if an Abaqus/Standard or Abaqus/Explicit analysis is terminated abruptly by an operating system command or due to a power failure, it is unlikely that the job can be recovered or restarted. In this situation, files that are open during the analysis process are not closed properly, which may result in loss of data and incomplete files.

Input File Usage: Use the following option to restart an analysis:

*RESTART, READ

When the READ parameter is included, the *RESTART option must appear as model data. It is normally the first option in the input file after the *HEADING option.

Abaqus/CAE Usage: Job module: job editor: toggle on **Restart** as the **Job Type**

Identifying the analysis to be restarted

In an Abaqus/Standard restart analysis you must specify the name of the restart file that contains the specified step and increment, iteration (for a direct cyclic analysis), or cycle (for a low-cycle fatigue analysis). In an Abaqus/Explicit or an Abaqus/CFD restart analysis you must specify the name of the state file that contains the specified step and interval.

Abaqus issues an error message if the step and increment, iteration, cycle, or interval number at which restart is requested do not exist in the specified restart or state file.

Input File Usage: Enter the following input on the command line:

abaqus job=*job-name* **oldjob**=*oldjob-name*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name***: **Restart**: toggle on **Read data from job** and enter the *oldjob-name*

Specifying the restart point

You can specify the point (step and increment, iteration, cycle, or interval) in the previous analysis from which to restart. Truncating a step in the previous analysis when you restart is discussed below.

Specifying the restart point for an Abaqus/Standard analysis (except when restarting from a direct cyclic or a low-cycle fatigue analysis)

An Abaqus/Standard analysis restarted from any analysis other than a direct cyclic or a low-cycle fatigue analysis will continue the analysis immediately after the user-specified step and increment. If you do not specify a step or increment, the analysis will restart at the last available step and increment found in the restart file.

Input File Usage: *RESTART, READ, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name***: **Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *increment*

Specifying the restart point for an Abaqus/Standard analysis restarted from a direct cyclic analysis

An Abaqus/Standard analysis restarted from a previous direct cyclic analysis can be restarted only from the end of a loading cycle. In this case you should specify the step and iteration number at which the new analysis will be resumed.

In a direct cyclic analysis that has not reached a stabilized cycle upon restart, you can increase the number of iterations or Fourier terms, thus allowing continuation of an analysis (see "Direct cyclic analysis," Section 6.2.6).

Input File Usage: *RESTART, READ, STEP=*step*, ITERATION=*iteration*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *iteration*

Specifying the restart point for an Abaqus/Standard analysis restarted from a low-cycle fatigue analysis

An Abaqus/Standard analysis restarted from a previous low-cycle fatigue analysis can be restarted only from the end of a loading cycle. In this case you should specify the step and cycle number at which the new analysis will be resumed.

Input File Usage: *RESTART, READ, STEP=*step*, CYCLE=*cycle*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *cycle*

Specifying the restart point for an Abaqus/Explicit analysis

An Abaqus/Explicit restart analysis will continue the analysis immediately after the user-specified step and interval. You must specify the step from which an Abaqus/Explicit restart analysis will continue. If you do not specify an interval from which to restart or that the current step should be terminated at a specified interval, the analysis is restarted from the last interval available in the state file for the specified step.

Input File Usage: *RESTART, READ, STEP=*step*, INTERVAL=*interval*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *interval*

Specifying the restart point for an Abaqus/CFD analysis

An Abaqus/CFD restart analysis will continue the analysis immediately after the user-specified step and increment. You must specify the step and increment from which an Abaqus/CFD restart analysis will continue. If you do not specify a step or increment, an error message will be issued.

Input File Usage: *RESTART, READ, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**, **Step name:** *step*, toggle on **Restart from increment, interval, iteration, or cycle**, and enter the *increment*

Continuing an analysis without changes

To continue an analysis without changes, only the steps subsequent to the step at which restart is being made should be defined in the restart analysis. All other information has been saved to the restart files. This feature cannot be used for an Abaqus analysis that uses the co-simulation technique and cannot be used for an Abaqus/CFD analysis.

Continuing an Abaqus/Standard analysis without changes

In Abaqus/Standard, in cases where restart is being performed simply to continue a long step (which might have been terminated because the time limit for the job was exceeded, for example), the data for the restart run may simply consist of the request to read restart data from another analysis.

Input File Usage: *RESTART, READ

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**

Continuing an Abaqus/Explicit analysis without changes

In Abaqus/Explicit, in cases where restart is being performed simply to continue a long step (which might have been terminated because a CPU time limit was exceeded, for example), do not use a restart analysis; instead, use a recover analysis. In this case no data are needed (unless user subroutines are being used).

Input File Usage: Enter the following input on the command line:

abaqus job=*job-name* **recover**

Abaqus/CAE Usage: Job module: job editor: toggle on **Recover (Explicit)** as the **Job Type**

Truncating a step

You can truncate an analysis step prior to its completion when you restart the analysis. For example, by default, if the previous analysis is an Abaqus/Standard procedure and you specify that the restart point is Step *p*, the restart analysis will restart from the last saved increment of Step *p* and continue the step to completion. However, if you specify that the restart point is increment *n* of Step *p* and that the step should be terminated before restart, the restart analysis will restart from increment *n* of Step *p*, end Step *p* at that point, and continue with newly defined steps. In this case the step from which the analysis is being restarted will be truncated at the time of restart, regardless of the step end time that had been given in the previous analysis. Thus, the step is considered to be completed even though all of the loading may not have been applied. Continuation of the analysis will be defined by history data provided in the restart run.

When you truncate an analysis step in an Abaqus/Explicit restart analysis, you must specify the interval after which the analysis should be restarted. When you truncate an analysis step in an Abaqus/CFD restart analysis, you must specify the increment after which the analysis should be restarted.

If the step from which the restart is being made completed normally, you can truncate the step to restart within the step so that you can request additional output, write to the restart file with a higher

frequency, etc. In Abaqus/Explicit it may be necessary to truncate an analysis step when an unforeseen event occurs within a step; for example, if contact surface definitions require modification due to unforeseen displacements. If the step from which the restart is being made completed normally and the restart is being made from the last increment, iteration, or interval, truncating the analysis step will have no effect.

If the restart is being made from a job that was truncated by the operating system (for example, because of insufficient disk space, run-time limit exceeded, etc.), you will usually not choose to truncate the analysis step, so that the old step will first be completed before a new step—if any exists—is started. If restart is being made from the end of a step that terminated prematurely inside Abaqus (for example, because it ran out of increments or it failed to converge), you must truncate the step and include a new step definition. If you do not truncate the step, Abaqus will try to continue the old step upon restart and will terminate the analysis in the same manner as before.

Input File Usage: Use the following option in Abaqus/Standard to restart from any analysis step other than a direct cyclic step:

*RESTART, READ, STEP=*p*, INC=*n*, END STEP

Use the following option in Abaqus/Standard to restart from a direct cyclic analysis step:

*RESTART, READ, STEP=*p*, ITERATION=*n*, END STEP

Use the following option in Abaqus/Standard to restart from a low-cycle fatigue analysis step:

*RESTART, READ, STEP=*p*, CYCLE=*n*, END STEP

Use the following option in Abaqus/Explicit:

*RESTART, READ, STEP=*p*, INTERVAL=*n*, END STEP

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*model_name*: Restart**: toggle on **Read data from job**; **Step name:** *step*; toggle on **Restart from increment, interval, iteration, or cycle**, enter the *increment*, *interval*, *iteration*, or *cycle*; and toggle on **and terminate the step at this point**

Amplitude references

Care should be taken if loads and boundary conditions refer to amplitude curves (“Amplitude curves,” Section 34.1.2). If the amplitude is given in terms of total time, the loads and boundary conditions will continue to be applied according to the amplitude definition. However, if the amplitude is given in terms of step time (default), the loads and boundary conditions will be held constant at their values at the time the step is terminated.

Temperatures, field variables, and mass flow rates applied in the old step will remain in the new step if they are not redefined. If an amplitude curve was not specified, these quantities will continue to be applied according to the default amplitude for the procedure.

Automatic stabilization in Abaqus/Standard

In Abaqus/Standard care should be exercised when automatic stabilization is active at the point at which a step is truncated. This may happen either in the middle of quasi-static procedures using automatic stabilization (see "Solving nonlinear problems," Section 7.1.1) or during contact analyses using automatic viscous damping (see "Adjusting contact controls in Abaqus/Standard," Section 36.3.6). In such cases viscous forces may be present, which will not be carried over to the subsequent step, therefore causing convergence difficulties.

In the case of quasi-static procedures using automatic stabilization it is recommended that the stabilization continue to be enforced during the following step and that you specify the damping factor directly, using the last value printed out by Abaqus/Standard in the message file. In the case of automatic viscous damping in a contact pair when contact has not yet been fully established, it is recommended that the damping be applied again, although there is no guarantee that the amount of damping applied will be the same as in the original step.

Choosing the initial time increment for an Abaqus/Standard restart analysis

In Abaqus/Standard take care in choosing the time period and initial time increment for the new step if the previous step was truncated. In transient analyses the initial time increment for the new step should be similar to the time increment that was used at the point of restart in the old step. In quasi-static analyses choose the initial time increment of the new step so that the increments in loads or prescribed boundary conditions are similar to those at the point of restart in the old step.

In a nonlinear analysis the increment of load applied in the first increment of the restart run should be similar to that applied in the last converged increment of the previous run. Let

Δl_1 = the load to be applied in the first increment of the restart run,

l_{rem} = the remaining load to be applied in the restart run,

Δt_1 = the initial time increment for the restart run, and

t_{step} = the total step time for the first step of the restart run.

The following equation can then be used to determine the initial time increment for the restart run:

$$\Delta t_1 = \frac{\Delta l_1}{l_{\text{rem}}} t_{\text{step}}.$$

Example

Suppose an Abaqus/Standard job stopped running because it reached the maximum number of increments specified for the step. The original input file was as follows:

```
*HEADING
 …
*STEP, INC=4
*STATIC, DIRECT
 0.1, 1.0
```

```
*CLOAD
 1, 2, 20.0
*RESTART, WRITE, FREQUENCY=2
*END STEP
```

This run ended at Step 1, increment 4 with a load of 8.0 applied. The following input file could be used to restart this job and to complete the loading:

```
*HEADING
*RESTART, READ, STEP=1, INC=4, END STEP
*STEP, INC=120
*STATIC, DIRECT
 0.1, 0.6
*CLOAD
 1, 2, 20.0
*END STEP
```

Notice that the concentrated load applied is the same as in the previous step.

In this example assume that a load increment of 2.0 was applied in the last converged increment of the previous run. Therefore, the initial time increment for the restart run is chosen such that the load increment applied during the first increment is also 2.0. The remaining load to be applied in the restart run is 12.0 (20.0 total − 8.0 applied in the previous run). Substitution into the equation for the initial time increment yields $\Delta t_1 = t_{\text{step}}/6$. The step time for the first step of the restarted job, t_{step}, is chosen to be 0.6 so that the total accumulated time is 1.0 when the applied load is 20.0 (at the end of the step). Thus, the initial time increment for the restart run, Δt_1, is set equal to 0.1.

Supplying additional data in the restart analysis

It is possible to define steps subsequent to the step at which restart is being made. It is also possible to supply new amplitude definitions, new surfaces, new node sets, and new element sets during the restart analysis. Existing sets cannot be modified.

In Abaqus/Standard additional surfaces defined in the model part of a restart analysis have the restriction that they can be referenced only from surface-based loading definitions (see "Loads," Section 34.4) or output requests for user-defined surface sections (see "Output to the data and results files," Section 4.1.2).

Example

For example, suppose a one-step Abaqus/Explicit job stopped prior to completion because a CPU time limit was exceeded and you have decided that a second step should be added with new boundary condition definitions. The following input file could be used to restart this job, complete the remaining part of Step 1, and complete Step 2:

```
*HEADING
*RESTART, READ, STEP=1
**
```

```
** This defines Step 2
**
*STEP
*DYNAMIC, EXPLICIT
 , .003
*BOUNDARY, OP=NEW
 …
*END STEP
```

Continuation of optional history data in restart analyses

The rules governing the continuation of optional analysis history data—loading, boundary conditions, output controls, and auxiliary controls (see "Defining a model in Abaqus," Section 1.3.1)—are the same for the steps defined in the restart analysis and the original analysis. For a discussion of the rules governing the continuation of optional history data, see "Defining an analysis," Section 6.1.2.

Prescribing predefined fields in the restart analysis

It is possible to prescribe predefined fields (see "Predefined fields," Section 34.6.1) in the restart analysis.

To specify predefined temperatures or field variables in an Abaqus/Standard restart analysis, the corresponding predefined field must have been specified in the original analysis as initial temperatures or field variables ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) or as predefined temperatures or field variables ("Predefined fields," Section 34.6.1).

Restarting with user subroutines

User subroutines are not written to the Abaqus/Standard restart file or to the Abaqus/Explicit state file. Therefore, if the original analysis contained any user subroutines, these subroutines must be included again in the restart run or when recovering additional results output from restart data (see "Recovering additional results output from restart data in Abaqus/Standard" in "Output," Section 4.1.1). These subroutines can be modified on restart; however, modifications should be made with caution because they may invalidate the solution from which the restart is being made.

Simultaneously reading and writing a restart file

You can continue a previous analysis as a restart analysis and write the results from the restart analysis to a new restart file or state file. For example, if the previous analysis is an Abaqus/Explicit procedure and in the current analysis you specify that the restart point is Step *p* and the restart output frequency is *n*, the analysis will be restarted from the last saved interval of Step *p* and restart states will be written in subsequent steps based on the new value of *n*.

To discontinue the writing of a restart file in Abaqus/Standard when you are restarting a previous analysis, specify a restart output frequency of 0; if you do not specify a frequency, the file will continue to be written at the frequency defined for the previous analysis.

The new restart file

Restart files can be very large for large models or for jobs involving many restart increments (unless you choose to overlay the restart data—see "Overlaying results in the restart files"). Therefore, the previous restart file is not copied to the beginning of the new restart file when a job is restarted: only the data at restart increments requested in the current run are saved to the new restart file. However, if an eigenfrequency extraction step ("Natural frequency extraction," Section 6.3.5) is restarted and additional eigenvalues are requested, the new restart file will contain those eigenvalues that converged during the first run as well as the additional eigenvalues.

Example: Abaqus/Standard

Suppose an Abaqus/Standard job stopped running because it ran out of disk space. The last complete information for an increment in the restart file is from Step 2, increment 4. The following two-line input file could be used to restart this job and continue writing the restart file:

```
*HEADING
*RESTART, READ, STEP=2, INC=4, WRITE
```

Example: Abaqus/Explicit

Suppose you stopped an Abaqus/Explicit job because too much output was being generated. The last information in the state file is from Step 2, Interval 4 at a time of .004. Step 2 has a time period of .010 and restart results were requested at 10 intervals. The following input file could be used to restart this job and redefine the remainder of the step with reduced output requests:

```
*HEADING
*RESTART, READ, END STEP, STEP=2, INTERVAL=4
*STEP
*DYNAMIC, EXPLICIT
 , .006
*RESTART, WRITE, NUMBER INTERVAL=2
*END STEP
```

Continuation of output upon restart

When you restart an analysis, Abaqus creates a new output database file (*job-name*`.odb`) and a new results file (*job-name*`.fil`; this file is not created in Abaqus/CFD) and writes output data to those files according to the criteria described below.

Output database (`.odb`) files

The Abaqus output database file (*job-name*`.odb`) contains results that can be used for postprocessing in Abaqus/CAE. By default, the output database file is not made continuous across restarts; Abaqus creates a new output database file each time a job is run. You can combine *X–Y* data extracted from multiple output database files in the Visualization module of Abaqus/CAE. Alternatively, you can also join field

and history results from an original analysis and a restart analysis by running the **abaqus restartjoin** execution procedure. For more information, see "Joining output database (`.odb`) files from restarted analyses," Section 3.2.23.

Results (`.fil`) files

The Abaqus results file created in Abaqus/Standard and Abaqus/Explicit (*job-name*`.fil`) contains user-specified results that can be used for postprocessing in external postprocessing packages. In Abaqus/Explicit results are also written to the selected results file (*job-name*`.sel`), which is then converted to the results file for postprocessing. See "Output," Section 4.1.1, for details.

Upon restart Abaqus/Standard will copy the information from the old results file into the results file for the new job up to the restart point and begin writing the new results to the new file following that point. Abaqus/Explicit will copy the information from the old selected results file into the selected results file for the new job up to the restart point and begin writing the new results to the new file following that point.

If the old results file is not provided, Abaqus/Standard will continue the analysis, writing the results of the restart analysis only to the new results file. Therefore, you will have segments of the analysis results in different files, which should be avoided in most cases since postprocessing programs assume that the results are in a single continuous file. You can merge such segmented results files, if necessary, by using the **abaqus append** execution procedure ("Joining results (`.fil`) files," Section 3.2.15).

Restart compatibility

A restart analysis in Abaqus/Standard can use the restart files generated from the same or any previous maintenance delivery of the same general release. Restart is not compatible between general releases. In Abaqus/Explicit and Abaqus/CFD the original analysis and the restart analysis must use precisely the same release.

A restart analysis in Abaqus and a recover analysis in Abaqus/Explicit must be run on a computer that is binary compatible with the computer used to generate the restart files.

9.2 Importing and transferring results

- "Transferring results between Abaqus analyses: overview," Section 9.2.1
- "Transferring results between Abaqus/Explicit and Abaqus/Standard," Section 9.2.2
- "Transferring results from one Abaqus/Standard analysis to another," Section 9.2.3
- "Transferring results from one Abaqus/Explicit analysis to another," Section 9.2.4

9.2.1 TRANSFERRING RESULTS BETWEEN Abaqus ANALYSES: OVERVIEW

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Transferring results between Abaqus/Explicit and Abaqus/Standard," Section 9.2.2
- "Transferring results from one Abaqus/Standard analysis to another," Section 9.2.3
- "Transferring results from one Abaqus/Explicit analysis to another," Section 9.2.4
- *IMPORT
- *IMPORT ELSET
- *IMPORT NSET
- *IMPORT CONTROLS
- *INSTANCE
- "Transferring results between Abaqus analyses," Section 16.6 of the Abaqus/CAE User's Guide

Overview

Abaqus provides the capability to import a deformed mesh and its associated material state from Abaqus/Standard into Abaqus/Explicit and vice versa. This capability is particularly useful in manufacturing problems; for example, the entire sheet metal forming process (which requires an initial preloading, forming, and subsequent springback) can be analyzed. In this case the initial preloading can be simulated with Abaqus/Standard using a static procedure and the subsequent forming process can be simulated with Abaqus/Explicit. Finally, the springback analysis can be performed with Abaqus/Standard.

Abaqus also provides the capability to transfer desired results and model information from an Abaqus/Standard analysis to a new Abaqus/Standard analysis or from an Abaqus/Explicit analysis to a new Abaqus/Explicit analysis, where additional model definitions may be specified before the analysis is continued. For example, during an assembly process an analyst may first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Standard or Abaqus/Explicit analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Standard or Abaqus/Explicit analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

For this capability to work, the same release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible. In addition, transfer of model and results can only be requested from one previous analysis; transfer from multiple analyses is not supported.

Saving the analysis results

The restart files from the original analysis contain the analysis results that are transferred from Abaqus/Standard or Abaqus/Explicit. Obtaining restart files is described in more detail in "Writing restart files" in "Restarting an analysis," Section 9.1.1; brief summaries are provided below. By default, Abaqus/Standard does not write any restart information and Abaqus/Explicit writes results at the beginning and end of each step.

Saving results from Abaqus/Standard

If the results are to be imported from an Abaqus/Standard analysis, the results from the original Abaqus/Standard job must be written to the restart (**`.res`**), analysis database (**`.mdl`** and **`.stt`**), part (**`.prt`**), and output database (**`.odb`**) files.

You can specify the increments at which restart information will be written. Restart information is always written at the end of a step in addition to the requested increments whenever you request restart data in Abaqus/Standard.

Input File Usage: ∗RESTART, WRITE, FREQUENCY=*n*

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter *n* in the **Frequency** column for each step

Saving results from Abaqus/Explicit

If the results are to be imported from an Abaqus/Explicit analysis, the results from the original Abaqus/Explicit job must be written to the state (**`.abq`**) file at the time when transfer of the state of the deformed body is required. The state (**`.abq`**), restart (**`.res`**), analysis database (**`.stt`**), package (**`.pac`**), part (**`.prt`**), and output database (**`.odb`**) files will be used for importing the results from Abaqus/Explicit.

You can specify whether the results are to be written at the exact time dictated by the specified time interval, *n*, during a step of an Abaqus/Explicit analysis or at the increment ending after the time dictated by the specified time interval. Results are always written at the end of a step, so it is not necessary to request results at the exact time intervals if results will be read only from the end of a step.

Input File Usage: Use the following option to request results at the increments ending immediately after each time interval:

∗RESTART, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=NO

Use the following option to request results at the exact time intervals:

∗RESTART, WRITE, NUMBER INTERVAL=*n*, TIME MARKS=YES

Abaqus/CAE Usage: Step module: **Output→Restart Requests**: enter *n* in the **Number Interval** column; click to check the **Time Marks** column for each step if you want the results written at the exact time intervals

Specifying the transfer of model data and results

The import capability is used to transfer model data and results from one analysis to another. The following sections describe how to specify the import request. You can import element sets from models that are not defined as assemblies of part instances, or you can import part instances from models that are defined as assemblies of part instances. In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Although elements of different types, (such as C3D4, C3D6, C3D8R, etc.) can be specified in the same element set used in a section definition, the maximum number of element types is limited to three if the model is to be used in an import analysis.

Specifying the transfer of model data and results for models that are not defined as assemblies of part instances

You can import element sets from a previous analysis to specify the transfer of model data and results for models that are not defined as assemblies of part instances. This import capability is illustrated in "Springback of two-dimensional draw bending," Section 1.5.1 of the Abaqus Example Problems Guide, and "Axisymmetric forming of a circular cup," Section 1.3.7 of the Abaqus Example Problems Guide.

Each element set to be imported must have been defined in the original analysis. You can import any element set, including nested element sets and those with overlapping elements. An imported element set can also be a subset of another imported element set. The elements in these sets as well as the element set definitions are imported. Even though an element may be included in multiple imported elements sets, each element is imported only once in the import analysis. You cannot use element sets that are internal to the original analysis.

Input File Usage: Use the following option to import element sets from a previous analysis:

*IMPORT
list of element sets that are to be imported

For example, the following input imports the element set definitions for BLANK1 and BLANK2 in addition to the elements and element set definition for BLANK:

Original analysis

```
*SHELL SECTION, MATERIAL=STEEL1, ELSET=BLANK1
.00082, 5
*SHELL SECTION, MATERIAL=STEEL2, ELSET=BLANK2
.00082, 5
*ELSET, ELSET=BLANK
BLANK1, BLANK2
```

Import analysis

```
*IMPORT
BLANK
```

To prevent any ambiguity regarding element and node definitions, the *IMPORT option must be specified before any options that define additional model data in the input file.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Repositioning elements in the model in an Abaqus/Explicit analysis

In an Abaqus/Explicit analysis you can reposition elements in the element sets from their original positions in the previous analysis to new positions in the import analysis. The new position is determined by a translation and/or rotation of the original position relative to the origin of the global coordinate system. The positioning data apply to all elements in the list of imported element sets. Element sets that require different positioning data need to be grouped separately during import.

Input File Usage: Use the following option to reposition imported element sets from a previous analysis:

*IMPORT
list of element sets that are to be imported and relocated
positioning data

Multiple *IMPORT options need to be defined for elements sets that require different positioning data.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Importing model data and results of element sets multiple times in an Abaqus/Explicit analysis

You can import model data and results of element sets from a previous analysis to an Abaqus/Explicit analysis multiple times. You must define a new name and a new position for an element set that has been imported more than once. You specify the old element set name used in the previous analysis followed by the new element set name to be used in the import analysis. The old element set name must have been defined in the previous analysis and use of internal sets is not supported.

New elements and nodes are generated with new element and node numbers for the renamed element sets. You specify element and node offsets; the new numbers are obtained by adding the offsets to the old numbers used in the previous analysis. It is your responsibility to select appropriate element and node offsets to preserve uniqueness of element and node numbering in the model.

The new position is determined as described in "Repositioning elements in the model in an Abaqus/Explicit analysis." To prevent multiple elements in the model occupying identical positions, an old element set name must not appear more than once in the list of imported element sets for each import definition. Element sets that require renaming must be grouped separately during import from those that do not require renaming.

Input File Usage: Use the following option to rename and reposition element sets to be imported from a previous analysis:

*IMPORT, RENAME, EOFFSET=*element-offset*, NOFFSET=*node-offset*

old_name_elem_set1, new_name_elem_set1
old_name_elem_set2, new_name_elem_set2
…
positioning data

Separate *IMPORT options need to be defined for element sets that require renaming and/or employ different positioning data, as shown in the following example:

```
*IMPORT, UPDATE=NO
ASSEMBLY_QUADRANT_1_LOWER,
ASSEMBLY_QUADRANT_1_UPPER,
*IMPORT, UPDATE=NO, RENAME, EOFFSET=80, NOFFSET=97
ASSEMBLY_QUADRANT_1_LOWER, ASSEMBLY_QUADRANT_2_LOWER
ASSEMBLY_QUADRANT_1_UPPER, ASSEMBLY_QUADRANT_2_UPPER
0., 0., 0.,
0., 0., 0., 0., 0., 1., 90.,
```

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Importing element set and node set definitions

All element set and node set definitions associated with the imported elements are imported by default. For models that are not defined as assemblies of part instances, you can also selectively import only specified element set or node set definitions. This capability provides a convenient way of selectively reusing the element or node sets defined in the original analysis. However, any members of such sets that do not belong to the imported elements are removed from the specified sets.

For example, suppose three element sets—**SHELL3D**, **MEMB**, and **ALL**—are defined in the original analysis. Element set **ALL** contains all of the elements in element sets **SHELL3D** and **MEMB**, as well as other elements. You choose to import only the element sets **SHELL3D** and **MEMB** (i.e., the elements in these sets as well as the element set definitions). In addition, you selectively import the element set definition **ALL** (but not the elements in this set). If element 100 belongs to element set **ALL** but not to either element set **SHELL3D** or element set **MEMB**, it will not be imported and will be removed from the list of elements belonging to element set **ALL**. The imported element set definitions are processed before any node or element definitions; therefore, even if element 100 is subsequently redefined in the import analysis, it will not belong to element set **ALL** (unless it is explicitly assigned to element set **ALL** in the import analysis).

Only node and element sets defined in the original or previous import analysis are available for importing. New sets defined during a restart run cannot be imported.

Input File Usage: Use either or both of the following options immediately following the *IMPORT option to import selected element or node set definitions:

*IMPORT ELSET
*IMPORT NSET

For models that are defined as assemblies of part instances, you cannot selectively import element and node set definitions. All element and node set definitions are imported automatically.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances. You cannot selectively import element and node set definitions in Abaqus/CAE. All element and node set definitions are imported automatically.

Importing element sets and node sets multiple times in an Abaqus/Explicit analysis

You can import element sets and node sets from a previous analysis to an Abaqus/Explicit analysis multiple times. You must define a new name for an element set or a node set that has been imported more than once. You specify the old element or node set name used in the previous analysis followed by the new element or node set name to be used in the import analysis. The old set name must have been defined in the previous analysis and use of internal sets is not supported.

New elements and nodes are generated with new element and node numbers for the renamed sets using the offsets that you specified for import; the new numbers are obtained by adding the offsets to the old numbers used in the previous analysis.

Element sets and node sets that require renaming must be grouped separately during import from those that do not require renaming.

Input File Usage: Use the following options to rename element sets and node sets to be imported from a previous analysis:

```
*IMPORT ELSET, RENAME
*IMPORT NSET, RENAME
```

The *IMPORT ELSET and *IMPORT NSET options must immediately follow an *IMPORT option with the RENAME, EOFFSET, and NOFFSET parameters specified. The offset values to be used are given by the parameter values of EOFFSET and NOFFSET of the *IMPORT option immediately preceding the *IMPORT ELSET and *IMPORT NSET options.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Specifying the transfer of model data and results for models that are defined as assemblies of part instances

You can import part instances from a previous analysis to specify the transfer of model data and results for models that are defined as assemblies of part instances. If you import more than one part instance, the part instances must be from the same output database (`.odb`) file and all import parameters must be the same for each imported part instance. Each instance name that you specify must be the same as the instance name in the original analysis. Only sets that are defined within the imported instance will be imported. Sets defined at the assembly level must be redefined in the import analysis. New set definitions and surface definitions can be added upon import. You cannot assign new sections, material orientations, normals, or beam orientations to the imported part instance.

Input File Usage: Use the following options to import a part instance from a previous analysis:

∗INSTANCE, INSTANCE=*instance-name*
Additional set and surface definitions (optional)
∗IMPORT
∗END INSTANCE

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select the instances to which the initial state should be assigned

Repositioning part instances in the model in an Abaqus/Explicit analysis

You can import a part instance in an Abaqus/Explicit analysis and specify a new position for the part instance in the imported model. The new position is determined by a translation and/or rotation of the original position relative to the origin of the assembly (global) coordinate system.

Input File Usage: Use the following options to import and reposition a part instance from a previous analysis:

∗INSTANCE, INSTANCE=*instance-name*
positioning data
Additional set and surface definitions (optional)
∗IMPORT
∗END INSTANCE

Abaqus/CAE Usage: Repositioning part instances in the model is not supported in Abaqus/CAE.

Importing part instances in the model multiple times in an Abaqus/Explicit analysis

In an Abaqus/Explicit analysis you can import a part instance from a previous analysis more than once. A part instance must be instanced separately in the assembly each time it is imported. In each instance, you must define a new name and a new position for a part instance that has been imported more than once. You specify the old name of the part instance in the previous analysis and a new name for the part instance.

The new position is determined by a translation and/or rotation of the original position relative to the origin of the assembly (global) coordinate system. Sets defined within the part instance will be imported and repositioned. Sets defined at the assembly level must be redefined in the import analysis. New set definitions and surface definitions can be added upon import. You cannot assign new sections, material orientations, normals, or beam orientations to the imported part instance. If you import more than one part instance, the part instances must be from the same output database (`.odb`) file and all import parameters must be the same for each imported part instance.

Input File Usage: Use the following options to import a part instance more than once from a previous analysis:

∗INSTANCE, NAME=*new instance-name*, INSTANCE=*old instance-name*

positioning data
Additional set and surface definitions (optional)
*IMPORT
*END INSTANCE

Abaqus/CAE Usage: Importing part instances in the model multiple times is not supported in Abaqus/CAE.

Identifying the analysis from which the data will be obtained

You must specify the name of the job from which the model and results data will be obtained.

Input File Usage: For all models you can enter the following input on the command line:

abaqus job=*job-name* oldjob=*oldjob-name*

If the **oldjob** parameter is omitted, Abaqus will prompt for the job name (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2) even if the current job is an Abaqus/Explicit analysis that uses the recover option to restart from the last available step and increment in the state file.

Alternatively, for models defined as assemblies of part instances, you can use the following option:

*INSTANCE, LIBRARY=*oldjob-name*

If you import more than one part instance, the *oldjob-name* specified by the LIBRARY parameter must be the same for each imported part instance.

If the job name is specified on the command line using the **oldjob** option, the command line specification will take precedence over the LIBRARY parameter.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: **Job name:** *output-database-name*

Importing model data

Element property definitions of imported elements can be redefined only if the reference configuration is updated (see "Updating the reference configuration") and the material state is not imported (see "Importing the material state"). In this case the material orientation definitions ("Orientations," Section 2.2.5), hourglass stiffness but not hourglass control definitions, and transverse shear stiffness definitions (in the case of shell elements) of the imported elements can also be redefined.

For other reference configuration and material state combinations, the information required to define the section for each imported element will be imported from the original analysis. Material orientations cannot be redefined in the import analysis; orientation names cannot be reused in the import analysis. For imported elements, the material orientations will be transferred from the original

analysis. Transverse shear stiffness for imported shell elements cannot be redefined; the values will be transferred from the original analysis. Hourglass stiffness for the imported elements cannot be redefined in an Abaqus/Standard import analysis; the default values will be used. The section control definitions (kinematic formulation, order of accuracy in the element formulation, and hourglass control approach) to be used for imported elements cannot be redefined (see "Transferring results between Abaqus/Explicit and Abaqus/Standard," Section 9.2.2, for details).

Mass adjustment contributions (see "Adjust and/or redistribute mass of an element set," Section 2.6.1) applied to an element set are always included when the element set is imported. There is no need to redefine these contributions in the import analysis unless different mass adjustment is required for the element set.

Nonstructural mass contributions (see "Nonstructural mass definition," Section 2.7.1) associated with an element set are not imported. These contributions need to be redefined in the import analysis if they are to be included in the model.

Only nodes associated with the imported elements are imported. New nodes can be defined in the import analysis.

Nodes or elements that use the same numbers as nodes or elements being imported can be defined provided that the reference configuration is updated, the material state is not imported, and the import is not done from an instance library. The new definitions will overwrite the imported definitions. If the reference configuration is not updated, new nodes or elements cannot use the imported node and element numbers irrespective of whether or not the material state is imported.

During results transfer from an Abaqus/Standard analysis to another Abaqus/Standard analysis or from an Abaqus/Explicit to another Abaqus/Explicit analysis, the coordinates of imported nodes can be modified from their imported values by respecifying the nodal definitions if the reference configuration is updated and the material state is not imported. This modification of the coordinates of imported nodes is not allowed during transfer of results from Abaqus/Explicit to Abaqus/Standard or vice versa.

Importing model data defined by a distribution

While transferring results from one Abaqus/Standard analysis to another Abaqus/Standard analysis, most element or material properties defined by a distribution (see "Distribution definition," Section 2.8.1) are imported along with the elements. The only exceptions are spatially varying thicknesses and orientation angles defined on the layers of composite shells and solids; in this case Abaqus issues an error message during input file preprocessing.

While transferring results from an Abaqus/Explicit analysis to an Abaqus/Standard analysis, the only spatially varying element properties defined by a distribution that can be imported are shell thicknesses and section orientations for shell and solid elements. If any other element or material properties are defined with a distribution, Abaqus issues an error message during input file preprocessing.

While transferring results from an Abaqus/Standard analysis to an Abaqus/Explicit analysis or from an Abaqus/Explicit analysis to another Abaqus/Explicit analysis, the only spatially varying element properties defined by a distribution that can be imported are shell thicknesses, section orientations for shell and solid elements, orientation angles defined for shell sections on the layers of composite shells, and section stiffness matrices specified directly for general shell sections. If any other element or material properties are defined with a distribution, Abaqus issues an error message during input file preprocessing.

Section and material properties of imported elements can be redefined with distributions only if the reference configuration is updated (see "Updating the reference configuration") and the material state is not imported (see "Importing the material state"). In this case the material orientation definitions ("Orientations," Section 2.2.5), hourglass stiffness but not hourglass control definitions, and transverse shear stiffness definitions (in the case of shell elements) of the imported elements can also be redefined.

Importing results from an Abaqus/Standard analysis (other than a direct cyclic analysis)

If the results are imported from an Abaqus/Standard analysis, you can specify the step and increment in the restart file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: *IMPORT, STEP=*step*, INCREMENT=*increment*

For models that are defined as assemblies of part instances, the *IMPORT option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *increment*

Importing results from an Abaqus/Standard direct cyclic analysis

If the results are imported from a direct cyclic analysis, you can specify the step and iteration number in the restart file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: *IMPORT, STEP=*step*, ITERATION=*iteration*

For models that are defined as assemblies of part instances, the *IMPORT option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *iteration*

Importing results from an Abaqus/Explicit analysis

If the results are imported from an Abaqus/Explicit analysis, you can specify the step and interval in the state file for which the results are to be imported. By default, the results written at the end of the analysis are imported.

Input File Usage: *IMPORT, STEP=*step*, INTERVAL=*interval*

For models that are defined as assemblies of part instances, the *IMPORT option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: select instances: **Step**: select **Specify**: *step* and **Frame**: select **Specify**: *interval*

Updating the reference configuration

Once the current model configuration of an Abaqus analysis is imported into Abaqus/Explicit or Abaqus/Standard, the analysis can be continued with or without updating the reference configuration to be the imported configuration. If the reference configuration is not updated to be the imported configuration, the displacements and strains are reported as total values relative to the original reference configuration and will, hence, be continuous. If the reference configuration is updated to be the imported configuration, displacements and strains reported in the import analysis are the total values relative to the updated reference configuration. This choice is useful if results need to be displayed relative to the imported configuration, such as may be desirable in springback analysis. The reference configuration cannot be updated if the imported analysis is geometrically linear.

The choice of whether or not to update the reference configuration can influence strain-free nodal adjustments associated with contact initialization in Abaqus/Standard. Strain-free adjustments can be used to resolve penetrations or gaps that exist in the reference configuration in Abaqus/Standard, so prior displacements are not considered by the strain-free adjustment algorithm upon import if the reference configuration is not updated. Strain-free nodal adjustments in Abaqus/Explicit are based on the current configuration rather than the reference configuration, so these adjustments are not sensitive to whether the reference configuration is updated in Abaqus/Explicit. Further details on strain-free adjustments are provided in "Default contact initialization method" in "Controlling initial contact status in Abaqus/Standard," Section 36.2.4; "Controlling initial contact status in Abaqus/Standard," Section 36.2.4; "Controlling initial contact status for general contact in Abaqus/Explicit," Section 36.4.4; and "Adjusting initial surface positions and specifying initial clearances for contact pairs in Abaqus/Explicit," Section 36.5.4.

If connector elements are imported, the configuration can be updated provided that the state is not imported.

When hyperelastic materials are imported, the configuration must be updated if the state is not imported.

Input File Usage: Use the following option to specify that the reference configuration is to be updated to the imported configuration:

∗IMPORT, STEP=*step*, UPDATE=YES

Use the following option to specify that the reference configuration should not be updated to the imported configuration:

∗IMPORT, STEP=*step*, UPDATE=NO

For models that are defined as assemblies of part instances, the ∗IMPORT option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances.

Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Initial State** for the **Types for Selected Step**: toggle **Update reference configuration** on or off

Importing the material state

You can specify whether or not the associated material state should be imported. If you choose to import the material state, the following are imported:

- stresses;
- equivalent plastic strains;
- back stresses for the kinematic hardening models;
- user-defined state variables;
- damage-related state variables for the concrete damaged plasticity model;
- damage-related state-variables for traction-separation response with cohesive elements;
- damage-related state variables for ductile metals;
- damage-related state variables for fiber-reinforced composites;
- maximum deviatoric strain energy density during deformation history for Mullins effect;
- internal strains and stresses for viscoelastic material models; and
- connector state variables such as plastic strains, frictional slip, and damage state.

Thus, the state is imported correctly for further analysis only for the following:

- linear elasticity,
- Mises plasticity (including the kinematic hardening models),
- extended Drucker-Prager plasticity,
- crushable foam plasticity,
- Mohr-Coulomb plasticity,
- critical state (clay) plasticity,
- cast iron plasticity,
- concrete damaged plasticity,
- Johnson-Cook plasticity,
- hyperelasticity (including Mullins effect),
- hyperfoam,
- viscoelasticity,
- traction-separation response with damage for cohesive elements,
- damage for ductile metals,
- damage for fiber-reinforced composites,
- connector behavior,

- materials defined in user subroutines **UMAT** and **VUMAT**, and
- materials defined using the parallel rheological framework for nonlinear viscoelastic-elastoplastic behavior.

For all other material models only stresses will be imported. No other state variables will be imported.

If the material behavior is defined in a user subroutine, you must ensure that the **UMAT** and **VUMAT** are consistent.

If connector elements are imported, the state can be imported provided that the configuration is not updated.

When hyperelastic materials are imported, the state must be imported if the configuration is not updated.

Input File Usage: Use the following option to specify that the material state should be imported:

*IMPORT, STATE=YES

Use the following option to specify that the material state should not be imported:

*IMPORT, STATE=NO

For models that are defined as assemblies of part instances, the *IMPORT option must appear within a part instance definition.

Abaqus/CAE Usage: In Abaqus/CAE you can import model data and results only from models that are defined as assemblies of part instances. Abaqus/CAE always imports the material state. If you want to import only the deformed mesh, you can import an orphan mesh from a selected step and increment of an output database; see "What kinds of files can be imported and exported from Abaqus/CAE?," Section 10.1.1 of the Abaqus/CAE User's Guide.

Defining constraints upon import

Most constraints (such as multi-point constraints and surface-based tie constraints) are not imported from the original analysis and must be redefined in the import analysis. Using the reference configuration of the original analysis without update ensures identical reproduction of these constraints in the import analysis.

If a new constraint is defined in the import analysis, it is important to ensure that the constraint is not in violation either in the reference configuration or in the starting configuration of the import analysis. These two configurations are one and the same for newly introduced nodes. If a new constraint involves nodes of the original analysis, it is appropriate to update the reference configuration for the import analysis (see "Updating the reference configuration" for more information).

In an Abaqus/Standard analysis with adaptive meshing and acoustic-to-structure tie constraints, the structural as well as the acoustic nodes may move from their initial positions. When such acoustic and structure meshes are imported from Abaqus/Standard into Abaqus/Explicit without updating the reference configuration, the acoustic elements at the interface may appear distorted when viewed in the undeformed plot mode in the Visualization module of Abaqus/CAE. This distortion appears because the

reference configuration for the acoustic nodes is updated automatically while the configuration for the non-acoustic nodes is not. The deformed plot at time=0 displays the correct mesh.

Specifying a tolerance for shell normals in the updated configuration

When the imported configuration is updated upon import, the mesh discretization may not satisfy the mesh geometry checks imposed in Abaqus/Explicit or Abaqus/Standard to evaluate whether or not a mesh is reasonable. In the case of highly warped shell elements it is possible that the normal at the center of the element that is calculated from the midsurface interpolation may differ from the normal that is interpolated from the rotated normals at the nodes. If the difference exceeds the tolerance specified, the analysis will terminate. This suggests that a fine mesh may be required to model areas of high curvature change to achieve a successful analysis.

The unit normal computed from the midsurface interpolation, $\mathbf{n_1}$, and that predicted by the interpolation of the rotated normals at the nodes, $\mathbf{n_2}$, must satisfy the condition:

$$1 - f_{tol} \leq |\mathbf{n_1} \cdot \mathbf{n_2}|,$$

where you can specify the tolerance, f_{tol}. If you do not specify a tolerance value, a default value of f_{tol} = 0.1 is used.

Input File Usage: If you update the reference configuration to be the imported configuration, you can specify a tolerance for error checking on shell normals:

*IMPORT CONTROLS, NORMAL TOL=f_{tol}

Abaqus/CAE Usage: The shell normal tolerance is not supported in Abaqus/CAE.

9.2.2 TRANSFERRING RESULTS BETWEEN Abaqus/Explicit AND Abaqus/Standard

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Transferring results between Abaqus analyses: overview," Section 9.2.1
- *IMPORT
- *IMPORT ELSET
- *IMPORT NSET
- *IMPORT CONTROLS
- *INSTANCE
- "Transferring results between Abaqus analyses," Section 16.6 of the Abaqus/CAE User's Guide

Overview

Abaqus provides the capability to import a deformed mesh and its associated material state from Abaqus/Standard into Abaqus/Explicit and vice versa. In addition, new model information can be specified during the import analysis. This capability is useful for problems that involve several analysis stages. For example, in manufacturing processes the preloading can be analyzed using Abaqus/Standard and the subsequent forming operation can be simulated using Abaqus/Explicit. Finally, the springback of the material can be performed in Abaqus/Standard.

For this capability to work, the same release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in "Transferring results between Abaqus analyses: overview," Section 9.2.1.

Specifying new data in an import analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New model definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see "Updating the reference configuration" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The usual Abaqus input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal transformation

Nodal transformations ("Transformed coordinate systems," Section 2.1.5) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying geometric nonlinearity in an import analysis

By default, Abaqus/Standard uses a small-strain formulation (i.e., geometric nonlinearity is ignored) and Abaqus/Explicit uses a large-deformation formulation (i.e., geometric nonlinearity is included). For each step of an analysis you can specify which formulation should be used; see "Geometric nonlinearity" in "General and linear perturbation procedures," Section 6.1.3, for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying initial conditions for imported elements and nodes

Initial conditions ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) can be specified on the imported elements or nodes only under certain conditions. Table 9.2.2–1 lists the initial conditions that are allowed depending on whether or not the material state is imported (see "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The reference configuration can be updated or not, as desired.

Table 9.2.2–1 Valid initial conditions.

Initial condition	Material state imported?
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Velocity	Yes or No
Void ratio	No

Procedures

Results can be imported into Abaqus/Explicit only from a general analysis step involving static stress analysis, dynamic stress analysis, or steady-state transport analysis in Abaqus/Standard. Results transfer from linear perturbation procedures ("General and linear perturbation procedures," Section 6.1.3) is not allowed.

Abaqus/Standard offers several analysis procedures that can be used in an import analysis. These procedures can be used to perform an eigenvalue analysis, static or dynamic stress analysis, buckling analysis, etc. See "Solving analysis problems: overview," Section 6.1.1, for a discussion of the available procedures.

For springback analysis of a formed component the first step in the Abaqus/Standard analysis usually consists of a static analysis procedure so that the initial out-of-balance forces can be removed gradually from the system. The removal of these forces is performed automatically by Abaqus/Standard during the first static analysis step, as described below. If the first step in the Abaqus/Standard analysis is not a static step (such as a dynamic step), the analysis proceeds directly from the state imported from the Abaqus/Explicit analysis.

Achieving static equilibrium when importing into Abaqus/Standard

When the current state of a deformed body in an explicit dynamic analysis is imported into a static analysis, the model will not initially be in static equilibrium. Initial out-of-balance forces must be applied to the deformed body in dynamic equilibrium to achieve static equilibrium. Both dynamic forces (inertia and damping) and boundary interaction forces contribute to the initial out-of-balance forces. The boundary forces are the result of interactions from fixed boundary and contact conditions. Any changes in the boundary and contact conditions from the Abaqus/Explicit analysis to the Abaqus/Standard analysis will contribute to the initial out-of-balance forces.

In general the instantaneous removal of the initial out-of-balance forces in a static analysis will lead to convergence problems. Hence, these forces need to be removed gradually until complete static equilibrium is achieved. During this process of removing the out-of-balance forces, the body will deform further and a redistribution of internal forces will occur, resulting in a new stress state. (This is essentially what occurs during "springback," when a formed product is removed from the worktools.)

When the first step in the Abaqus/Standard import analysis is a static procedure, the following algorithm is used to remove the initial out-of-balance forces automatically:

1. The imported stresses are defined at the start of the analysis as the initial stresses in the material.
2. An additional set of artificial stresses is defined at each material point. These stresses are equal in magnitude to the imported stresses but are of opposite sign. The sum of the material point stresses and these artificial stresses, thus, creates zero internal forces at the beginning of the step.
3. The internal artificial stresses are ramped off linearly in time during the first step. Thus, at the end of the step the artificial stresses have been removed completely and the remaining stresses in the material will be the residual stress state associated with static equilibrium.

Once static equilibrium has been obtained, subsequent steps can be defined using any analysis procedure that would normally follow a static analysis in Abaqus.

When the first step is not a static analysis, no artificial stress state is applied and the imported stresses are used in the internal force computations for the element.

Boundary conditions

Boundary conditions, including any connector motion, specified in the original analysis are not imported. They must be defined again in the import analysis. In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see "Amplitude curves," Section 34.1.2). If boundary conditions in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5), the same coordinate system should be defined and used in the import analysis.

For a discussion of applying boundary conditions, see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Loads

Loads, including those applied for connector actuation, defined in the original analysis are not imported. Loads may, therefore, need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see "Amplitude curves," Section 34.1.2). If point loads in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See "Applying loads: overview," Section 34.4.1, for an overview of the loading types available in Abaqus.

Predefined fields

The field variables at nodes are not imported. If the elements being imported are coupled temperature-displacement elements, the temperature is imported if the associated material state is imported. The temperature is also imported for an adiabatic analysis if the associated material state is imported. For all other cases the temperatures at nodes are not imported.

If the original analysis uses predefined temperature fields ("Predefined temperature" in "Predefined fields," Section 34.6.1) to vary the temperatures at nodes, the import analysis will not be allowed to continue. If the original analysis uses predefined field variable definitions ("Predefined field variables" in "Predefined fields," Section 34.6.1) to vary the field variables at nodes, the import analysis will

be allowed to continue only if all the elements being imported are coupled temperature-displacement elements; however, the field variables are not imported. If the original analysis uses initial temperature ("Defining initial temperatures" in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) and field variable ("Defining initial values of predefined field variables" in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) conditions, the import analysis will be allowed to continue only if all the elements being imported are coupled temperature-displacement elements.

In addition, specification of initial conditions for temperatures and field variables is not allowed in an import analysis, unless all the elements being imported are coupled temperature-displacement elements. In this case initial conditions for temperatures and field variables can be specified on the imported nodes if the reference configuration is updated and the material state is not imported. Initial temperatures can be specified in the import analysis if it is an adiabatic analysis.

Material options

All material property definitions and the orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. All relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Connector elements

When connector elements are imported, any associated connector behavior definitions are imported by default. The imported connector behavior definitions can be modified only if the state is not imported.

Mass scaling

If mass scaling ("Mass scaling," Section 11.6.1) is used in Abaqus/Explicit, the scaled masses will not be transferred to the subsequent import analysis in Abaqus/Standard. The mass of the model for the Abaqus/Standard analysis will be based on either the imported or the redefined density definitions.

Material damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to material definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It may sometimes be possible to simplify the material definition. For example, if a Mises

plasticity model was used in the Abaqus/Explicit analysis and no further plastic yielding is expected in the Abaqus/Standard analysis (as is generally the case for springback simulations), a linear elastic material can be used for the Abaqus/Standard analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for first-order continuum, modified triangular and tetrahedral elements, conventional shell, continuum shell, membrane, beam (both linear and quadratic), pipe (linear), truss, connector, rigid, and surface elements that are common to both Abaqus/Explicit and Abaqus/Standard, as defined in Table 9.2.2–2.

Table 9.2.2–2 Common element types that can be transferred between Abaqus/Explicit and Abaqus/Standard.

Common element types
CPS3, CPS3T, CPS4R, CPS4RT, CPS6M, CPS6MT
CPE3, CPE3T, CPE4R, CPE4RT, CPE6M, CPE6MT
CAX3, CAX3T, CAX4R, CAX4RT, CAX6M, CAX6MT
C3D4, C3D4T, C3D6, C3D6T, C3D8, C3D8R, C3D8T, C3D8RT, C3D10M, C3D10MT
M3D3, M3D4, M3D4R
R2D2
R3D3, R3D4
RAX2
S4, S4R, S3R, S4RT, S3RT
SC8R, SC8RT, SC6R, SC6RT
SAX1
SFM3D3, SFM3D4R
T2D2
T3D2
B21, B22, PIPE21
B31, B32, PIPE31

Common element types
CONN2D2[1], CONN3D2[1]
AC2D3, AC2D4R, AC2D4, ACIN2D2
AC3D4, AC3D6, AC3D8R, AC3D8, ACIN3D3, ACIN3D4
ACAX3, ACAX4R, ACAX4, ACINAX2
COH2D4, COHAX4, COH3D6, COH3D8
MASS, ROTARYI
[1] Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit; but not vice versa.

When S3R shell elements are imported from Abaqus/Explicit into Abaqus/Standard, they are converted into degenerated S4R elements automatically. However, when S3R shell elements are imported from Abaqus/Standard into Abaqus/Explicit, they remain S3R elements. When C3D6 and C3D6T solid elements are imported from Abaqus/Explicit into Abaqus/Standard, the results at the single point integration are applied to both integration points in Abaqus/Standard and the full integration is used automatically. However, when C3D6 and C3D6T solid elements are imported from Abaqus/Standard into Abaqus/Explicit, only the results at the first integration point are imported and are used in the reduced integration. When quadrilateral and hexahedral acoustic finite elements are imported between Abaqus/Explicit and Abaqus/Standard, they are converted to or from reduced-integration types, as required.

The following restrictions apply to the import capability:

- Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit but not vice versa. Further, if connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- Rebars defined using rebar layers (“Defining reinforcement,” Section 2.2.3) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique (“Embedded elements,” Section 35.4.1) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property (“Defining rebar as an element property,” Section 2.2.4) cannot be imported.
- Infinite elements and fluid elements cannot be imported.
- Rigid elements for which the thickness is interpolated from the nodes in an Abaqus/Explicit analysis will not be imported into Abaqus/Standard.
- A rigid body containing both deformable and rigid elements cannot be imported. A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported into Abaqus/Explicit when the element set used to define the rigid body is specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms

of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.

- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.
- Failed elements in Abaqus/Explicit will not be imported into Abaqus/Standard.
- Elements that are being removed or are inactive (see "Element and contact pair removal and reactivation," Section 11.2.1) in Abaqus/Standard will not be imported into Abaqus/Explicit.
- Rigid surfaces will not be imported.

When importing results from an Abaqus/Standard analysis to an Abaqus/Explicit analysis, each element set specified can contain only compatible element types listed in Table 9.2.2–3 and can contain at most three different element types.

Table 9.2.2–3 Compatible element types.

ACINAX2, ACIN2D2, ACIN3D3, ACIN3D4
CPE4R, CPE3, AC2D3, AC2D4
CPS4R, CPS3
CAX4R, CAX3, ACAX3, ACAX4
AC3D4, AC3D6, AC3D8, C3D8, C3D8R, C3D4, C3D6
M3D4R, M3D3, M3D4
R3D3, R3D4
S4R, S3R, SC6R, SC8R, S4
SFM3D3, SFM3D4R
CAX6M, C3D10M
C3D8T, C3D4T, C3D6T
SC6RT, SC8RT, S4T, S4RT, S3T, S3RT
MASS, ROTARYI

Using section controls in an import analysis

When transferring results between Abaqus/Standard and Abaqus/Explicit, it is important that the hourglass forces are computed consistently. The enhanced hourglass control formulation (see "Enhanced hourglass control approach in Abaqus/Standard and Abaqus/Explicit" in "Section controls," Section 27.1.4) is recommended for computing hourglass forces in the original as well as all subsequent import analyses.

Once section controls have been defined in the original analysis, they cannot be modified in any subsequent Abaqus/Standard or Abaqus/Explicit analysis. Therefore, if section controls are to be used in any one analysis in a series of import analyses, they must be specified in the very first analysis. The section controls specified for an element set in the original analysis will be used for the elements belonging to that element set in all subsequent import analyses.

Section controls other than the hourglass control formulation have appropriate defaults depending on the type of analysis and, generally, do not need to be changed. Nondefault values can be chosen subject to certain restrictions.

In an Abaqus/Standard analysis only the average strain kinematic formulation and second-order accurate element formulation are available; other kinematic formulations, element formulations, or section controls that are relevant only in an Abaqus/Explicit analysis can be specified in the Abaqus/Standard analysis. Such controls will be ignored in the Abaqus/Standard analysis but retained for the subsequent Abaqus/Explicit import analysis.

If a kinematic formulation other than average strain is used for solid elements in the Abaqus/Explicit analysis, the differences in the kinematic formulations may lead to errors in Abaqus/Standard if the elements are distorted or undergo large rotations.

Using the first-order accurate element formulation (default) in Abaqus/Explicit and the second-order accurate element formulation (the only available formulation) in Abaqus/Standard is not expected to cause significant errors, since the time increment size in Abaqus/Explicit is inherently small. One exception to this is when the Abaqus/Explicit analysis involves components undergoing several revolutions, in which case it is recommended that the second-order accurate element formulation be used in Abaqus/Explicit.

Input File Usage: Use the following options in the original analysis:

*MEMBRANE SECTION, CONTROLS=*name1*, ELSET=*elset1*
*SHELL SECTION, CONTROLS=*name2*, ELSET=*elset2*
*SHELL GENERAL SECTION, CONTROLS=*name3*, ELSET=*elset3*
*SOLID SECTION, CONTROLS=*name4*, ELSET=*elset4*

Use options similar to the following one in the original analysis:

*SECTION CONTROLS, NAME=*name1*

Abaqus/CAE Usage: Define section controls when you assign the element type in the original analysis:

Mesh module: **Mesh→Element Type**: **Element Controls**

Membrane and shell element thickness computation

The computations for membrane and shell element thicknesses are described below.

Shell elements defined using a general shell section

For shells defined using a general shell section, the current thickness is computed based on the effective Poisson's ratio, which is 0.5 by default, in both Abaqus/Explicit and Abaqus/Standard.

Input File Usage: *SHELL GENERAL SECTION, POISSON=ν_{eff}

Abaqus/CAE Usage: Property module: homogeneous or composite shell section editor: **Section integration: Before analysis**: **Advanced**: **Section Poisson's ratio**

Shell elements defined using shell sections integrated during analysis and membrane elements

For shells defined using shell sections integrated during analysis and for membranes in Abaqus/Standard, the current thickness is computed based on the effective Poisson's ratio, which is 0.5 by default. In Abaqus/Explicit, on the other hand, the computation of the thickness could be based either on the effective Poisson's ratio or the through-thickness strains, with the computation based on the through-thickness strains used by default.

If you do not specify a section Poisson's ratio for shell sections integrated during analysis or for membrane sections in an original Abaqus/Explicit or Abaqus/Standard analysis, the thickness computations in the original and all subsequent import analyses are carried out using the default methods. In other words, the thicknesses in all Abaqus/Standard analyses are computed using the default effective Poisson's ratio of 0.5, while the thicknesses in all Abaqus/Explicit analyses are computed using the through-thickness strains.

When the section Poisson's ratio is assigned a numerical value in an original Abaqus/Standard or Abaqus/Explicit analysis, the thickness computations in the original analysis and all subsequent import analyses are performed using the specified value for the effective Poisson's ratio.

Input File Usage: Use one of the following options:

*SHELL SECTION, POISSON=ν_{eff}
*SHELL SECTION, POISSON=MATERIAL
*MEMBRANE SECTION, POISSON=ν_{eff}
*MEMBRANE SECTION, POISSON=MATERIAL

Abaqus/CAE Usage: Property module:
Homogeneous or composite shell section editor: **Section integration: During analysis**: **Advanced**: **Section Poisson's ratio**
Membrane section editor: **Section Poisson's ratio**

Contact angle computation in SLIPRING-type connector elements

The contact angle, α, made by the belt wrapping around node b (see "Connection-type library," Section 31.1.5) is computed automatically in Abaqus/Explicit, ignoring the value specified within the Abaqus/Standard analysis.

Constraints

Most types of kinematic constraints (including multi-point constraints and surface-based tie constraints) specified in the original analysis are not imported and must be defined again in the import analysis; however, embedded element constraints are imported by default. See "Kinematic constraints: overview," Section 35.1.1, for a discussion of the various types of kinematic constraints.

Interactions

Contact definitions specified in the original analysis and the contact state are not imported. Contact can be defined again in the import analysis by specifying the surfaces and contact pairs; however, you may not be able to use the exact contact definitions that were used in the original analysis because of differences in the contact capabilities between Abaqus/Standard and Abaqus/Explicit.

The contact constraint enforcement may be different in Abaqus/Standard and Abaqus/Explicit. Examples of potential causes for differences include:

- Abaqus/Standard typically uses a "pure master-slave" approach, whereas Abaqus/Explicit typically uses a "balanced master-slave" approach.
- Depending on the contact formulations used, Abaqus/Standard and Abaqus/Explicit sometimes treat shell thicknesses and midsurface offsets differently.

Thus, when the contact conditions are defined in the import analysis, the contact state that existed in the previous analysis may not be reproduced at the beginning of the import analysis. This could lead to a redistribution of stresses and an analysis that differs from what you desire. In some cases this problem can be mitigated by using nondefault options, such as ignoring shell thicknesses in the contact calculations, to match behaviors in Abaqus/Standard and Abaqus/Explicit.

For a detailed description of the contact capabilities in Abaqus and the differences in the contact capabilities between Abaqus/Standard and Abaqus/Explicit, see "Contact interaction analysis: overview," Section 36.1.1.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. The output variables available in Abaqus/Standard are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1. The output variables available in Abaqus/Explicit are listed in "Abaqus/Explicit output variable identifiers," Section 4.2.2.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for **`UMAT`** and **`VUMAT`**. Similarly, for a connector behavior, the plastic relative displacement (CUP), kinematic hardening shift force (CALPHAF), overall damage (CDMG), damage initiation criteria (CDIF, CDIM, CDIP), friction accumulated slip (CASU), and connector status (CSLST, CFAILST) will be continuous.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration. Accelerations are recomputed at the start of an import analysis in Abaqus/Explicit and may be different from those obtained at the end of an Abaqus/Standard analysis. The differences in accelerations arise from the recalculation of the internal forces created by the imported stresses using the Abaqus/Explicit element formulation algorithms.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Explicit and Abaqus/Standard must be run on computers that are binary compatible.
- The capability is not available for fluid elements, infinite elements, and spring and dashpot elements. Connector elements can be imported from Abaqus/Standard to Abaqus/Explicit but not vice versa. See the discussion on "Elements" earlier in this section for further details.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- Node sets that are generated from existing element sets (see "Node definition," Section 2.1.1) must be defined in the original analysis.
- Surface definitions, contact pair definitions, and general contact definitions are not imported. Analytical rigid surfaces will not be imported.
- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, Mullins effect, hyperfoam, viscoelasticity, Mises plasticity (including the kinematic hardening models), extended Drucker-Prager plasticity, crushable foam plasticity, Mohr-Coulomb plasticity, critical state (clay) plasticity, cast iron plasticity, concrete damaged plasticity, damage for cohesive elements, damage for ductile metals, or damage for fiber-reinforced composites. See "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1, for details.
- If the state is imported for connector elements with behavior defined, the plastic displacements, the frictional slip, and the damage state are imported and the connector forces are recomputed. Some of the connector output variables, such as CU, are also recomputed on import. The recomputed variables may differ slightly at the point of import due to precision and algorithmic differences between the two solvers across import. See "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1, for details.
- Temperatures and field variables at nodes are not imported. If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it will be imported if the material state is imported. See the discussion on "Predefined fields" for details.
- Loads, boundary conditions, multi-point constraints, and equations are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint will not be imported unless the reference node is part of another element definition that is imported.

- The rigid body must be redefined if the element set containing the assigned elements is imported more than once in an Abaqus/Explicit import analysis.
- Embedded elements must be redefined if the host element set is imported more than once in an Abaqus/Explicit import analysis.
- Element and contact pair removal/reactivation ("Element and contact pair removal and reactivation," Section 11.2.1) cannot be used in the first step of an import analysis in Abaqus/Standard. It can be used in the subsequent steps.
- In a series of Abaqus/Standard and Abaqus/Explicit import analyses in the order Abaqus/Explicit(1) → Abaqus/Standard(1) → Abaqus/Explicit(2) →Abaqus/Standard(2), if elements in an element set are removed in the Abaqus/Standard(1) analysis, the subsequent Abaqus/Standard(2) import analysis does not recognize that this element set was removed in a previous analysis and fails with an error message stating that the element set is not found in the restart file. Such failures can be avoided by using flattened input files and requesting only the active element sets for import.
- Section controls must be defined in the original analysis if any of a series of import analyses uses nondefault element formulations since section controls cannot be changed in an import analysis. See the discussion on "Using section controls in an import analysis" earlier in this section.
- The symmetric model generation capability ("Symmetric model generation," Section 10.4.1) cannot be used in an import analysis in Abaqus/Standard.
- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- An Abaqus/Standard import analysis where the reference configuration is not updated is not allowed if the adaptive meshing capability ("ALE adaptive meshing: overview," Section 12.2.1) was used in the previous Abaqus/Explicit analysis.
- Mesh-independent spot welds (see "Mesh-independent fasteners," Section 35.3.4) and tie constraints (see "Mesh tie constraints," Section 35.3.1) are not imported. These constraints can be redefined in the import analysis and are formed using the reference configuration of the import model. If the reference configuration is updated, the redefined constraints may not match the old constraints exactly due to the differences in geometry. If new constraints are defined and the reference configuration of the import model is not updated, they may not initially be in compliance if the nodes involved in the constraint have nonzero displacements. This may cause numerical difficulty and potential abort of the import analysis. In this case it is recommended that you update the reference configuration upon import.
- The first step after an import when the reference conference is updated should not be used to generate a substructure.
- For beam structures that have acute curvatures and undergo large permanent changes in curvatures, slightly different equilibrated configurations will be seen when using import depending on whether or not the reference configuration is to be updated to the imported configuration (see "Updating the reference configuration" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). This configuration difference is due to beam element formulation differences between Abaqus/Standard and Abaqus/Explicit.

Input file template

Transferring results between Abaqus/Explicit and Abaqus/Standard using models that are not defined as assemblies of part instances:

Abaqus/Explicit analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Standard analysis:

```
*HEADING
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to redefine linear elasticity
*PLASTIC
Data lines to redefine Mises plasticity
 …
*BOUNDARY
```

Data lines to redefine boundary conditions
***STEP, NLGEOM=YES**
***STATIC**
…
***END STEP**

Transferring results between Abaqus/Standard and Abaqus/Explicit using models that are not defined as assemblies of part instances:

Abaqus/Standard analysis:

***HEADING**
…
***MATERIAL, NAME=mat1**
***ELASTIC**
Data lines to define linear elasticity
***PLASTIC**
Data lines to define Mises plasticity
***DENSITY**
Data line to define the density of the material
…
***BOUNDARY**
Data lines to define boundary conditions
***STEP**
***STATIC**
…
RESTART, WRITE, FREQUENCY=n*
***END STEP**

Abaqus/Explicit analysis:

***HEADING**
IMPORT, STEP=step*, **INCREMENT=***increment*, **STATE=YES, UPDATE=NO**
Data lines to specify element sets to be imported
***IMPORT ELSET**
Data lines to specify element set definitions to be imported
***IMPORT NSET**
Data lines to specify node set definitions to be imported
******
***** Optionally redefine the material block**
******
***MATERIAL, NAME=mat1**
***ELASTIC**
Data lines to redefine linear elasticity
***PLASTIC**
Data lines to redefine Mises plasticity

```
 …
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Transferring results between Abaqus/Explicit and Abaqus/Standard using models defined as assemblies of part instances:

Abaqus/Explicit analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Standard analysis:

```
*HEADING
Part definitions (optional)
```

```
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
*END INSTANCE
Additional part instance definitions (optional)
Assembly level set and surface definitions
 …
*END ASSEMBLY
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring results between Abaqus/Standard and Abaqus/Explicit using models defined as assemblies of part instances:

Abaqus/Standard analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
```

```
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*STATIC
 …
*RESTART, WRITE, FREQUENCY=n
*END STEP
```

Abaqus/Explicit analysis:

```
*HEADING
Part definitions (optional)
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name
Additional set and surface definitions (optional)
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
*END INSTANCE
Additional part instance definitions (optional)
Assembly level set and surface definitions
*END ASSEMBLY
**
*** Optionally redefine the material block
**
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to redefine linear elasticity
*PLASTIC
Data lines to redefine Mises plasticity
 …
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

9.2.3 TRANSFERRING RESULTS FROM ONE Abaqus/Standard ANALYSIS TO ANOTHER

Products: Abaqus/Standard Abaqus/CAE

References

- "Transferring results between Abaqus analyses: overview," Section 9.2.1
- *IMPORT
- *IMPORT ELSET
- *IMPORT NSET
- *IMPORT CONTROLS
- *INSTANCE
- "Transferring results between Abaqus analyses," Section 16.6 of the Abaqus/CAE User's Guide

Overview

Abaqus provides the capability to transfer desired results and model information from an Abaqus/Standard analysis to a new Abaqus/Standard analysis, where additional model definitions may be specified before the analysis is continued. For example, during an assembly process an analyst may first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Standard analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Standard analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

For this capability to work, the same release of Abaqus/Standard must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in "Transferring results between Abaqus analyses: overview," Section 9.2.1.

Comparison with the restart capability

Both the import and restart capabilities in Abaqus/Standard allow for the transfer of results and model information from one Abaqus/Standard analysis to another Abaqus/Standard analysis. However, the two capabilities have been designed for different applications.

The restart capability allows a completed Abaqus/Standard analysis to be restarted and continued. The entire model and results from the original analysis are transferred to the restart run, where additional analysis steps can be defined. Not much new model data can be specified in the restarted analysis; only model information such as new amplitude definitions, new node sets, and new element sets are allowed. Detailed information on the restart capability is given in "Restarting an analysis," Section 9.1.1.

The import capability also allows a completed Abaqus/Standard analysis to be continued. In addition, this capability allows for the analysis to be continued with only desired components from the

original analysis; the entire model need not be transferred. New model data—such as elements, nodes, surfaces, contact pairs, etc.—can be specified during the import analysis. During the import analysis it is possible to choose whether only model information from the previous analysis is to be transferred or if the results associated with that model also are to be transferred.

For situations where the goal is to continue the original analysis with no change to the model information, it is recommended that the restart capability be used. For situations where the model information requires changes, or for cases where you require control over the transfer of results, the import capability should be used.

Specifying new data in an import analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New model definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see "Updating the reference configuration" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The usual Abaqus/Standard input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal transformation

Nodal transformations ("Transformed coordinate systems," Section 2.1.5) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying geometric nonlinearity in an import analysis

By default, Abaqus/Standard uses a small-strain formulation (i.e., geometric nonlinearity is ignored). For each step of an analysis you can specify whether or not geometric nonlinearity should be included; see "Geometric nonlinearity" in "General and linear perturbation procedures," Section 6.1.3, for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying initial conditions for imported elements and nodes

Initial conditions can be specified on the imported elements or nodes only under certain conditions. Table 9.2.3–1 lists the initial conditions that are allowed depending on whether or not the material

state is imported (see "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The reference configuration can be updated or not, as desired, with one exception: for initial temperature or field variable conditions, the reference configuration must be updated.

Table 9.2.3–1 Valid initial conditions.

Initial condition	Material state imported?
Field variable	No
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Temperature	No
Velocity	Yes or No
Void ratio	No

Procedures

Results can be imported only from a general analysis step involving static stress analysis, dynamic stress analysis, steady-state transport analysis, coupled temperature-displacement analysis, or thermal-electrical-structural analysis in Abaqus/Standard. Results transfer from linear perturbation procedures ("General and linear perturbation procedures," Section 6.1.3) is not allowed.

Abaqus/Standard offers several analysis procedures that can be used in an import analysis. These procedures can be used to perform an eigenvalue analysis, static or dynamic stress analysis, buckling analysis, etc. See "Solving analysis problems: overview," Section 6.1.1, for a discussion of the available procedures.

When results are transferred from an Abaqus/Standard dynamic analysis to another Abaqus/Standard analysis where the first step is a static procedure, the initial out-of-balance forces must be removed gradually from the system. The removal of these forces is performed automatically by Abaqus/Standard during the first static analysis step, as described below. If the first step in the Abaqus/Standard analysis is not a static step (such as a dynamic step), the analysis proceeds directly from the state imported from the previous Abaqus/Standard analysis.

Achieving static equilibrium when importing from a dynamic analysis to a static analysis

When the current state of a deformed body in a dynamic analysis is imported into a static analysis, the model will not initially be in static equilibrium. Initial out-of-balance forces must be applied to the deformed body in dynamic equilibrium to achieve static equilibrium. Both dynamic forces (inertia and damping) and boundary interaction forces contribute to the initial out-of-balance forces. The boundary forces are the result of interactions from fixed boundary and contact conditions. Any changes in the boundary and contact conditions will contribute to the initial out-of-balance forces.

In general, the instantaneous removal of the initial out-of-balance forces in a static analysis will lead to convergence problems. Hence, these forces need to be removed gradually until complete static equilibrium is achieved. During this process of removing the out-of-balance forces, the body will deform further and a redistribution of internal forces will occur, resulting in a new stress state. (This is essentially what occurs during "springback," when a formed product is removed from the worktools.)

When the first step in the Abaqus/Standard import analysis is a static procedure, the following algorithm is used to remove the initial out-of-balance forces automatically:

1. The imported stresses are defined at the start of the analysis as the initial stresses in the material.
2. An additional set of artificial stresses is defined at each material point. These stresses are equal in magnitude to the imported stresses but are of opposite sign. The sum of the material point stresses and these artificial stresses, thus, creates zero internal forces at the beginning of the step.
3. The internal artificial stresses are ramped off linearly in time during the first step. Thus, at the end of the step the artificial stresses have been removed completely and the remaining stresses in the material will be the residual stress state associated with static equilibrium.

Once static equilibrium has been obtained, subsequent steps can be defined using any analysis procedure that would normally follow a static analysis.

When the first step is not a static analysis, no artificial stress state is applied and the imported stresses are used in the internal force computations for the element.

Boundary conditions

Boundary conditions specified in the original analysis are not imported; they must be redefined in the import analysis.

In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see "Amplitude curves," Section 34.1.2). If boundary conditions in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5), the same coordinate system should be defined and used in the import analysis.

For discussions on applying boundary conditions and multi-point constraints, see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1, and "Kinematic constraints: overview," Section 35.1.1.

Loads

Loads defined in the original analysis are not imported. Therefore, loads may need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see "Amplitude curves," Section 34.1.2). If point loads in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See "Applying loads: overview," Section 34.4.1, for an overview of the loading types available in Abaqus/Standard.

Predefined fields

Temperatures, whether they are prescribed or are degrees of freedom (as in a coupled thermal-stress analysis), and field variables at nodes are imported if the material state is imported.

If the reference configuration is updated and the material state is imported, the initial conditions for temperatures and field variables at the imported nodes will be reset to the imported values; for example, the thermal strains will now be measured relative to the imported temperatures. If the reference configuration is updated but the material state is not imported, the initial conditions are reset to zero. In this case you can respecify the initial conditions on the imported nodes.

If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it will be imported if the material state is imported.

Material options

All material property definitions and orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. In this case all relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Material damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to material definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It may sometimes be possible to simplify the material definition. For example, if a Mises plasticity model was used in the first Abaqus/Standard analysis and no further plastic yielding is expected in a subsequent Abaqus/Standard analysis, a linear elastic material can be used for the Abaqus/Standard analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for thermal-electrical-structural elements and a subset of the stress/displacement and coupled temperature-displacement continuum, shell, membrane, truss, rigid, and surface elements available in Abaqus/Standard. The complete list of supported elements is provided in Table 9.2.3–2. If elements that are removed (see "Element and contact pair removal and reactivation," Section 11.2.1) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.

Table 9.2.3–2 Element types that can be transferred from one Abaqus/Standard analysis to another.

Element Type	Supported Elements
Plane strain continuum	CPE3, CPE3H, CPE3T, CPE4, CPE4H, CPE4HT, CPE4I, CPE4IH, CPE4R, CPE4RHT, CPE4RT, CPE4T
	CPE6, CPE6H, CPE6M, CPE6MH, CPE6MHT, CPE6MT, CPE8, CPE8H, CPE8HT, CPE8R, CPE8RH, CPE8RHT, CPE8RT, CPE8T
Plane stress continuum	CPS3, CPS3T, CPS4, CPS4I, CPS4R, CPS4T
	CPS6, CPS6M, CPS6MT, CPS8, CPS8R, CPS8RT, CPS8T

Element Type	Supported Elements
Three-dimensional continuum	C3D4, C3D4H, C3D4T, C3D6, C3D6H, C3D6T, C3D8, C3D8H, C3D8HT, C3D8I, C3D8IH, C3D8R, C3D8RH, C3D8RHT, C3D8RT, C3D8S, C3D8HS, C3D8T, Q3D4, Q3D6, Q3D8, Q3D8H, Q3D8R, Q3D8RH
	C3D10, C3D10H, C3D10HS, C3D10M, C3D10MH, C3D10MHT, C3D10MT, C3D15, C3D15H, C3D15V, C3D15VH, C3D20, C3D20H, C3D20HT, C3D20R, C3D20RHT, C3D20RT, C3D20T, C3D27, C3D27H, C3D27RH, Q3D10M, Q3D10MH, Q3D20, Q3D20H, Q3D20R, Q3D20RH
Axisymmetric continuum	CAX3, CAX3H, CAX3T, CAX4, CAX4H, CAX4HT, CAX4I, CAX4IH, CAX4R, CAX4RH, CAX4RHT, CAX4RT, CAX4T
	CAX6, CAX6M, CAX6MH, CAX6MHT, CAX6MT, CAX8, CAX8H, CAX8HT, CAX8R, CAX8RH, CAX8RHT, CAX8RT, CAX8T
Membrane	M3D3, M3D4R
Two-dimensional rigid	R2D2
Three-dimensional rigid	R3D3, R3D4
Axisymmetric rigid	RAX2
Three-dimensional shell	S4R, S3R, S4RT, S3RT, S4T, S3T
Axisymmetric shell	SAX1
Continuum shell	SC6RT, SC8RT
Surface	SFM3D3, SFM3D4R
Two-dimensional truss	T2D2, T2D2T
Three-dimensional truss	T3D2, T3D2T
Cohesive	COH2D4, COHAX4, COH3D6, COH3D8
Inertial	MASS, ROTARYI

The following element types cannot be imported:

- Acoustic elements
- Axisymmetric-asymmetric continuum and shell elements
- Beam elements
- Connector elements
- Coupled thermal-electrical elements

- Diffusive heat transfer/mass diffusion elements and forced convection/diffusion elements
- Generalized plane strain elements
- Gasket elements
- Heat capacitance elements
- Infinite elements
- Piezoelectric elements
- Special-purpose elements
- Substructures
- User-defined elements

In addition, the following restrictions apply to the import capability:

- Rebars defined using rebar layers ("Defining reinforcement," Section 2.2.3) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique ("Embedded elements," Section 35.4.1) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property ("Defining rebar as an element property," Section 2.2.4) cannot be imported.
- A rigid body containing both deformable and rigid elements cannot be imported. A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported when the element set used to define the rigid body is specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.
- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.

Constraints

Most types of kinematic constraints specified in the original analysis are not imported and must be defined again in the import analysis; however, surface-based tie constraints are imported by default. See "Kinematic constraints: overview," Section 35.1.1, for a discussion of the various types of kinematic constraints.

Interactions

The various aspects of most surface-based mechanical contact definitions (including the surface, contact pair, and contact property definitions) can be imported. Thermal interactions, electrical interactions, and pore fluid surface interactions cannot be imported. Certain types of mechanical contact aspects—pressure, penetration loads, and debonded surfaces—cannot be imported. The most commonly used mechanical contact aspects—pressure-overclosure behavior, frictional behavior, and damping—can be imported.

The ability to import element-based and node-based surfaces is determined by whether or not the underlying elements and nodes defining these surfaces are imported. If the underlying elements or nodes of a surface are not imported, that surface will not be imported. If only some of the underlying nodes or elements used in the original definition of the surface are imported, only that part of the surface corresponding to the imported elements will be imported. Rigid surface definitions are imported when the associated slave surface is also imported. Contact pair definitions are imported provided that all the slave and master surfaces used in the original definition of the contact pair are also imported.

Contact conditions modeled with contact elements will be ignored during the transfer process.

The contact state associated with a stress/displacement analysis is imported if the material state is imported. If the reference configuration is updated, the accumulated contact strains will be set to zero. The contact state associated with thermal, electrical, or pore fluid surface interactions is not imported. The contact state associated with a crack propagation analysis is not imported; initially bonded contact surface definitions are not transferred. If a contact pair was inactive in the step from which the import was done due to the use of contact pair removal (see "Removing and reactivating contact pairs" in "Defining contact pairs in Abaqus/Standard," Section 36.3.1), it must be deactivated again in the first step of the import analysis.

Additional contact information can be defined in the import analysis by specifying new surfaces, contact pairs, and interactions. New contact pair definitions can use the imported surface interaction definitions.

For a detailed description of the contact capabilities in Abaqus/Standard, refer to "Contact interaction analysis: overview," Section 36.1.1.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. Output requests in the original analysis are not transferred to the import analysis; output requests in the import analysis have to be respecified. The output variables available in Abaqus/Standard are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for **UMAT**.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Standard must be run on computers that are binary compatible.
- The capability is not available for fluid elements; infinite elements; and spring, dashpot, and connector elements. See the discussion on "Elements" earlier in this section for further details.
- Surfaces are not imported when the model is defined as an assembly of part instances.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- The contact state associated with thermal, electrical, and pore fluid surface interactions is not imported; the contact state associated with crack propagation is not imported.
- General contact definitions are not imported.
- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, hyperfoam, viscoelasticity, Mises plasticity, and damage for cohesive elements. See "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1, for details.
- Loads, boundary conditions, multi-point constraints, and equations are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint will not be imported unless the reference node is part of another element definition that is imported.
- When you import part instances individually from a previous analysis that was defined as an assembly of part instances, reference nodes associated with rigid body or coupling constraints defined on the imported instances will not be available in the import analysis for load or boundary condition application.
- Pre-tension section definitions are not imported; they have to be redefined in the import analysis.
- The capability is not available for elements with composite solid section definitions.
- If the elements that are removed in the original analysis (see "Element and contact pair removal and reactivation," Section 11.2.1) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.
- The symmetric model generation capability cannot be used in an import analysis in Abaqus/Standard.
- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- There may be a slight discontinuity during the transfer of state variables for analyses using fully integrated, first-order continuum elements if the elements are significantly deformed and the reference configuration is updated.
- Mesh-independent spot welds (see "Mesh-independent fasteners," Section 35.3.4) are not imported. However, the spot weld reference nodes are imported and can be used to redefine spot welds in

the import analysis. The locations of the spot weld reference nodes and projection points are computed based upon the reference configuration of the import analysis. Therefore, if the deformed configuration of the imported model is significantly different from its reference configuration, it is recommended that the reference configuration be updated.

- If the value of the friction coefficient is changed from the value given in the model data of the original analysis, the changed value must be respecified in the first step of the import analysis (see "Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5).
- The capability is not available if adaptive meshing (see "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7) is used in the original analysis.
- Enriched features (see "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1) are not imported.
- Restart files from the original analysis are used in the analysis preprocessor and in the Abaqus/Standard execution in the import analysis. When the import job is run in parallel on computer clusters by using MPI-based parallelization, these restart files are copied to each host machine. The original job restart files are not decomposed to match the import analysis parallel domain and may be large relative to the local disk space available on the host machines. You can minimize this file size by requesting restart output only for the increment from which import will occur.

Input file template

Transferring results using models that are not defined as assemblies of part instances:

First Abaqus/Standard analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*RESTART, WRITE
*END STEP
```

Abaqus/Standard import analysis:

```
*HEADING
*IMPORT, STEP=step, INCREMENT=increment, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
**
*** Optionally define additional model information
**
*BOUNDARY
Data lines to redefine boundary conditions
*STEP, NLGEOM=YES
*STATIC
 …
*END STEP
```

Transferring results using models defined as assemblies of part instances:

First Abaqus/Standard analysis:

```
*HEADING
*PART, NAME=Part-1
Node, element, section, set, and surface definitions
*END PART
*ASSEMBLY, NAME=Assembly-1
*INSTANCE, NAME=i1, PART=Part-1
<positioning data>
Additional set and surface definitions (optional)
*END INSTANCE
Assembly level set and surface definitions
 …
*END ASSEMBLY
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
```

Data lines to define boundary conditions
***STEP**
***STATIC**
…
RESTART, WRITE, FREQUENCY=n*
***END STEP**

Abaqus/Standard import analysis:

***HEADING**
Part definitions (optional)
***ASSEMBLY, NAME=Assembly-1**
INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name*
Additional set and surface definitions (optional)
IMPORT, STEP=step***, INCREMENT=***increment***, STATE=YES, UPDATE=NO**
***END INSTANCE**
Additional part instance definitions (optional)
Assembly level set and surface definitions
…
***END ASSEMBLY**
******
***** Optionally define additional model information**
******
***BOUNDARY**
Data lines to define boundary conditions
***STEP, NLGEOM=YES**
***STATIC**
…
***END STEP**

9.2.4 TRANSFERRING RESULTS FROM ONE Abaqus/Explicit ANALYSIS TO ANOTHER

Products: Abaqus/Explicit Abaqus/CAE

References

- "Transferring results between Abaqus analyses: overview," Section 9.2.1
- *IMPORT
- *IMPORT ELSET
- *IMPORT NSET
- *IMPORT CONTROLS
- *INSTANCE
- "Transferring results between Abaqus analyses," Section 16.6 of the Abaqus/CAE User's Guide

Overview

Abaqus provides the capability to transfer desired results and model information from an Abaqus/Explicit analysis to a new Abaqus/Explicit analysis, where additional model definitions may be specified before the analysis is continued. For example, during an assembly process an analyst may first be interested in the local behavior of a particular component but later is concerned with the behavior of the assembled product. In this case the local behavior can first be analyzed in an Abaqus/Explicit analysis. Subsequently, the model information and results from this analysis can be transferred to a second Abaqus/Explicit analysis, where additional model definitions for the other components can be specified, and the behavior of the entire product can then be analyzed.

For this capability to work, the same release of Abaqus/Explicit must be run on computers that are binary compatible.

Information about how to transfer results between Abaqus analyses is provided in "Transferring results between Abaqus analyses: overview," Section 9.2.1.

Comparison with the restart capability

Both the import and restart capabilities in Abaqus/Explicit allow for the transfer of results and model information from one Abaqus/Explicit analysis to another Abaqus/Explicit analysis. However, the two capabilities have been designed for different applications.

The restart capability allows a completed Abaqus/Explicit analysis to be restarted and continued. The entire model and results from the original analysis are transferred to the restart run, where additional analysis steps can be defined. Not much new model data can be specified in the restarted analysis; only model information such as new amplitude definitions, new node sets, and new element sets are allowed. Detailed information on the restart capability is given in "Restarting an analysis," Section 9.1.1.

The import capability also allows a completed Abaqus/Explicit analysis to be continued. In addition, this capability allows for the analysis to be continued with only desired components from the original

analysis; the entire model need not be transferred. New model data—such as elements, nodes, surfaces, contact pairs, etc.—can be specified during the import analysis. During the import analysis it is possible to choose whether only model information from the previous analysis is to be transferred or if the results associated with that model also are to be transferred.

For situations where the goal is to continue the original analysis with no change to the model information, it is recommended that the restart capability be used. For situations where the model information requires changes, or for cases where you require control over the transfer of results, the import capability should be used.

Specifying new data in an import analysis

Additional model definitions such as new elements, nodes, surfaces, etc. can be defined during the import analysis. Initial conditions can also be specified during the import analysis.

New model definitions

New nodes, elements, and material properties can be added to the model in an import analysis once import has been specified. Nodal coordinates must be defined in the updated configuration, regardless of whether or not the reference configuration is updated on import (see "Updating the reference configuration" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The usual Abaqus/Explicit input can be used. Imported material definitions can be used with the new elements (which will need new section property definitions).

Nodal transformation

Nodal transformations ("Transformed coordinate systems," Section 2.1.5) are not imported; transformations can be defined independently in the import analysis. Continuous displacements, velocities, etc. are obtained only if the nodal transformations in the import analysis are the same as those in the original analysis. Use of the same transformations is also recommended for nodes with boundary conditions or point loads defined in a local system.

Specifying geometric nonlinearity in an import analysis

By default, Abaqus/Explicit uses a large-strain formulation. For each step of an analysis you can specify whether or not geometric nonlinearity should be included; see "Geometric nonlinearity" in "General and linear perturbation procedures," Section 6.1.3, for details.

The default value for the formulation in an import analysis is the same as the value at the time of import. Once the large-displacement formulation is used during a given step in any analysis, it will remain active in all the subsequent steps, whether or not the analysis is imported.

If the small-displacement formulation is used at the time of import, the reference configuration cannot be updated.

Specifying initial conditions for imported elements and nodes

Initial conditions can be specified on the imported elements or nodes only under certain conditions. Table 9.2.4–1 lists the initial conditions that are allowed depending on whether or not the material

state is imported (see "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1). The reference configuration can be updated or not, as desired, with one exception: for initial temperature or field variable conditions, the reference configuration must be updated.

Table 9.2.4–1 Valid initial conditions.

Initial condition	Material state imported
Field variable	No
Hardening	No
Relative density	No
Rotational velocity	Yes or No
Solution-dependent state variables	No
Stress	No
Temperature	No
Velocity	Yes or No
Void ratio	No

Boundary conditions

Boundary conditions (including connector motion) specified in the original analysis are not imported. They must be redefined in the import analysis.

In some cases nonzero boundary conditions imposed in the original analysis need to be maintained at the same values in the import analysis when the imported configuration is not updated. In such cases you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) so that the newly applied boundary conditions are applied instantaneously and held at that value for the duration of the step. Alternatively, you can refer to an amplitude curve in the boundary condition definition (see "Amplitude curves," Section 34.1.2). If boundary conditions in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5), the same coordinate system should be defined and used in the import analysis.

For discussions on applying boundary conditions and multi-point constraints, see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1, and "Kinematic constraints: overview," Section 35.1.1.

Loads

Loads, including those applied for connector actuation, defined in the original analysis are not imported. Therefore, loads may need to be redefined in the import analysis. There are no restrictions on the loads that can be applied when results are imported from one analysis to the other. In cases when the loads need to be maintained at the same values as in the original analysis, you can prescribe a constant (step function) amplitude variation for the analysis step (see "Prescribing nondefault amplitude variations" in "Defining an analysis," Section 6.1.2) to apply the loads instantaneously at the start of the step and hold them for the duration of the step. Alternatively, you can refer to an amplitude curve in the load definition (see "Amplitude curves," Section 34.1.2). If point loads in the original analysis are applied in a transformed coordinate system (see "Transformed coordinate systems," Section 2.1.5) and the loads must be maintained in the import analysis, the load application is simplified if the same coordinate system is defined and used in the import analysis.

See "Applying loads: overview," Section 34.4.1, for an overview of the loading types available in Abaqus/Explicit.

Predefined fields

Temperatures, whether they are prescribed or are degrees of freedom (as in a coupled thermal-stress analysis), and field variables at nodes are imported if the material state is imported.

If the reference configuration is updated and the material state is imported, the initial conditions for temperatures and field variables at the imported nodes will be reset to the imported values; for example, the thermal strains will now be measured relative to the imported temperatures. If the reference configuration is updated but the material state is not imported, the initial conditions are reset to zero. In this case you can respecify the initial conditions on the imported nodes.

If the temperature is a state variable (as in an adiabatic analysis where temperature is an integration point quantity), it will be imported if the material state is imported.

Material options

All material property definitions and orientations associated with imported elements are imported by default. Material properties can be changed by respecifying the material property definitions with the same material name. In this case all relevant material properties must be redefined since the old definitions that were imported by default will be overwritten. Material orientations associated with imported elements can be changed only if the reference configuration is updated and the material state is not imported; the material orientations associated with imported elements cannot be redefined for other combinations of the reference configuration and material state.

Hyperelastic materials

When hyperelastic materials are imported, the state must be imported if the configuration is not updated; if the state is not imported, the configuration must be updated.

Connector elements

When connector elements are imported, any associated connector behavior definitions are imported by default. The imported connector behavior definitions can be modified only if the state is not imported.

Material damping

The material model must be redefined in the import analysis if changes to material damping are required.

Changes to material definitions

When material definitions are changed, care must be taken to ensure that a consistent material state is maintained. It may sometimes be possible to simplify the material definition. For example, if a Mises plasticity model was used in the first Abaqus/Explicit analysis and no further plastic yielding is expected in a subsequent Abaqus/Explicit analysis, a linear elastic material can be used for the subsequent Abaqus/Explicit analysis. However, if further nonlinear material behavior is expected, no changes to the existing material definitions should be made. The history of the state variables will not be maintained if the material models are not the same in both the original analysis and the import analysis.

Elements

The import capability is available for a subset of the stress/displacement and coupled temperature-displacement continuum, shell, membrane, truss, connector, rigid, and surface elements available in Abaqus/Explicit. The complete list of supported elements is provided in Table 9.2.4–2. If elements that are removed (see "Element and contact pair removal and reactivation," Section 11.2.1) are imported, they become active in the import analysis and should be removed in the first step of the import analysis.

Table 9.2.4–2 Element types that can be transferred from one Abaqus/Explicit analysis to another.

Element Type	Supported Elements
Plane strain continuum	CPE3, CPE4R, CPE4RT, CPE6M, CPE6MT, CPE3T
Plane stress continuum	CPS3, CPS4R, CPS4RT, CPS6M, CPS6MT, CPS3T
Three-dimensional continuum	C3D4, C3D4T, C3D6, C3D6T, C3D8R, C3D8RT, C3D10M, C3D10MT, C3D8, C3D8T, C3D8I
Axisymmetric continuum	CAX3, CAX4R, CAX3T, CAX4RT, CAX6M, CAX6MT
Membrane	M3D3, M3D4 M3D4R
Two-dimensional rigid	R2D2
Three-dimensional rigid	R3D3, R3D4
Axisymmetric rigid	RAX2
Three-dimensional shell	S4R, S3R, S3, S4, S4RS, S4RSW, S3RS, S3T, S3RT, S4T, S4RT

Element Type	Supported Elements
Continuum shell elements	SC6R, SC8R, SC6RT, SC8RT
Axisymmetric shell	SAX1
Surface	SFM3D3, SFM3D4R
Two-dimensional truss	T2D2
Three-dimensional truss	T3D2
Two-dimensional beam	B21, B22
Three-dimensional beam	B31, B32
Connector elements	CONN2D2, CONN3D2
Cohesive	COH2D4, COHAX4, COH3D6, COH3D8
Infinite elements	CINPS4, CINPE4, CINAX4, CIN3D8, ACIN2D2, ACIN3D3, ACINAX2
Acoustic elements	AC2D3, AC2D4R, AC3D4, AC3D6, ACAX3, ACAX4R, AC3D8R
Inertial elements	MASS, ROTARYI

The following element types cannot be imported:

- Heat capacitance elements
- Eulerian elements (EC3D8R and EC3D8RT)
- Particle elements (PC3D)

In addition, the following restrictions apply to the import capability:

- Rebars defined using rebar layers (“Defining reinforcement,” Section 2.2.3) are imported provided the underlying elements are also imported. Rebar reinforcements defined using the embedded element technique (“Embedded elements,” Section 35.4.1) are imported if the host and embedded elements used in this definition are also imported. Rebars defined as an element property (“Defining rebar as an element property,” Section 2.2.4) cannot be imported.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- A rigid body containing both deformable and rigid elements cannot be imported. A rigid body that includes rigid elements is imported when the element set used to define the rigid body is specified for import. A rigid body that includes deformable elements is imported when the element set used to define the rigid body is specified for import. The imported rigid body definition is overwritten if it is respecified using the same element set. When the model is defined in terms of an assembly of part instances, the reference node of an imported rigid body must belong to an imported instance.

- When a rigid body is imported, any associated data such as pin node sets and tie node sets are part of the imported definition. However, these sets as imported contain only those nodes that are connected to the imported elements.

Constraints

Kinematic constraints (including multi-point constraints and surface-based tie constraints) specified in the original analysis are not imported and must be defined again in the import analysis. See "Kinematic constraints: overview," Section 35.1.1, for a discussion of the various types of kinematic constraints.

Interactions

For general contact, the contact state is imported if general contact is defined in both analyses. Contact states are not imported for elements contained in element sets that are imported more than once.

For contact defined by contact pairs, contact definitions specified in the original analysis and the contact state are not imported. Contact can be defined again in the import analysis by specifying the surfaces and contact pairs.

Additional contact information can be defined in the import analysis by specifying new surfaces, contact pairs, and interactions.

For a detailed description of the contact capabilities in Abaqus/Explicit, refer to "Contact interaction analysis: overview," Section 36.1.1.

Output

Output can be requested for an import analysis in the same way as for an analysis in which the results are not imported. Output requests in the original analysis are not transferred to the import analysis; output requests in the import analysis have to be respecified. The output variables available in Abaqus/Explicit are listed in "Abaqus/Explicit output variable identifiers," Section 4.2.2.

The values of the following material point output variables will be continuous in an import analysis when the material state is imported: stress, equivalent plastic strain (PEEQ), and solution-dependent state variables (SDV) for **VUMAT**. Similarly, for a connector behavior, the plastic relative displacement (CUP), kinematic hardening shift force (CALPHAF), overall damage (CDMG), damage initiation criteria (CDIF, CDIM, CDIP), friction accumulated slip (CASU), and connector status (CSLST, CFAILST) will be continuous.

If the reference configuration is not updated, the displacements, strains, whole element variables, section variables, and energy quantities will be reported relative to the original configuration.

If the reference configuration is updated, displacements, strains, whole element variables, section variables, and energy quantities will not be continuous in an import analysis and will be reported relative to the updated reference configuration.

Time and step number will not be continuous between the original and the import analyses if the reference configuration is updated. Time and step number will be continuous only if the reference configuration is not updated.

Limitations

The import capability has the following known limitations. Where applicable, details are given in the relevant sections.

- The same release of Abaqus/Explicit must be run on computers that are binary compatible.
- The capability is not available for spring and dashpot elements. See the discussion on "Elements" earlier in this section for further details.
- If connector elements are imported, the configuration can be updated provided that the state is not imported and the state can be imported provided that the configuration is not updated.
- Surfaces are not imported.
- All elements and nodes must be included in at least one set in the original analysis when importing part instances.
- The rigid body must be redefined if the element set containing the assigned elements is imported more than once.
- Embedded elements must be redefined if the host element set is imported more than once in an Abaqus/Explicit import analysis.
- The contact state for contact pairs is not imported.
- If the material state is imported, only stresses will be imported for material models other than those defined by linear elasticity, hyperelasticity, hyperfoam, viscoelasticity, Mises plasticity, and damage for cohesive elements. See "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1, for details. For a connector behavior, the plastic displacements, the frictional slip, and the damage state are imported and the connector forces are recomputed. See "Importing the material state" in "Transferring results between Abaqus analyses: overview," Section 9.2.1, for details.
- Loads, boundary conditions, multi-point constraints, equations, and surface-based tie constraints are not imported.
- Kinematic and distributing coupling constraints are not imported. In addition, the reference node of a coupling constraint will not be imported unless the reference node is part of another element definition that is imported.
- The results file, restart file, or output database file generated during the import analysis is not appended to the results file, restart file, or output database file of the original analysis.
- Mesh-independent spot welds (see "Mesh-independent fasteners," Section 35.3.4) and tie constraints (see "Mesh tie constraints," Section 35.3.1) are not imported. These constraints can be redefined in the import analysis and are formed using the reference configuration of the import model. If the reference configuration is updated, the redefined constraints may not match the old constraints exactly due to the differences in geometry. If new constraints are defined and the reference configuration of the import model is not updated, they may not initially be in compliance if the nodes involved in the constraint have nonzero displacements. This may cause numerical difficulty and potential abort of the import analysis. In this case it is recommended that you update the reference configuration upon import.

Input file template

Transferring results using models that are not defined as assemblies of part instances:

First Abaqus/Explicit analysis:

```
*HEADING
 …
*MATERIAL, NAME=mat1
*ELASTIC
Data lines to define linear elasticity
*PLASTIC
Data lines to define Mises plasticity
*DENSITY
Data line to define the density of the material
 …
*BOUNDARY
Data lines to define boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*RESTART, WRITE, NUMBER INTERVAL=n
*END STEP
```

Abaqus/Explicit import analysis:

```
*HEADING
*IMPORT, STEP=step, INTERVAL=interval, STATE=YES, UPDATE=NO
Data lines to specify element sets to be imported
*IMPORT ELSET
Data lines to specify element set definitions to be imported
*IMPORT NSET
Data lines to specify node set definitions to be imported
**
*** Optionally define additional model information
**
*BOUNDARY
Data lines to redefine boundary conditions
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

Transferring results using models defined as assemblies of part instances:

First Abaqus/Explicit analysis:

***HEADING**
***PART, NAME=Part-1**
Node, element, section, set, and surface definitions
***END PART**
***ASSEMBLY, NAME=Assembly-1**
***INSTANCE, NAME=i1, PART=Part-1**
<positioning data>
Additional set and surface definitions (optional)
***END INSTANCE**
Assembly level set and surface definitions
…
***END ASSEMBLY**
***MATERIAL, NAME=mat1**
***ELASTIC**
Data lines to define linear elasticity
***PLASTIC**
Data lines to define Mises plasticity
***DENSITY**
Data line to define the density of the material
…
***BOUNDARY**
Data lines to define boundary conditions
***STEP**
***DYNAMIC, EXPLICIT**
…
RESTART, WRITE, NUMBER INTERVAL=n*
***END STEP**

Abaqus/Explicit import analysis:

***HEADING**
Part definitions (optional)
***ASSEMBLY, NAME=Assembly-1**
INSTANCE, INSTANCE=i1, LIBRARY=oldjob-name*
Additional set and surface definitions (optional)
IMPORT, STEP=step***, INTERVAL=***interval***, STATE=YES, UPDATE=NO**
***END INSTANCE**
Additional part instance definitions (optional)
Assembly level set and surface definitions
…

```
*END ASSEMBLY
**
*** Optionally define additional model information
**
*BOUNDARY
```
Data lines to define boundary conditions
```
*STEP
*DYNAMIC, EXPLICIT
 …
*END STEP
```

10. Modeling Abstractions

Substructuring 10.1
Submodeling 10.2
Generating matrices 10.3
Symmetric model generation, results transfer, and analysis of cyclic symmetry models 10.4
Periodic media analysis 10.5
Meshed beam cross-sections 10.6
Modeling discontinuities as an enriched figure using extended finite element method 10.7

10.1 Substructuring

- “Using substructures,” Section 10.1.1
- “Defining substructures,” Section 10.1.2

10.1.1 USING SUBSTRUCTURES

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining substructures," Section 10.1.2
- *SLOAD
- *SUBSTRUCTURE PATH
- *SUBSTRUCTURE PROPERTY
- *SUBSTRUCTURE DAMPING
- *SUBSTRUCTURE DAMPING CONTROLS
- *SUBSTRUCTURE MODAL DAMPING

Overview

Substructures:

- allow a collection of elements to be grouped together and all but the retained degrees of freedom eliminated on the basis of linear response within the group;
- are used in the same manner as any of the standard element types in the Abaqus element library once created as described in "Defining substructures," Section 10.1.2;
- can be used in stress/displacement and in coupled acoustic-structural analyses;
- have linear response but allow for large translations and large rotations;
- are particularly useful in cases where identical pieces appear several times in a structure (such as the teeth of a gear) since a single substructure can be used repeatedly;
- can be translated, rotated with respect to the global system, and reflected in a plane when they are used;
- are connected to the rest of the model by the retained degrees of freedom at the retained nodes;
- may contain a set of internal load cases and boundary conditions that can be activated and scaled;
- can include dynamic effects by including retained eigenmodes; and
- appear to the rest of the model as a stiffness, optional mass, damping, and a set of scalable load vectors.

Substructures

Substructures are collections of elements from which the internal degrees of freedom have been eliminated. Retained nodes and degrees of freedom are those that will be recognized externally at the usage level (when the substructure is used in an analysis), and they are defined during generation of

the substructure. Factors that determine how many and which nodes and degrees of freedom should be retained are discussed below and in "Defining substructures," Section 10.1.2.

Substructures versus superelements

In the finite element literature substructures are also referred to as *superelements*. In earlier releases of Abaqus a distinction was made between substructures and superelements. The term "substructure" was used when it was needed to make clear that results were recovered within the substructure. Otherwise, both terms were used interchangeably. To avoid confusion, the term "superelement" will no longer be used.

Why use substructures?

There are a number of good reasons to use substructures.

Computational advantages

- System matrices (stiffness, mass) are small as a result of substructuring. Subsequent to the creation of the substructure, only the retained degrees of freedom and the associated reduced stiffness (and mass) matrix are used in the analysis until it is necessary to recover the solution internal to the substructure.
- Efficiency is improved when the same substructure is used multiple times. The stiffness calculation and substructure reduction are done only once; however, the substructure itself can be used many times, resulting in a significant savings in computational effort.
- Substructuring can isolate possible changes outside substructures to save time during reanalysis. During the design process large portions of the structure will often remain unchanged; these portions can be isolated in a substructure to save the computational effort involved in forming the stiffness of that part of the structure.
- In a problem with local nonlinearities, such as a model that includes interfaces with possible separation or contact, the iterations to resolve these local nonlinearities can be made on a very much reduced number of degrees of freedom if the substructure capability is used to condense the model down to just those degrees of freedom involved in the local nonlinearity.

Organizational advantages

- Substructuring provides a systematic approach to complex analyses. The design process often begins with independent analyses of naturally occurring substructures. Therefore, it is efficient to perform the final design analysis with the use of substructure data obtained during these independent analyses.
- Substructure libraries allow analysts to share substructures. In large design projects large groups of engineers must often conduct analyses using the same structures. Substructure libraries provide a clean and simple way of sharing structural information.
- Many practical structures are so large and complex that a finite element model of the complete structure places excessive demands on available computational resources. Such a large linear problem can be solved by building the model, substructure by substructure, and stacking these

level by level until the whole structure is complete and then recovering the displacements and stresses locally, as required.

Valid procedures

Substructures can be used without restriction in the following procedures:

- "Static stress analysis," Section 6.2.2
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Direct-solution steady-state dynamic analysis," Section 6.3.4
- "Natural frequency extraction," Section 6.3.5
- "Complex eigenvalue extraction," Section 6.3.6
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- "Transient modal dynamic analysis," Section 6.3.7

Substructures can also be used in the following procedures, but recovery of eliminated degrees of freedom is not supported:

- "Response spectrum analysis," Section 6.3.10
- "Random response analysis," Section 6.3.11

Using substructures in static analysis

Substructuring introduces no additional approximation in linear static structural analysis: the substructure is an exact representation of the linear, static behavior of its members. The principal drawback to the use of substructures in stress/displacement analyses is that a substructure's stiffness matrix is fully populated (no zero terms) and, therefore, may be very large if the substructure has a large number of retained degrees of freedom. This, in turn, may mean that the wavefront of the model within which substructures are used may be large, thus leading to long computer times to solve the equations.

This difficulty can often be avoided by choosing the substructure's boundaries carefully or by reusing several smaller substructures rather than a single larger substructure. In some cases it is possible to take advantage of the fact that Abaqus/Standard allows individual degrees of freedom to be retained, rather than the whole set of degrees of freedom at a node. For example, in contact problems without friction only the displacement component normal to the surface need be retained for the contact solution. Nodal transformations can be helpful in orienting the displacement components at surface nodes for this purpose (see "Transformed coordinate systems," Section 2.1.5).

In a static analysis involving a substructure containing acoustic elements, the results will differ from the results obtained in an equivalent static analysis without substructures. The acoustic-structural coupling is taken into account in the substructure (leading to hydrostatic contributions of the acoustic fluid), while the coupling is ignored in a static analysis without substructures.

Using substructures in dynamic analysis

Substructures introduce approximations in dynamic analysis. The default approach to the dynamic representation of a substructure is to reduce its mass and damping matrix with the same transformation

as is used for its stiffness matrix, which is known as "Guyan reduction." This approach assumes that the response between the eliminated and retained degrees of freedom is correctly represented by the static modes only. This representation may not be accurate if dynamic modes within the substructure are important. The dynamic representation may be improved for Guyan reduction by retaining additional physical degrees of freedom that are not required to connect the substructure to the rest of the model. For example, if the substructure is a plate or a beam, some transverse displacements (and, perhaps, in-surface rotation components) might be included as retained degrees of freedom for this purpose. For more details regarding Guyan reduction, see "Substructuring and substructure analysis," Section 2.14.1 of the Abaqus Theory Guide.

"Dynamic mode addition" can be used as an alternative to Guyan reduction. This approach involves adding generalized degrees of freedom associated with the eigenmodes extracted for the substructure, with all of the physical retained degrees of freedom automatically constrained. This improves dynamic behavior, but it introduces the additional cost of extracting the eigenmodes for the constrained substructure. For more details regarding dynamic mode addition, see "Substructuring and substructure analysis," Section 2.14.1 of the Abaqus Theory Guide.

The reduction methods can be applied simultaneously to different substructures within the same structure. Definition of the reduced mass matrix is discussed further in "Defining substructures," Section 10.1.2.

Using substructures in geometrically nonlinear stress/displacement analysis

Substructures may undergo large motions if geometric nonlinearities are considered in a particular stress/displacement analysis (see "Static stress analysis procedures: overview," Section 6.2.1). Abaqus/Standard will account for the large rigid body rotations and translations of the substructure. However, the substructure is assumed to undergo small (linear elastic) deformations at all times during the geometrically nonlinear analysis. An equivalent rigid body rotation for each substructure is computed during each equilibrium iteration using the retained nodes of the substructure. The substructure's mass, damping, stiffness matrix (including the retained eigenmodes), and force vectors are then rotated appropriately using the equivalent rigid body rotation. Appropriate (rotated) linear perturbation displacements (strain-inducing displacements relative to the rotating reference configuration) are used to compute the internal force associated with the substructure. Degrees of freedom at a node should not be retained selectively if the substructure is to be used in geometrically nonlinear analysis. Coupled acoustic-structural substructures should not be used in geometrically nonlinear analyses.

Comparison with component mode synthesis

The component mode synthesis method has been developed to permit the structure to be subdivided into components (substructures), with most of the analysis being done on the smaller components to develop an approximate model for the entire structure.

The substructures in Abaqus/Standard are, in fact, a particular case of the component mode synthesis method extended to allow for large rotations and translations of the substructure (component) in the geometrically nonlinear analysis. The component mode synthesis method is based on the assumption that the small deformations of a substructure can be modeled using a collection of modes. The most frequently used modes in the literature are typically referred to as follows:

- constraint modes, which are static shapes obtained by giving each retained degree of freedom in the substructure a unit displacement while holding all other retained degrees of freedom fixed;
- fixed-interface normal modes, which are obtained by fixing the retained degrees of freedom and computing the eigenmodes of the substructure;
- free-interface normal modes, which are obtained by computing the eigenmodes of the substructure with free (not fixed) retained degrees of freedom; and
- mixed-interface normal modes, which are obtained by fixing a part of the retained degrees of freedom and computing the eigenmodes of the substructure.

The constraint modes are precisely the static modes (see "Substructuring and substructure analysis," Section 2.14.1 of the Abaqus Theory Guide) used by Abaqus/Standard. You include these modes in the substructure's representation by specifying the degrees of freedom that are to be retained (see "Defining the retained degrees of freedom" in "Defining substructures," Section 10.1.2). The fixed-interface, free-interface, or mixed-interface normal modes are the eigenmodes extracted in the eigenfrequency extraction step at the generation level, and these modes represent particular cases of substructure dynamic modes allowed in Abaqus (see "Defining the generalized degrees of freedom" in "Defining substructures," Section 10.1.2). You include the dynamic modes in the substructure's representation by selecting the eigenmodes to be used.

Including substructures in a model

When a substructure is used in a model, it is assigned an element number and defined by nodes just like any other element.

Use an element definition ("Element definition," Section 2.2.1) with a substructure identifier to include substructures in the definition of another substructure (nested substructure) or in an analysis model. The substructure can be read from a substructure library. A maximum of 500 libraries can be accessed to read substructure data within a given analysis.

In the element definition you define the substructure's element number at the usage level and assign node numbers to the substructure's retained nodes. More than one substructure can be defined per element definition.

Once a substructure has been introduced by an element definition, it is treated like any other element in the model, except that its response can be linear only (although it can be used as a part of a model that includes nonlinear effects, including large displacements).

Using substructures requires that the substructure database be available. All the files generated for a substructure including the **`.sup`** and **`.sim`** files and/or the **`.prt`**, **`.stt`**, and **`.mdl`** files must be available.

Input File Usage: Use the following option to include one or more substructures in a model:

∗ELEMENT, TYPE=Z*n*

Abaqus/CAE Usage: Use the following option to include one substructure in a model:

All modules: **File→Import→Part**: **File Filter**: **Substructure**

Repeat the import process for each substructure that you want to include in the model.

Ordering of the substructure nodes on the usage level

The node numbers that are used when a substructure is created and the node numbers that are associated with the substructure when it is used are entirely independent. The ordering of the retained nodes when the substructure is used can be defined in two different ways:

1. The nodes can be provided in the same order that they were listed in the substructure definition. In this case you must prevent the sorting of the retained nodes when you specify the retained degrees of freedom (see "Preventing the degrees of freedom from being sorted" in "Defining substructures," Section 10.1.2). Duplicate nodes are not combined if the retained nodes are not sorted. Therefore, if the same nodes are specified more than once in the list of retained degrees of freedom to retain different degrees of freedom, the corresponding nodes at the usage level must appear the same number of times.
2. The substructure nodes must be specified in the same order as the retained nodes sorted into ascending numerical order according to their numbers used within the substructure. This approach is the default when you specify the retained degrees of freedom.

In either case you must ensure that the nodes match up properly whenever a substructure is used.

Reading the substructure definition from a substructure library

You can read the substructure definition from a substructure library.

Input File Usage: *ELEMENT, TYPE=Z*n*, FILE=*substructure_library_name*

Abaqus/CAE Usage: Substructure libraries are not supported in Abaqus/CAE.

Interpreting the model output in the data file

If model definition data are written to the data file ("Controlling the amount of analysis input file processor information written to the data file" in "Output," Section 4.1.1), substructure instances are identified in the data (`.dat`) file by the substructure identifier followed by an F and two digits that indicate the substructure library number. The full name of the substructure library associated with this number is also contained in the model output.

Defining the substructure's properties

You associate a property definition with each substructure in the model. The property definition serves the following purposes:

1. It defines any translation, rotation, and reflection of the substructure at the usage level.
2. It allows a tolerance to be set to ensure that the coordinates of the usage level nodes match the coordinates of the nodes used to generate the substructure.
3. It controls using various sources of substructure damping in the dynamic analysis at the usage level.

Input File Usage: *SUBSTRUCTURE PROPERTY, ELSET=*name*

Abaqus/CAE Usage: Use the following options to define translation and rotation of the substructure:

Assembly module: **Instance→Translate** or **Instance→Rotate**

Reflection of the substructure is not supported in Abaqus/CAE.

Use the following option to apply constraints that connect the retained nodes with the usage level nodes:

Interaction module: **Constraint→Create**

Translating, rotating, and reflecting a substructure

Translation, rotation, and/or reflection (in that order) of a substructure can be specified in a substructure property definition.

Specify a translation by giving a translation vector. Specify a rotation by giving two points, a and b, defining a rotation axis plus a right-handed angular rotation around that axis. Specify a reflection by giving three non-colinear points in the reflection plane.

A translation does not affect the substructure's stiffness or mass: the principal reason to apply a translation is to enable the tolerance check on nodal coordinates as discussed later. Rotation and/or reflection of a substructure affect the substructure's stiffness and mass. The substructure load case definitions are rotated and/or reflected in the same way as the substructure's stiffness and mass; therefore, all loads within substructure load cases are applied in the local directions associated with the substructure when it was created.

For distributed loads (for example, pressure loading of a surface) this application is precisely what is desired. However, distributed body forces in coordinate directions (BX, BY, BZ) are applied in the substructure's local directions instead of in the global directions, which may not be what is needed. Similarly, distributed loadings that depend on position (for example, hydrostatic pressure or centrifugal loads) are based on the substructure's local coordinates and not on the substructure position during usage. Be careful to ensure that loading of a rotated or shifted substructure is correct for its usage.

Whenever a substructure is translated, rotated, and/or reflected, the degrees of freedom at any retained nodes are with respect to the coordinate directions at the usage level. Therefore, if all of the degrees of freedom of a node are not retained or if a two-dimensional substructure is used in a three-dimensional model with rotation out of the x–y plane, additional degrees of freedom may be activated due to rotation and/or reflection. Be careful to check the validity of the substructure usage in such cases.

Setting a tolerance on the substructure nodes

One difficulty with using large substructures is ensuring that the retained nodes in the substructure are connected to the correct nodes on the usage level (after substructure translation, rotation, and/or reflection, if applicable). Therefore, Abaqus/Standard checks that the coordinates of the retained nodes match the coordinates of the corresponding nodes on the usage level. A substructure does not require any coordinates on the usage level because it consists only of a stiffness matrix, a mass matrix, and a number of load cases. Nevertheless, it is usually a good check of a model's validity to verify that the substructure and the model into which it is introduced are geometrically consistent.

To check the coordinates, you can set a tolerance on the distance between usage level nodes and the corresponding substructure nodes. This tolerance indicates the largest deviation allowable before a warning is issued. If you do not specify this tolerance, the default is to use a tolerance of 10^{-4} times the largest overall dimension within the substructure. If you specify a tolerance of 0.0, the position of the retained nodes is not checked.

The geometric check is based on the coordinates of the retained nodes after translation, rotation, and/or reflection of the substructure at the usage level; motions of these nodes that occur as a result of geometrically nonlinear preloading during generation of the substructure are not considered in this check.

Input File Usage: *SUBSTRUCTURE PROPERTY, ELSET=*name*, POSITION TOL=*tolerance*

Abaqus/CAE Usage: Assembly module: **Instance→Translate** and **Instance→Rotate**

Defining substructure damping

Defining substructure damping at the substructure usage level means defining viscous and structural damping matrices for the finite elements associated with the substructures. Abaqus allows you to choose a particular source of damping for a substructure, to add several sources, or to exclude the damping effects for a substructure at the usage level. All options defining the substructure damping belong to a substructure property definition and affect only the finite elements of the substructure type associated with the substructure property.

Sources of substructure damping

You can choose to model the damping of a substructure at the usage stage by using the reduced substructure damping matrices computed during the generation stage and stored on the substructure database. We denote the reduced viscous damping matrix of a substructure as $\overline{C}_{reduced}$ and the reduced structural damping matrix of a substructure as $\overline{S}_{reduced}$. Alternatively, you can introduce the stiffness and mass proportional damping matrices by multiplying the reduced substructure stiffness and mass matrices, $\overline{K}_{reduced}$ and $\overline{M}_{reduced}$, respectively, with the factors defined within the substructure property definition at the usage stage. You can also combine both damping sources or exclude the effects of damping altogether at the usage level. Finally, you can introduce viscous and structural modal damping matrices for a substructure specifying damping coefficients for the substructure eigenmodes calculated at the generation stage and stored on the substructure database.

The substructure modal damping contributes to the damping matrices for the finite elements associated with a substructure, and it can be used instead of or together with the other substructure damping sources. To define the substructure modal damping matrix, you specify the diagonal damping matrix on the substructure modal subspace. This matrix is transformed to the substructure degrees of freedom space to be added to the damping matrix of the finite element associated with the substructure.

Controlling the sources of substructure viscous damping

In the general case the substructure type element viscous damping matrix at the usage stage is defined by the following matrix equation:

$$\overline{C} = \overline{C}_{reduced} + \alpha\overline{M} + \beta\overline{K} + \overline{C}_{modal}.$$

You can specify substructure viscous damping using substructure damping controls and/or substructure viscous modal damping. If you specify substructure viscous modal damping, it is used in combination with all other activated viscous damping sources to form the viscous damping matrix of the finite element. Defining the substructure viscous modal damping is discussed in more detail in "Defining substructure viscous modal damping" below.

Input File Usage: To activate only the generated condensed viscous damping matrix of the substructure (the first term on the right-hand side), use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, VISCOUS=ELEMENT

To activate only the stiffness and mass proportional substructure viscous damping, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, VISCOUS=FACTOR

To activate the combined generated and proportional substructure viscous damping matrix, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, VISCOUS=COMBINED

To exclude the effects of generated and proportional substructure viscous damping altogether at the usage level, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, VISCOUS=NONE

To specify the substructure viscous modal damping matrix (the last term on the right-hand side), use the following option:

*SUBSTRUCTURE MODAL DAMPING

Abaqus/CAE Usage: Substructure damping controls and substructure modal damping are not supported in Abaqus/CAE.

Controlling the sources of substructure structural damping

In the general case the substructure type element structural damping matrix is defined by the following equation:

$$\overline{S} = \overline{S}_{reduced} + \gamma \overline{K}_{reduced} + \overline{S}_{modal}.$$

You can specify substructure structural damping using substructure damping controls and/or substructure structural modal damping. If you specify substructure structural modal damping, it is used in combination with all other activated structural damping sources to form the structural damping matrix of the finite element. Defining the substructure structural modal damping is discussed in more detail in "Defining substructure structural modal damping" below.

Input File Usage: To activate only the generated reduced structural damping matrix of the substructure (the first term on the right-hand side), use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, STRUCTURAL=ELEMENT

To activate only the stiffness proportional substructure structural damping matrix, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, STRUCTURAL=FACTOR

To activate the combined generated and stiffness proportional structural damping matrix, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, STRUCTURAL=COMBINED

To exclude the generated and stiffness proportional structural damping matrices, use the following option:

*SUBSTRUCTURE DAMPING CONTROLS, STRUCTURAL=NONE

To specify the substructure structural modal damping matrix (the last term on the right-hand side), use the following option:

*SUBSTRUCTURE MODAL DAMPING

Abaqus/CAE Usage: Substructure damping controls and substructure modal damping are not supported in Abaqus/CAE.

Defining substructure damping factors

By default, the damping factors, α and β, and the structural damping factor, γ, used to define stiffness proportional and mass proportional damping for a substructure are zeros.

Input File Usage: Use the following options to define the values of the substructure damping factors at the usage level:

*SUBSTRUCTURE DAMPING, ALPHA=α, BETA=β, STRUCTURAL=γ

Abaqus/CAE Usage: Defining substructure damping factors is not supported in Abaqus/CAE.

Defining substructure viscous modal damping

Substructure viscous modal damping is defined for the substructure eigenmodes extracted at the substructure generation level. The mode numbers and the eigenfrequencies used to define substructure viscous modal damping come from the solution of the substructure eigenvalue problem at the generation level.

Input File Usage: Use the following option to define the fraction of critical damping for a substructure by specifying mode numbers:

*SUBSTRUCTURE MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=MODE NUMBERS

Use the following option to define the fraction of critical damping for a substructure by specifying a frequency range:

*SUBSTRUCTURE MODAL DAMPING, VISCOUS=FRACTION OF CRITICAL DAMPING, DEFINITION=FREQUENCY RANGE

Use the following option to define substructure modal Rayleigh damping by specifying the substructure mode numbers:

*SUBSTRUCTURE MODAL DAMPING, VISCOUS=RAYLEIGH, DEFINITION=MODE NUMBERS

Use the following option to define substructure modal Rayleigh damping by specifying a frequency range:

*SUBSTRUCTURE MODAL DAMPING, VISCOUS=RAYLEIGH, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Substructure modal damping is not supported in Abaqus/CAE.

Defining substructure structural modal damping

Substructure structural modal damping is defined for the substructure eigenmodes extracted at the substructure generation level. The mode numbers and the eigenfrequencies used to define substructure structural modal damping come from the solution of the substructure eigenvalue problem at the generation level.

Input File Usage: Use the following option to define substructure structural modal damping by specifying mode numbers:

*SUBSTRUCTURE MODAL DAMPING, STRUCTURAL, DEFINITION=MODE NUMBERS

Use the following option to define substructure structural modal damping by specifying a frequency range:

*SUBSTRUCTURE MODAL DAMPING, STRUCTURAL, DEFINITION=FREQUENCY RANGE

Abaqus/CAE Usage: Substructure modal damping is not supported in Abaqus/CAE.

Defining kinematic constraints and transformations

All kinematic boundary conditions, MPCs, and transformations can be applied to retained degrees of freedom at the usage level. These specifications can be changed from step to step in the usual way. In this respect substructures and their retained nodes act in an identical manner to regular elements and their nodes.

Defining transformations at retained nodes

If a nodal transformation (“Transformed coordinate systems,” Section 2.1.5) is used during substructure generation at a retained node, the transformations are built into the substructure. This creates an inconsistency when the substructure node is attached to a standard Abaqus element since Abaqus/Standard uses the retained degrees of freedom directly without checking their directions. Therefore, it is suggested that this situation be avoided.

If a nodal transformation must be used, the resulting inconsistency can be resolved by retaining all degrees of freedom at the node and applying a linear constraint equation (“Linear constraint equations,”

Section 35.2.1) as follows. At any point where such a transformed substructure node is attached to a global model, define two coincident nodes on the usage level, P and Q, for example. Use node P for the substructure at the usage level (defined with an element definition); the local directions of the degrees of freedom are already built in at this node. Use node Q for all standard Abaqus elements attached to this point. Use a local transformation at node Q to transform the degrees of freedom to the same local directions that are built-in for node P. Now use a linear constraint equation to equate the individual degrees of freedom at nodes P and Q.

Applying loads to a substructure

Loads or boundary conditions that are to be applied to a substructure within an analysis (at the usage level) must be specified during the substructure generation step by defining a substructure load case or by requesting that the substructure's gravity load vectors be calculated (see "Defining substructure load cases for subsequent loading in an analysis" in "Defining substructures," Section 10.1.2). A load case can be made up of any combination of loadings and nonzero boundary conditions, and multiple load cases can be defined for any given substructure.

When you activate load cases created for a substructure, you specify the element number or element set name of the substructures, the associated substructure load case names, and the scaling multipliers for the specified substructure load case loads and/or boundary conditions. To reproduce the loading conditions defined during substructure generation exactly, use a magnitude of 1.0.

Boundary conditions specified during a substructure's generation are always present, whether the substructure load case that they are part of is active or not. They are effectively built into the substructure and can only be scaled if desired but not removed. See "Defining substructures," Section 10.1.2, for further information about defining boundary conditions in substructures.

Input File Usage: Use the following option to activate a substructure load case:

*SLOAD

Abaqus/CAE Usage: Use the following option to activate a substructure load case:

Load module: load editor: **Category**: **Mechanical**: **Types for Selected Step**: **Substructure load**

Modifying or removing load cases

By default, substructure loads are applied as modifications of existing loads or in addition to any loads previously defined. You can remove all previously defined loads and, optionally, specify new loads when you activate a load case. Boundary conditions cannot be removed.

Input File Usage: Use the following option to modify load cases:

*SLOAD, OP=MOD

Use the following option to remove load cases:

*SLOAD, OP=NEW

Abaqus/CAE Usage: Use the following option to modify load cases:

Load module: **Load Case Manager**: click **Edit**

Use the following option to remove load cases:

Load module: **Load Case Manager**: click **Delete**

Specifying time-dependent load cases

The magnitude of substructure loads can be varied with time by referring to an amplitude definition ("Amplitude curves," Section 34.1.2).

Input File Usage: Use the following options to define time-dependent load cases:

*AMPLITUDE, NAME=*amplitude*
*SLOAD, AMPLITUDE=*amplitude*

Abaqus/CAE Usage: Use the following options to define time-dependent load cases:

Load module: amplitude editor: **Create Amplitude**: **Amplitude**: *amplitude*

Load module: load editor: **Category**: **Mechanical**: **Types for Selected Step**: **Substructure load**: **Amplitude**: *amplitude*

Load cases in geometrically nonlinear analyses

All substructure loads and boundary conditions are applied in a local system associated with the substructure. Since this local system rotates with the substructure when large motions are present, these loads and boundary conditions will rotate as well. As a consequence, you should be careful when using substructure loads in geometrically nonlinear analyses to ensure that the loading is in the appropriate direction at the usage level. This situation is similar to rotating the substructure via a substructure property definition.

Gravity loading

A distributed load definition can be used to apply gravity loading to a substructure with a user-defined magnitude, scaled by an amplitude definition, and acting in a specifed direction. To enable gravity loading for a substructure, you must request the calculation of the substructure's gravity load vectors during the substructure generation step (see "Gravity loading" in "Defining substructures," Section 10.1.2). In this case gravity loading should not be defined as part of a substructure load case.

Input File Usage: Use the following option to define gravity loading:

*DLOAD, AMPLITUDE=*amplitude*
element set or element number, GRAV, *magnitude, direction*

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Gravity** for the **Types for Selected Step**

Obtaining output of results within a substructure

You can obtain output within substructures used in static, dynamic, eigenfrequency extraction, and steady-state and transient modal dynamic analyses. The recovery of output is not possible for substructures used in response spectrum and random response analyses. Output within a substructure

does not include the displacements, stresses, etc. resulting from the preload deformation of a substructure.

Output within substructures is available in the data (**.dat**) file, in the results (**.fil**) file, and in output database (**.odb**) files. Separate output database files are created for each substructure using the naming convention *inputfile-name_substructure-number***.odb**. If a substructure contains a nested substructure, a file called *inputfile-name_substructure-number_nested-substructure-number***.odb** is created containing the output for the nested substructure. The **abaqus substructurecombine** execution procedure can combine model and results data from two substructure output databases into a single output database. For more information, see "Combining output from substructures," Section 3.2.24.

Recovery of the solution within substructures requires that the information for recovering the data within a substructure be available from the **.sup**, **.sim**, **.prt**, **.stt**, and **.mdl** files.

Output is organized substructure by substructure: you direct Abaqus/Standard to go inside a particular substructure and then request output for that substructure. Results can be recovered within nested multilevel substructures only if the substructure libraries for all substructures in the chain are available.

Substructure output requests are most easily pictured by thinking of substructures as "levels" of detailed modeling. At the global (top) level we have the analysis model (for example, an airplane). Dropping down from this level to the first substructure level, we have the main components of the model defined as substructures (wings, stabilizer, fuselage, etc.). Dropping down to the second substructure level, we have other substructures (flaps, tanks, floors, etc.), which, in turn, may contain third level substructures (spars, stringers, etc.), and so on. To obtain output, you move down and back up through these various levels using substructure paths, similar to the way you navigate a tree structure for file directories. Each substructure path definition consists of entering into a substructure at the next level down or leaving the current substructure and moving up one level in the tree.

At the start of the output requests, Abaqus/Standard is at the global model level. You must always enter and leave a substructure consistently, so that after a set of substructure output requests Abaqus/Standard is left at the global model level. You must return to the global level (outside all substructures) before the end of the step definition.

If you enter and leave in the same substructure path definition, the effect is to leave the substructure and enter another substructure at the same level.

Entering a substructure for output

To enter a particular substructure for output, you identify the substructure by the element number *n* chosen for it in the model. All subsequent output requests are for output within that substructure and must be given in terms of its internal node and element numbers (the node and element numbers used when the substructure was created).

Input File Usage: *SUBSTRUCTURE PATH, ENTER ELEMENT=*n*

Abaqus/CAE Usage: Step module: field output request editor: **Domain**: **Substructure**: click the **Edit** button, and select substructure sets

Leaving a substructure after obtaining output

After you have obtained output for a substructure, you must return to the level of the model of which the substructure forms a part, thus indicating the end of the output requests for variables within that substructure.

Input File Usage: *SUBSTRUCTURE PATH, LEAVE

Abaqus/CAE Usage: Step module: field output request editor: **Domain**: **Substructure**: click the **Edit** button, and select substructure sets

Obtaining output if substructures are nested

You must enter several substructures if substructures are used at multiple levels and output is required several levels down. Nesting of substructures is not supported in Abaqus/CAE.

Example: obtaining output within nested substructures

For example, suppose that a model includes several substructures at two levels. Printed output of stress components is required in some elements within two substructures at the second level, as well as printed output of the displacements at some of the nodes of one of the first-level substructures. (Recall that "first-level" refers to substructures used directly in the analysis model; "second-level" substructures are used as components of first-level substructures.)

The data might be as follows:

```
*SUBSTRUCTURE PATH, ENTER ELEMENT=N
** This option takes us into element number N, which must be a substructure.
*SUBSTRUCTURE PATH, ENTER ELEMENT=M
** We now drop down into element number M of this substructure.
** M is the element number used for this substructure when N was created.
** M must refer to a substructure.*EL PRINT, ELSET=A1
S
** This option requests stress output in element set A1 of this substructure.
** This element set must have been defined during the creation of substructure M.
*SUBSTRUCTURE PATH, LEAVE
** This option takes us back up into first-level substructure N.
*SUBSTRUCTURE PATH, ENTER ELEMENT=P
** This option takes us down into element P, which must again be a substructure in element N.
*EL PRINT, ELSET=A1
S
** This option requests the printing of stress output in element set A1. It is possible that
** this is the same set of elements in the same substructure as was used in the request above
** because substructures M and P may both be copies of the same substructure.
** However, the stresses will presumably be different because they represent the same
** component in different locations in the model.
```

```
*SUBSTRUCTURE PATH, LEAVE
** Back to N.
*SUBSTRUCTURE PATH, LEAVE
** We are now back at the global level.
*SUBSTRUCTURE PATH, ENTER ELEMENT=R
** Enter element R at the global level: this element is the substructure in which we want
** to print the displacements.
*NODE PRINT, NSET=FLANGE
U
** This option prints the displacements at all nodes in node set
** FLANGE of the substructure.
** Again, FLANGE must have been defined when the substructure was
** created.
*SUBSTRUCTURE PATH, LEAVE
** Back to the global level.
```

Interpreting nodal variable output

The nodal displacements within the substructure do not include the displacements resulting from the preload deformation if it exists.

If a substructure is rotated and/or reflected, nodal variables are output relative to the global coordinate system of the analysis. In a geometrically nonlinear analysis, the nodal displacements will include the large motions associated with the translation and rotation of the substructure in addition to the small-strain displacements. If a nodal transformation ("Transformed coordinate systems," Section 2.1.5) has been used, nodal output will be in either the local or the global directions, depending on the nodal output request (see "Output to the data and results files," Section 4.1.2). If a nodal transformation has been used during substructure generation, the transformed directions are rotated with the substructure.

Interpreting element variable output

Element output variables within a substructure do not include the values of the variable resulting from the preload deformation if it exists.

Element variables in continuum elements are output relative to the global coordinate system of the analysis model or in the local (material) coordinate system if one has been used ("Orientations," Section 2.2.5). Element output for structural elements is always given with respect to the element coordinate system used during substructure generation. Integration point coordinates and local material directions (see "Output to the data and results files," Section 4.1.2) are given with respect to the global coordinate system.

Element quantities associated with nonlinear preload response (plastic strains, creep strains, etc.) can be output during a substructure recovery. Since the response in a substructure during its usage is entirely linear, these quantities, which are part of the base state, do not change from the values computed during the preload.

If a substructure was reflected, the element connectivities of continuum elements written to the substructure instance output database are adjusted so as not to violate the Abaqus convention for counterclockwise element numbering.

You cannot directly obtain the element output for the element centroidal values or the element output at the element nodes when you recover results within substructures. This output data can be calculated from the substructure-related data in the output database file using commands in the Abaqus Scripting Interface.

Interpreting results written to the results file

Results within substructures can be written to the results file. Substructure path records are inserted in the results file to indicate the switch into a substructure: all records following such a record belong to the substructure defined on that record until the next substructure path record appears in the file.

Requests for output to the results file will cause Abaqus/Standard to write the definitions of elements and nodes at the global level and within all substructures in the model to the file. As with the results records themselves, these records for nodes and elements within substructures will be preceded and followed by substructure path records to indicate that they belong to that substructure.

Node and element numbers within each substructure are local to that substructure, so that the same node and element numbers may appear in several substructures and in the global level model. In such a case the substructure path records must be used to identify the location of a particular node or element within the model. If you can ensure that node and element numbers are unique throughout the entire model, including all substructures, the substructure path records in the results file can be ignored.

Visualizing substructure results

While Abaqus/CAE does not support substructures directly, you can view substructure results by combining all of the substructure instance output database (**.odb**) files into a single file. See "Combining output from substructures," Section 3.2.24, for details.

You can also load and view each individual substructure instance output database (**.odb**) file separately in Abaqus/CAE.

Substructure library compatibility

A substructure usage analysis can use the substructure libraries generated from the same or any previous maintenance delivery of the same general release. The substructure library is not compatible between general releases.

A substructure usage analysis must be run on a computer that is binary compatible with the computer used to generate the substructure library.

Input file template

The following template can be used to generate a substructure:

```
*HEADING
...
*NODE,NSET=N1
```

Data lines to define the nodes.
…
***NSET,NSET=N3**
Data lines to define the node set members.
…
***ELEMENT, TYPE=CPE8, ELSET=E1**
Data lines to define the elements that make up the substructure.
…
***ELSET,ELSET=E3**
Data lines to define the element set members.
…
***SOLID SECTION, ELSET=E1, MATERIAL=M1**
***MATERIAL, NAME=M1**
***ELASTIC**
30.E6, 0.3
***DENSITY**
0.0007324
***STEP**
***FREQUENCY**
*Data line to specify the number of modes (≥ m). The *FREQUENCY option is required if modes are requested using the *SELECT EIGENMODES option.*
***END STEP**
***STEP**
***STATIC**
…
Options to define a linear or nonlinear static preload.
…
***END STEP**
***STEP**
***SUBSTRUCTURE GENERATE, TYPE=Z101, OVERWRITE, MASS MATRIX=YES, VISCOUS DAMPING MATRIX=YES, STRUCTURAL DAMPING MATRIX=YES, RECOVERY MATRIX=YES, NSET=N3, ELSET=E3**
***RETAINED NODAL DOFS**
Data lines to define the retained degrees of freedom.
***SELECT EIGENMODES, GENERATE**
1, *m*, **1**
***SUBSTRUCTURE LOAD CASE, NAME=LOADS**
***CLOAD**
Data lines to define concentrated loading.
***DLOAD**
Data lines to define distributed loading.
***END STEP**

The following template can be used to define substructure instances:

```
*HEADING
…
*ELEMENT, TYPE=Z101, ELSET=E2
Data line to define the element.
*SUBSTRUCTURE PROPERTY, ELSET=E2
*BOUNDARY
…
*RESTART, WRITE
*STEP
*STATIC
…
*BOUNDARY
…
*SLOAD
E2, LOADS, scale factor
*SUBSTRUCTURE PATH, ENTER ELEMENT=n
*EL PRINT
S,
*NODE PRINT
U,
*SUBSTRUCTURE PATH, LEAVE
*END STEP
*STEP
*DYNAMIC
…
*BOUNDARY
…
*SUBSTRUCTURE PATH, ENTER ELEMENT=n
*EL PRINT
S,
*NODE PRINT
U, V
*SUBSTRUCTURE PATH, LEAVE
*END STEP
```

10.1.2 DEFINING SUBSTRUCTURES

Products: Abaqus/Standard Abaqus/CAE

References

- "Using substructures," Section 10.1.1
- *RETAINED NODAL DOFS
- *SELECT EIGENMODES
- *SUBSTRUCTURE COPY
- *SUBSTRUCTURE DELETE
- *SUBSTRUCTURE DIRECTORY
- *SUBSTRUCTURE GENERATE
- *SUBSTRUCTURE LOAD CASE
- *SUBSTRUCTURE MATRIX OUTPUT
- *MATRIX CHECK
- *FLEXIBLE BODY

Overview

This section describes how individual substructures are defined. See "Using substructures," Section 10.1.1, for information regarding how they are used in a model.

Substructures are defined using the substructure generation procedure. The substructure creation and usage cannot be included in the same analysis. Multiple substructures can be generated in an analysis. Any substructure can consist of one or more other substructures; if this is the case, the nested-level substructures must be defined first. The substructure library is not organized in terms of part instances; therefore, substructures cannot be generated from models that have an assembly defined. None of the substructure options are supported in models that have an assembly defined.

To define a typical substructure generation step, do the following:

- Invoke the substructure generation procedure.
- Define the nodes and degrees of freedom that are to be retained as external degrees of freedom when the substructure is used.
- Optionally, retain extra dynamic modes to improve the dynamic behavior of the substructure during usage.
- Optionally, specify substructure load cases.
- Optionally, write the recovery matrix, substructure's stiffness matrix, mass matrix, and/or load case vectors to a file.

Generating a substructure

When you generate a substructure, you specify an identifier that will be assigned to this substructure in a substructure library. The identifier must begin with the letter Z followed by a number that cannot exceed 9999.

Substructure identifiers must be unique within a library. If a substructure with this same identifier already exists in the library, the analysis will terminate with an error message unless you have specified that the existing substructure should be overwritten, as described below.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: *n*

Substructure database

A substructure database is the set of files that describe the geometry of a substructure, and Abaqus writes all substructure data to the substructure database during the analysis. The substructure database can include files with the following extensions: **.sup**, **.sim**, **.prt**, **.mdl**, and **.stt**; the **.sup** file is called the substructure library. By default, substructure data are written to a substructure database named *jobname*, and the substructure files are named *jobname***.sup**, *jobname_Zn***.sim**, *jobname_Zn***.prt**, *jobname_Zn***.mdl**, and *jobname_Zn***.stt**. Files with the extensions **.sup** and **.sim** are generated for all substructures. Files with the extensions **.prt**, **.mdl**, and **.stt** are generated only if the solution within the substructure can be fully or partially recovered. Several substructures can share a substructure library file, but other files are individual for each substructure. It is strongly recommended that the substructure library name be different for different substructures.

You can choose to write the data to a user-specified substructure database.

If you specify the substructure library name, the files will be named *library_name_Zn***.sim**, *library_name_Zn***.prt**, *library_name_Zn***.mdl**, and *library_name_Zn***.stt**.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*, LIBRARY=*library_name*

Abaqus/CAE Usage: Definition of substructure libraries is not supported in Abaqus/CAE.

Overwriting the substructure data in a library

If a substructure generation analysis is rerun using the same *jobname* without deleting the substructure library and one substructure or more will be regenerated, you must specify that the existing substructures can be overwritten. This requirement also holds true if the *jobname* is different for the second analysis but the same *library_name* is specified.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*, LIBRARY=*library_name*, OVERWRITE

Abaqus/CAE Usage: Definition of substructure libraries is not supported in Abaqus/CAE.

Recovery within a substructure

By default, the solution at any degree of freedom in the substructure can be recovered. Abaqus must have access to the substructure's **.mdl**, **.prt**, and **.stt** files to perform a full recovery. These files all reside in the substructure database.

You can specify that a recovery of element or nodal information will not be required within this substructure. This reduces the size of the substructure database significantly for a large substructure because the information that is needed to recover eliminated variables is not stored. However, this information cannot be recreated at a later time except by regenerating the entire substructure with recovery enabled.

Input File Usage: Use the following option to enable recovery for a substructure:

*SUBSTRUCTURE GENERATE, TYPE=Z*n*, RECOVERY MATRIX=YES (default)

Use the following option to disable recovery for a substructure:

*SUBSTRUCTURE GENERATE, TYPE=Z*n*, RECOVERY MATRIX=NO

Abaqus/CAE Usage: Use the following option to enable recovery for a substructure:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Whole model**

Use the following option to disable recovery for a substructure:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle off **Evaluate recovery matrix for**

Using the selective recovery method

If results recovery is desired only at a subset of the internal degrees of freedom, disk usage can be reduced substantially by using the selective recovery method. To enable selective recovery, the region where recovery is desired can be specified directly.

Input File Usage: Use the following option to define the node set for selective recovery:

*SUBSTRUCTURE GENERATE, RECOVERY MATRIX=YES, NSET=*Node set name*

Use the following option to define the element set for selective recovery:

*SUBSTRUCTURE GENERATE, RECOVERY MATRIX=YES, ELSET=*Element set name*

Abaqus/CAE Usage: Use the following option to define the node set for selective recovery:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Region**: *Node set name*

Use the following option to define the element set for selective recovery:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Region**: *Element set name*

Evaluating frequency-dependent material properties

When frequency-dependent material properties are specified, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in substructure generation. If you do not choose the frequency, Abaqus/Standard evaluates the stiffness at zero frequency and does not consider the stiffness contributions from frequency-domain viscoelasticity. If you do specify a frequency, only the real part of the stiffness contributions from frequency-domain viscoelasticity is considered.

Input File Usage: *SUBSTRUCTURE GENERATE, PROPERTY EVALUATION=*frequency*

Abaqus/CAE Usage: Step module: Step editor: **Substructure generate**: **Options** tabbed page: toggle on **Evaluate frequency-dependent properties at frequency**: *frequency*

Defining the retained degrees of freedom

The degrees of freedom at a node can be divided into retained degrees of freedom (for use at the usage level of the substructure) and eliminated degrees of freedom (internal to the substructure). Abaqus/Standard allows any of the degrees of freedom at any of the nodes of a substructure to be retained with one exception: if an acoustic-structural substructure is generated, based on coupled or uncoupled modes, only structural degrees of freedom can be retained. You must make sure that the choice of retained degrees of freedom is reasonable so that the substructure can be connected correctly to the rest of the model.

Any degrees of freedom where kinematic constraints may have to be respecified during usage of the substructure should be kept as retained degrees of freedom.

If any degrees of freedom of nodes used to define distributing coupling elements are retained, the degrees of freedom of an internal node associated with the Lagrange multipliers are added automatically to the list of the retained degrees of freedom of the substructure.

To define the retained degrees of freedom, specify the node number or node set label and, optionally, the first and the last degree of freedom to be retained.

By default, the nodes associated with the retained degrees of freedom will be sorted into ascending numerical order.

Input File Usage: *RETAINED NODAL DOFS

Abaqus/CAE Usage: Load module: boundary condition editor: **Category**: **Mechanical**: **Types for Selected Step**: **Retained nodal dofs**

Preventing the degrees of freedom from being sorted

You can prevent the degrees of freedom from being sorted. The ordering of the nodes when using a substructure is then the same as the ordering used when specifying the retained nodes.

Input File Usage: *RETAINED NODAL DOFS, SORTED=NO

Abaqus/CAE Usage: You cannot prevent retained nodes from being sorted in Abaqus/CAE.

Retaining degrees of freedom when the substructure is intended for geometrically nonlinear analysis at the usage level

When the substructure is intended for use in geometrically nonlinear analyses, it is recommended to retain all translational and/or all rotational degrees of freedom from a particular node. Even in the case when only a single translational/rotational degree of freedom of a particular node is deemed as needed at the usage level, you should retain all translational/rotational degrees of freedom associated with that node. Otherwise, as the substructure rotates during a geometrically nonlinear analysis, local numerical instabilities (negative eigenvalues) may occur since the rotated substructure may have no stiffness in particular degrees of freedom.

You must choose an appropriate number of nodes that will allow for the computation of an equivalent rigid body motion of the substructure. In two-dimensional or axisymmetric analyses, retaining two nodes with all translational degrees of freedom or one node with all translational and rotational degrees of freedom is sufficient to compute an equivalent rigid body motion of the substructure at the usage level. In three-dimensional analysis, three non-colinear nodes with all translational degrees of freedom retained or one node with all translations and rotations are needed. If the retained nodes are colinear or fewer than three nodes are retained, you must retain at least one node with all rotational degrees of freedom. When Abaqus/Standard cannot compute an equivalent rigid body motion for the substructure during the analysis at the usage level because the number of retained degrees of freedom is not appropriate, a warning message is issued and any geometrically nonlinear effects associated with the substructure are ignored.

Defining kinematic constraints

Kinematic constraints are defined as described in "Kinematic constraints: overview," Section 35.1.1. The following rules apply:

- All kinematic boundary conditions associated with degrees of freedom that are not retained must be specified when the substructure is generated. The conditions are built into the substructure and remain imposed any time that it is used. Once the substructure is generated, kinematic constraints on internal variables cannot be respecified; they can be modified or removed only by erasing and recreating the substructure in the library. The magnitude of a prescribed boundary condition applied to an internal degree of freedom can be associated with a substructure load case and can be changed at the usage level. The restraint itself is built into the substructure and cannot be removed by omitting a reference to the load case.
- During substructure generation, multi-point constraints in which some of the substructure's retained degrees of freedom are eliminated in favor of internal degrees of freedom must be avoided. If it

is desirable to retain certain degrees of freedom that are eliminated by the multi-point constraints, you must reassign all of the variables appearing in the multi-point constraints as retained degrees of freedom and impose the constraints at the usage level.

Defining the generalized degrees of freedom

An effective technique for modeling the dynamic behavior of a substructure is to augment the response within the substructure by including some generalized degrees of freedom associated with the dynamic modes. You can select the modes to retain, which must be calculated in a previous frequency extraction step ("Natural frequency extraction," Section 6.3.5). For some cases of the substructure generation, the dynamic modes have to be fully recovered; if they were computed with the AMS eigensolver and only partially recovered, an error message is issued in such cases. For example, if a substructure includes the substructure load cases or structural-acoustic coupling (or it will be used for flexible body generation) the eigenmodes have to be fully recovered. The modes will include eigenmodes and, if activated in the eigenfrequency extraction step, residual modes. If all retained degrees of freedom of the substructure are constrained in the frequency extraction step, this technique is commonly referred to as the Craig-Bampton method. If all retained degrees of freedom of the substructure are not constrained in the frequency extraction step, this technique is commonly referred to as the Craig-Chang method. The substructure dynamic modes in the Craig-Bampton method are commonly referred to as the fixed-interface modes, and the substructure dynamic modes in the Craig-Chang method are commonly referred to as the free-interface modes. If some retained degrees of freedom of the substructure are constrained and other retained degrees of freedom are not constrained in the frequency extraction step, the dynamic modes are called mixed-interface modes. If the free-interface or mixed-interface dynamic modes are selected, the substructure generation time can increase substantially compared to the case when the same number of fixed-interface dynamic modes is used. Abaqus issues a warning message in this case. However, better solution accuracy can sometimes be achieved with a significantly smaller number of free- or mixed-interface dynamic modes than by using fixed-interface modes.

A sufficient number of the dynamic modes should be selected to provide adequate dynamic representation of the substructure. You should examine loading frequencies and frequency content of the structure to determine this range. Specify a shift point and/or a cutoff frequency in the eigenfrequency extraction step definition to obtain modes in the desired frequency range only. Inclusion of generalized degrees of freedom adds the cost of the frequency extraction to the substructure generation step but greatly improves the accuracy of the solution if the substructure is used in a subsequent dynamic ("Implicit dynamic analysis using direct integration," Section 6.3.2), steady-state dynamic ("Direct-solution steady-state dynamic analysis," Section 6.3.4), or frequency extraction ("Natural frequency extraction," Section 6.3.5) analysis.

In the case of the displacement normalization of the eigenvectors in a frequency extraction analysis, a substructure must have at least one physical degree of freedom active on the usage level; otherwise, the modes cannot be normalized properly. See "Substructuring and substructure analysis," Section 2.14.1 of the Abaqus Theory Guide, for additional details.

The retained eigenmodes must be selected when an acoustic-structural substructure is generated.

The effect of acoustic-structural coupling can be included in the retained eigenmodes during the natural frequency extraction procedure. To calculate the coupled structural-acoustic eigenmodes, use a

frequency extraction analysis with the default Lanczos eigensolver and include the effect of acoustic-structural coupling during the natural frequency extraction procedure ("Natural frequency extraction," Section 6.3.5).

Abaqus can also use uncoupled eigenmodes, generated from either SIM-based Lanczos or AMS eigensolver, to generate a coupled acoustic-structural substructure. In this case the effect of acoustic-structural coupling is included during the substructure generation. Both structural and acoustic eigenmodes have to be retained for the substructure generation, and the selection of the acoustic zero-frequency modes, if such modes are present, is required to get an accurate substructure.

Selecting the modes to be used in a substructure generation analysis by their mode numbers

You can directly specify the eigenmodes to be used in a substructure generation analysis by their mode numbers.

Input File Usage: ∗SELECT EIGENMODES
eigenmode 1, *eigenmode 2*, *etc.*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by mode range, with each row in the data table specifying a single mode number. The starting mode number and ending mode number in each row should be equal, and the increment value should be zero.

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Mode range**:
Start Mode: *eigenmode 1*: **End Mode**: *eigenmode 1*: **Increment**: 0
Start Mode: *eigenmode 2*: **End Mode**: *eigenmode 2*: **Increment**: 0
etc.

Generating a list of the eigenmodes by mode range

Instead of listing all the retained eigenmode numbers, you can generate the list of eigenmodes.

Input File Usage: Use the following option to generate the list of eigenmodes by mode range, with each data line specifying the start mode number, the end mode number, and the increment in mode numbers between these two values:

∗SELECT EIGENMODES, GENERATE
first mode number, *last mode number*, *increment*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by mode range, with each row in the data table specifying the start mode number, the end mode number, and the increment in mode numbers between these two values:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Mode range**: **Start Mode**: *first mode number*: **End Mode**: *last mode number*: **Increment**: *increment*

Generating a list of the eigenmodes by frequency range

You can select all the modes from the specified frequency range including frequency boundaries.

Input File Usage: Use the following option to generate the list of eigenmodes by frequency range, with each data line specifying the lower boundary of the frequency range and the upper boundary of the frequency range:

∗SELECT EIGENMODES, DEFINITION=FREQUENCY RANGE
lower boundary of the frequency range, *upper boundary of the frequency range*

Abaqus/CAE Usage: Use the following option to generate the list of eigenmodes by frequency range, with each row in the data table specifying the lower boundary of the frequency range and the upper boundary of the frequency range:

Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Specify retained eigenmodes by**: **Frequency range**: **Lower Frequency**: *lower boundary of the frequency range*: **Upper Frequency**: *upper boundary of the frequency range*

Preloading a substructure

Substructures can be used in models that exhibit nonlinear response (associated with standard Abaqus elements or with contact definitions), but the response within a substructure assumes linear small deformations. However, a substructure's response may be a linear perturbation about a predeformed (possibly rotating and translating) base state, defined on the basis of nonlinear response within the substructure during its preload history.

When the substructure is intended for use in geometrically nonlinear analyses, the substructure preloading should be limited to loads that generate self-equilibrating stresses only (such as thermal stresses or interference fits). In most cases, preload stresses are not self-equilibrating (such as stresses from specified boundary conditions or applied loads). If non-self-equilibrating prestress exists and the substructure undergoes a rigid body motion at the usage level, additional stress is generated in the substructure. Such usage level stresses are non-physical and will lead to convergence problems and results that are difficult to interpret. Therefore, you should use extreme care when preloading a substructure intended for use in geometrically nonlinear analyses.

This preloading concept allows such effects as stress stiffening to be included in a substructure. Preloading is a part of the state of the substructure: the preload is self-equilibrating and so does not generate a load vector when the substructure is used. Any loading of the substructure during its use in a model is in addition to the preload.

It is important to distinguish the difference between a preload and a load case. Both are allowed during a substructure generation analysis, but only the preloads are actually applied to the substructure during generation. Load cases, defined during substructure generation, can only be applied at the usage level (see "Applying loads to a substructure" in "Using substructures," Section 10.1.1). Load cases are discussed in more detail later.

Computation of the total response of a variable

Any recovered response variable within a substructure (such as stress or displacement) is defined to be a perturbation (with some exceptions for geometrically nonlinear analyses) from the preloaded base state. For geometrically nonlinear analyses, the displacement output includes both the equivalent rigid body rotation and translation associated with the substructure and the strain-inducing small-displacement perturbation. If the total response of a variable is desired, it can be computed by adding the perturbation result to the final result computed during the substructure preload.

Computation of the tangent stiffness of a preloaded substructure

The rules for calculating the stiffness matrix of a preloaded substructure are the same as those for a static linear perturbation step. See "General and linear perturbation procedures," Section 6.1.3, for a detailed description of the rules.

Defining a preloading history

Specify the loading history that defines the preload state for a substructure.

Input File Usage: Use the following options:

```
*STEP
Options to define the preloading history.
*END STEP
Any number of steps can be defined.
*STEP
*SUBSTRUCTURE GENERATE
Options to define the substructure.
*END STEP
```

Abaqus/CAE Usage: The **Substructure generation** step must be defined after the preloading steps in an Abaqus/CAE analysis.

Prescribing boundary conditions at retained degrees of freedom during preloading steps

During substructure preloading, boundary conditions can be prescribed at retained degrees of freedom. When the preloaded substructure is subsequently created in a substructure generation step, you must release all the retained degrees of freedom (see "Removing boundary conditions" in "Boundary conditions in Abaqus/CFD," Section 34.3.2). An error message will be issued if some of the retained degrees of freedom are not released. The reaction forces at the released degrees of freedom become concentrated loads that are in equilibrium with the stresses within the substructure. These concentrated loads cannot be removed without changing the preload.

The preloaded substructure is, thus, in equilibrium. If the preload in a substructure must effectively apply loading to other parts of the structure, a substructure load case corresponding to the loads applied in the preload history must be created.

The technique is demonstrated in "Analysis of a rotating fan using substructures and cyclic symmetry," Section 2.2.1 of the Abaqus Example Problems Guide.

Generating a reduced mass matrix for a substructure

You can generate a reduced mass matrix for a substructure.

A reduced mass matrix is calculated by projecting the global mass matrix to the subspace of the substructure modes. This technique is known as Guyan reduction if only the static modes associated with the nodal retained degrees of freedom are used. Using only the static modes may not be sufficient to define the dynamic response of the substructure accurately. Additional dynamic modes must be used to improve the response inside the substructure.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*, MASS MATRIX=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced mass matrix**

Generating a reduced viscous damping matrix for a substructure

You can generate a reduced viscous damping matrix for a substructure.

The reduced viscous damping matrix is calculated in a manner similar to that used for the reduced mass matrix.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*, VISCOUS DAMPING MATRIX=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced viscous damping matrix**

Generating a reduced structural damping matrix for a substructure

You can generate a reduced structural damping matrix for a substructure.

The reduced structural damping matrix is calculated in a manner similar to that used for the reduced mass matrix.

Input File Usage: *SUBSTRUCTURE GENERATE, TYPE=Z*n*, STRUCTURAL DAMPING MATRIX=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute reduced structural damping matrix**

Generating substructures with unsymmetric damping matrices

When a coupled acoustic-structural substructure, generated from coupled or uncoupled modes, is generated from a model with damping specified on the acoustic domain, the substructure damping matrices are unsymmetric. The substructure viscous damping matrix will be unsymmetric if a substructure is generated from the rolling tire.

Abaqus does not automatically generate an unsymmetric substructure in these cases. You must explicitly select the unsymmetric solver (see "Defining an analysis," Section 6.1.2) for the substructure generation step to obtain correct substructure damping matrices with unsymmetric contributions.

Defining substructure load cases for subsequent loading in an analysis

The load cases defined during the generation of a substructure and activated at the usage level are the equivalent of the elemental loading types available for the regular elements in Abaqus. They can be made up of any combination of loadings (distributed loads, concentrated nodal loads, thermal expansion, and load cases defined for any substructures that may be used as part of the definition of this substructure).

The load cases are needed so that, when the substructure is subsequently used in a model, the consistent loads on the retained degrees of freedom need be scaled only by the appropriate magnitudes of the particular loads applied: it is not necessary to go inside the substructure and repeat the basic element calculations to distribute the loads.

Each such load case can be applied when the substructure is used by associating it with an amplitude/time curve and a magnitude ("Amplitude curves," Section 34.1.2). When a substructure is used, the substructure load case loadings that were created when the substructure was generated are the only loads that can be used in that substructure. Except for gravity loading, when using the substructure, you cannot apply distributed loads, temperature loads, etc. to the elements that make up any substructure. These loads must be built into the substructure during its creation.

You can define multiple substructure load cases during the substructure generation to define different loadings for the substructure. Each load case is assigned a name that will be used when the load case is applied on the usage level.

You can use any combination of concentrated load, distributed load, substructure load, and temperature fields ("Concentrated loads," Section 34.4.2; and "Distributed loads," Section 34.4.3) to define each load case.

You assign each basic loading a reference magnitude, which will then be scaled by the actual magnitude specified when the substructure load is applied. The reference magnitude assigned to each basic loading must be defined as the change in load or boundary condition from the base state, not the total of the base state plus the perturbation value. Initial conditions applied within the substructure generation are not included as part of a load case definition.

For temperature loads, the load vector for the substructure load case will contain only the contributions due to thermal expansion. If temperature-dependent properties are present, they are evaluated at the temperatures specified in the preloaded state. Consequently, to take into account nonzero initial temperature fields prescribed as initial conditions ("Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1), it is necessary to preload the structure before creating the substructure. When using temperature loading in a substructure load case, the data cannot be read from a results file. The temperatures specified must be defined as the change in the temperatures from the base state.

Abaqus/Standard currently has a limitation when a substructure load case definition includes acoustic loading during a substructure generation procedure in which retained modes are specified: the contribution of the singular (constant pressure) acoustic modes ("Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1) is not taken into account in the generated load case.

Since the contribution of this mode is significant for low frequency response, the generated load case will inadequately represent the specified acoustic load in these cases. If there are no singular acoustic regions in the coupled acoustic-structure substructure, the acoustic loads are represented accurately.

It is important to distinguish the difference between a load case and a preload. Both are defined during substructure generation, but only the preloads are actually applied to the substructure on the generation level; load cases, defined on the generation level, can only be applied on the usage level, and they act on a preloaded base state if one has been specified. (Preloads were discussed earlier.)

In general analysis steps and perturbation steps substructure loads are treated in the same way as other loads, such as concentrated loads and distributed loads ("Concentrated loads," Section 34.4.2, and "Distributed loads," Section 34.4.3). For example, if a general analysis step is followed by another general analysis step, the substructure loads will be retained in the second step with their magnitude equal to that at the end of the previous general analysis step, unless the substructure load is modified or removed. In a linear perturbation step the substructure load represents an incremental load.

If a substructure load is used to apply Coriolis loading in a direct-solution steady-state dynamic analysis, the unsymmetric load stiffness contribution is not taken into account.

Input File Usage: Use the following options:

∗SUBSTRUCTURE LOAD CASE, NAME=*name*
∗CLOAD and/or
∗DLOAD and/or
∗DSLOAD and/or
∗TEMPERATURE

The load case defined via the ∗SUBSTRUCTURE LOAD CASE option ends when an option other than ∗CLOAD, ∗DLOAD, ∗DSLOAD, or ∗TEMPERATURE is encountered. The load definitions can be specified in any order.

Abaqus/CAE Usage: Use the following option to define a substructure load case and the loads included in it:

Load module: **Create Load Case**: click the **Add** button: select loads

Defining boundary conditions

All boundary conditions to be built into the substructure matrices must be specified using a boundary condition definition. These cannot be part of a substructure load case specification. Once a kinematic boundary condition is specified on a particular nodal degree of freedom, it is built into the substructure matrices, is in effect for all load cases, and cannot be removed (or redefined at the usage level). The boundary conditions specified as part of the preloading history are built into the substructure matrices.

If there is any doubt whether a restraint is permanent or not, it is better to make the degree of freedom a retained degree of freedom and not specify any restraint in the substructure definition. The restraint can then be included as needed in each analysis step.

Load cases when the substructure is used in geometrically nonlinear analyses

All loads included in a substructure load case at the generation level and applied as a substructure load at the usage level are applied in a local system associated with the substructure. Since this system rotates with the substructure when large motions are present, these loads will rotate as well. As a consequence, you should be careful when using substructure load cases in geometrically nonlinear analyses to ensure that the loading is in the appropriate direction at the usage level. This situation is similar to rotating the substructure using a substructure property definition.

Gravity loading

To apply gravity loading, density must be defined for at least some of the elements included in the substructure. A gravity load can be applied to a substructure in two different ways with two different interpretations. If a distributed load definition is used as a part of a substructure load case during substructure generation (as described in "Defining substructure load cases for subsequent loading in an analysis" above), the gravity loading becomes part of the substructure load case and, hence, rotates to follow the substructure's local system during usage (the local system may rotate by rotating the substructure via a substructure property definition or due to geometrically nonlinear response).

To define gravity loading that acts in a fixed global direction during usage, you can request that the substructure's gravity load vectors be calculated during substructure generation. In this case gravity loading should not be defined as part of a substructure load case. When the gravity load vectors are calculated, Abaqus/Standard generates a gravity load vector for each global direction (three for three-dimensional analyses and two for two-dimensional/axisymmetric analyses). At the usage level, a distributed load definition can be used (see "Gravity loading" in "Using substructures," Section 10.1.1) to specify gravity loading on the substructure that acts in a fixed global direction with the specified magnitude.

Input File Usage: Use the following option to calculate the substructure's gravity load vectors during substructure generation:

*SUBSTRUCTURE GENERATE, GRAVITY LOAD=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Options** tabbed page: toggle on **Compute gravity load vectors**

Substructure eigenvalue problem

We define the substructure eigenvalue problem as the generalized eigenvalue problem for reduced substructure stiffness and mass matrices. The reduced stiffness matrix is always generated for Abaqus substructures. If a generated substructure has the reduced mass matrix, the substructure eigenvalue problem can be solved and the substructure eigenmodes can be extracted. The substructure eigenfrequencies provide useful information about the substructure dynamic properties. The substructure eigenmodes can be used to define the substructure modal damping at the substructure usage stage, and they are required for the flexible body generation. By default, the substructure eigenvalue problem is solved when it is possible (when the reduced mass matrix is available). If generation of the reduced mass matrix is not requested but generation of a flexible body from a substructure is performed,

we solve the substructure eigenvalue problem; but instead of the conventional reduced mass matrix, we use a projection of the lumped mass matrix on the substructure modal subspace. The lumped mass matrix is created from the global mass matrix of the finite element model by the commonly used heuristic algorithm. If the substructure eigenvalue problem is solved, the obtained substructure eigenvalues and eigenfrequences are printed in the data (**.dat**) file. If desired, you can disable the solve of the substructure eigenvalue problem.

Input File Usage: Use the following option to enable solving of the substructure eigenvalue problem:

*SUBSTRUCTURE GENERATE, EIGENPROBLEM=YES (default)

Use the following option to disable solving of the substructure eigenvalue problem:

*SUBSTRUCTURE GENERATE, EIGENPROBLEM=NO

Abaqus/CAE Usage: Disabling the substructure eigenvalue problem solution is not supported in Abaqus/CAE. When it is possible, the substructure eigenvalue problem is solved.

Checking generated substructure matrices

In a substructure generation analysis, you can check the quality of the generated substructure stiffness and mass matrices. The matrix check generates six "artificial" rigid body modes and projects the substructure matrices onto the rigid body modal subspace. It is expected that the projected 6×6 stiffness matrix (also known as the rigid body energy matrix) is close to zero in the absence of the boundary conditions and constraints. The total inertia statistics for the model are extracted from the projected 6×6 rigid body mass matrix. You can specify the center of rotation for creating the artificial rotational rigid body modes and calculating the global inertia tensor. If the matrix quality check is requested, the check results are printed in the data (**.dat**) file.

Input File Usage: Use the following option to request the matrix check:

*MATRIX CHECK

Use the following option to specify the center of rotation of the coordinate frame to be used for the matrix check:

*MATRIX CHECK, REFERENCE NODE=*node_label*

Abaqus/CAE Usage: Checking generated substructure matrices is not supported in Abaqus/CAE.

Generating a flexible body

Abaqus/Standard can generate a flexible body from a substructure. The generated flexible body can be used in flexible body dynamic simulations using external solvers. Abaqus/Standard supports generation of several flexible body types that can be used by external flexible body dynamics solvers. The generated flexible body entities are stored in the substructure **.sim** file and can be converted to conventional flexible body representations by postprocessing programs.

Input File Usage: Use the following option to generate a flexible body:

*FLEXIBLE BODY, TYPE=*type*

Abaqus/CAE Usage: Generating a flexible body from a substructure is not supported in Abaqus/CAE.

Writing the recovery matrix, reduced stiffness matrix, mass matrix, load case vectors, and gravity vectors to a file

You can write a substructure's recovery matrix, reduced stiffness matrix, mass matrix, and load case vectors to a file. This output is useful when the substructure is to be used in another program.

The output records can be written either to the Abaqus/Standard results file, to a user-defined file, or to the output database file (see below). In each case you must specify which output to write out: the mass matrix, the recovery matrix, the load case vectors, the stiffness matrix, and/or the gravity load vectors. By default, no output will be generated.

Repeat the substructure matrix output request in the substructure generation file of each substructure for which the substructure matrix output is required.

If substructure load case vector output is requested for a preloaded substructure, the output will contain a record with a load case number that is equal to zero. This load vector contains the forces that were necessary to equilibrate any stresses that were generated during the previous steps.

Input File Usage: *SUBSTRUCTURE MATRIX OUTPUT, MASS=YES, RECOVERY MATRIX=YES, SLOAD=YES, STIFFNESS=YES, GRAVITY LOAD=YES

Abaqus/CAE Usage: Step module: **Create Step**: **Linear perturbation**: **Substructure generation**: **Basic** tabbed page: toggle on **Evaluate recovery matrix for**: select **Whole model** or **Region**

Writing the records to the Abaqus/Standard results file

By default, the requested matrices are written to the Abaqus/Standard results file corresponding to the substructure generation input file name. The record formats for the results file are described in "Results file output format," Section 5.1.2. The file can be written in either binary or ASCII format ("Output," Section 4.1.1).

Input File Usage: *SUBSTRUCTURE MATRIX OUTPUT, OUTPUT FILE=RESULTS FILE

Abaqus/CAE Usage: Abaqus/CAE automatically writes the requested matrices to the Abaqus results file when you run an analysis with a **Substructure generation** step.

Writing the records to a user-defined file

You can specify the name of the file (without an extension) to which the data will be written. The records are written to be compatible with a linear user-defined element. The record formats are described in "User-defined elements," Section 32.17.1. An **`.mtx`** extension will be added to the file name specified.

Input File Usage: *SUBSTRUCTURE MATRIX OUTPUT,
OUTPUT FILE=USER DEFINED, FILE NAME=*file_name*

Abaqus/CAE Usage: Job module: job editor: **Name**

Managing substructures inside libraries

Substructures are stored in a collection of libraries. Housekeeping functions are provided to help maintain extensive libraries; for example, substructures can be deleted from a library or moved to a different library.

Once a substructure library has been generated, the disk files can be made read-only to protect the library from accidental deletion or modification. A substructure library must be write-accessible during a substructure's generation and when substructures are added or deleted from a library using the substructure housekeeping functions.

When multiple analyses are used to generate a substructure library, these analyses must be run one after another; they cannot be run simultaneously. Abaqus may not be able to provide any indication that the substructure library being written may already be in use by another Abaqus analysis. If several analyses write to the same library simultaneously, the library may get corrupted. If this occurs and the library is used in a subsequent analysis, the result may be a large preprocessor memory demand.

Input File Usage: Use any of the following options (described in detail below) to perform housekeeping functions on substructure libraries:

*SUBSTRUCTURE COPY
*SUBSTRUCTURE DELETE
*SUBSTRUCTURE DIRECTORY

The housekeeping options can appear anywhere within the model portion of the input file ("Defining a model in Abaqus," Section 1.3.1). An input file can consist of merely the *HEADING option and one or more of the housekeeping options. In this case the files and substructures to which the housekeeping options refer must exist at the start of the analysis.

Abaqus/CAE Usage: Substructure libraries are not supported in Abaqus/CAE.

Listing the substructures stored in a substructure library

You can obtain a summary of information about the substructures stored in a substructure library. If necessary, you can identify a nondefault name for the library (the default name is *jobname*).

Input File Usage: *SUBSTRUCTURE DIRECTORY, LIBRARY=*substructure_library_name*

Abaqus/CAE Usage: Substructure libraries are not supported in Abaqus/CAE.

Removing a substructure from a substructure library

You can remove a specified substructure from a substructure library. If necessary, you can identify the name of the library.

Input File Usage: *SUBSTRUCTURE DELETE, TYPE=Z*n*,
LIBRARY=*substructure_library_name*

Abaqus/CAE Usage: Substructure libraries are not supported in Abaqus/CAE.

Copying or moving a substructure definition

You can copy a substructure definition from one library to another or from one substructure to another within the same library. You must identify the substructure being copied and assign a name to the substructure being created.

When copying substructures from library to library, you can identify the name of the library containing the substructure being copied. Similarly, you can identify the name of the new library to which the substructure will be copied. This new library need not exist prior to the substructure being copied; it will be created in this case.

If the original substructure is to be deleted, you can follow the copy with a delete (see above).

Input File Usage: *SUBSTRUCTURE COPY, OLD TYPE=Z*n*, NEW TYPE=Z*n*,
OLD LIBRARY=*substructure_library_name*,
NEW LIBRARY=*substructure_library_name*

Abaqus/CAE Usage: Substructure libraries are not supported in Abaqus/CAE.

Renaming substructure libraries

Once a substructure library has been generated, the disk file should not be renamed manually. To rename a substructure library, copy the existing substructures to a new library. The new library need not exist prior to the first substructure being copied. You can then delete the original disk file manually if you do not need it anymore.

10.2 Submodeling

- “Submodeling: overview,” Section 10.2.1
- “Node-based submodeling,” Section 10.2.2
- “Surface-based submodeling,” Section 10.2.3

10.2.1 SUBMODELING: OVERVIEW

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Node-based submodeling," Section 10.2.2
- "Surface-based submodeling," Section 10.2.3
- *SUBMODEL
- Chapter 38, "Submodeling," of the Abaqus/CAE User's Guide

Overview

The submodeling technique:

- is used to study a local part of a model with a refined mesh based on interpolation of the solution from an initial (undeformed), relatively coarse, global model;
- is most useful when it is necessary to obtain an accurate, detailed solution in a local region and the detailed modeling of that local region has negligible effect on the overall solution;
- can be used to drive a local part of the model by nodal results, such as displacements (see "Node-based submodeling," later in this section), or by the element stress results (see "Surface-based submodeling," later in this section) from the global mesh;
- can be used to analyze an acoustic model driven by displacements from a structural, global model when the acoustic fluid has negligible effect on the structural solution;
- can be used for the analysis of a structure driven by acoustic pressures from an acoustic or coupled acoustic-structural, global model;
- can use a combination of Abaqus/Explicit and Abaqus/Standard procedures;
- can use a combination of linear and nonlinear procedures; and
- cannot be used in an import analysis.

Terminology

The model whose solution is interpolated onto the relevant parts of the boundary of the submodel is referred to as the "global" model (even though it may itself be a submodel of a larger "global" model). Driven variables are defined as those variables in the submodel that are constrained to match results from the global model. Driven variables can be degrees of freedom at nodes in the node-based technique, or they can be components of stress tensor at the integration points of element faces in the surface-based technique.

Submodeling techniques

Submodeling can be applied quite generally in Abaqus. The material response defined for the submodel may be different from that defined for the global model. Both the global model and the submodel can have nonlinear response. See "Shell-to-solid submodeling and shell-to-solid coupling of a pipe joint," Section 1.1.10 of the Abaqus Example Problems Guide, for an example application of the submodeling technique.

Vehicle-occupant/pedestrian interaction simulations are an example where the submodeling technique can be used efficiently. Crash safety simulation generally includes interaction between a vehicle and its occupant or a vehicle and a pedestrian. In some cases the influence of the human on the structural response of the vehicle is so small as to be negligible. In these cases, the global analysis of the vehicle is performed without the human or with a simple representation of the human, and the part of the vehicle surrounding the human is then used via the submodeling technique to study the detailed interaction with a number of human models.

Submodeling is classified first according to which of two basic techniques is used. The most common, and more general technique, is node-based submodeling, which uses a nodal results field (including displacement, temperature, or pressure degrees of freedom) to interpolate global model results onto the submodel nodes. The alternative surface-based technique uses the stress field to interpolate global model results onto the submodel integration points on the driven element-based surface facets.

You can choose either the node-based or surface-based technique or a combination of the two in your submodel. The following factors should be considered in deciding on the technique to be used:

- Whether you are performing solid-to-solid submodeling in a general static analysis in Abaqus/Standard:
  - Surface-based submodeling is available only for solid models and static analyses.
  - For all other procedures use the node-based technique.
- Whether the global model and submodel differ significantly in their average stiffness in the region of the submodel:
  - When the stiffness of the models is comparable, node-based submodeling of displacements will provide comparable results to the surface-based technique with a lesser likelihood of numerical issues associated with rigid-body modes.
  - When the stiffness of the models differs and the global model is exposed primarily to a load-controlled environment, the surface-based technique will generally provide more accurate stress results. Stiffness differences may arise due to additional detail in the submodel, such as explicit modeling of a fillet or a hole. In other cases stiffness changes may result from minor geometry changes for which a reanalysis of the global model is not warranted.
- Whether your model is subjected to large deformations or rotations:
  - Node-based submodeling of displacements will result in more accurate transmission of large deformation and rotation to the submodel.

- Whether the displacement response of the global model corresponds to the displacement response of the submodel:
  - When the displacements in the global model correspond closely with the expected displacements in the submodel, node-based submodeling is generally preferable.
  - Surface-based submodeling should be used when the submodel displacement response is expected to differ from the global model response. This situation can occur when thermal strains are modeled and the temperature history of the submodel differs from that of the global model; for example, when heat transfer submodeling is performed as part of a sequential thermal-structural analysis.
- The stiffness of the structure:
  - Surface-based submodeling may provide more accurate results for very stiff structures. When the structure is so stiff that only a small component of the global model displacement field contributes to the stress response, numerical roundoff in the displacement results can become significant; for example, when the global model displacement is dominated by a rigid-body motion component.
- The type of output you are interested in from the submodel:
  - Node-based submodeling of displacements will result in more accurate transmission of a displacement field.
  - Surface-based submodeling will result in more accurate transmission of a stress field, and determination of reaction forces in the submodel.

You can use both node-based submodeling and stress-based submodeling in the same model.

Node-based submodeling

Node-based submodeling is the more general technique, supporting a variety of element type combinations and procedures in both Abaqus/Explicit and Abaqus/Standard.

Input File Usage: *SUBMODEL, TYPE=NODE

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Element types supported

Different element types can be used in the submodel than those used to model the corresponding region in the global model.

The following types of submodeling are provided for the node-based approach (global-to-submodel):

- Two-dimensional models:
  - Solid-to-solid
  - Acoustic-to-structure

- Three-dimensional models:
  - Solid-to-solid
  - Shell-to-shell
  - Membrane-to-membrane
  - Shell-to-solid
  - Acoustic-to-structure

A global or submodel region that is meshed with continuum shell elements constitutes a three-dimensional solid region in the submodeling technique. Hence, the use of the submodeling technique for models involving continuum shell elements is the same as with models involving continuum solid elements such as C3D8R or C3D6.

Procedures supported

Both the global model and the submodel can have nonlinear response and can be analyzed for any sequence of analysis procedures. These procedures do not have to be the same for both models. For example, the linear or nonlinear dynamic response of the global model can be used to drive the static, nonlinear response of the submodel on the grounds that the submodel is too small for dynamic effects to be significant in that local region. The global procedure can be performed in Abaqus/Standard to drive a submodeling procedure in Abaqus/Explicit and vice versa. For example, a static analysis performed in Abaqus/Standard can drive a quasi-static Abaqus/Explicit analysis in the submodel. The step time used in these analyses can be different; the time variable of the amplitude functions generated at the driven nodes can be scaled to the step time used in the submodel.

Your submodel cannot refer to a global model step that includes multiple load cases (see "Multiple load case analysis," Section 6.1.4). You must perform the global analysis with a single load definition for the step that will drive the submodel.

Surface-based submodeling

Surface-based submodeling is provided as a complement to the node-based technique, enabling you to drive the submodel with stresses from the global model.

Input File Usage: *SUBMODEL, TYPE=SURFACE

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Element types supported

The following types of submodeling are provided for the surface-based approach (global-to-submodel):

- Two-dimensional models:
  - Solid-to-solid
- Three-dimensional models:
  - Solid-to-solid

Different element types can be used in the submodel than those used to model the corresponding region in the global model. Continuum elements supported for the static analysis procedure are supported for surface-based submodeling, with the following exceptions:

- Cylindrical elements are not supported.
- Continuum shell elements are not supported.

Procedures supported

The surface-based technique is implemented only for static analysis in Abaqus/Standard.

Your submodel cannot refer to a global model step that includes multiple load cases (see "Multiple load case analysis," Section 6.1.4). You must perform the global analysis with a single load definition for the step that will drive the submodel.

Performing a submodeling analysis

A submodeling analysis consists of:

- running a global analysis and saving the results in the vicinity of the submodel boundary;
- defining the total set of driven nodes or driven surfaces in the submodel;
- defining the time variation of the driven variables in the submodel analysis by specifying the actual nodes and degrees of freedom or element-based surfaces to be driven in each step; and
- running the submodel analysis using the "driven variables" to drive the solution.

Linking the global model and the submodel

The submodel is run as a separate analysis from the global analysis. The only link between the submodel and the global model is the transfer of the time-dependent values of variables saved in the global analysis to the relevant boundary nodes of the submodel or to the relevant boundary surfaces. The results from the global model are saved in the results (**`.fil`**) file, in the output database (**`.odb`**) file, or in the SIM database (**`.sim`**) file for the node-based technique. For the stress-based technique, the global model results are saved either in the output database (**`.odb`**) file or in the SIM database (**`.sim`**) file. The transfer is achieved by then reading these results into the submodel analysis. If the global model is defined in terms of an assembly of part instances, the part (**`.prt`**) file from the global model is required for the submodel analysis. Since the submodel is a separate analysis, submodeling can be used to any number of levels; a submodel can be used as the global model for a subsequent submodel.

Accuracy

The global model in a submodeling analysis must define the submodel boundary response with sufficient accuracy. It is your responsibility to ensure that any particular use of the submodeling technique provides physically meaningful results. In general, the solution at the boundary of the submodel must not be altered significantly by the different local modeling. There is no built-in check of this criterion in Abaqus; it is a matter of judgment on your part. In general, accuracy can be checked by comparing contour plots of important variables near the boundaries of the submodeled region.

Specifying the global elements used to drive the submodel

By default, the global model in the vicinity of the submodel is searched for elements that encompass the locations of driven nodes or driven surfaces' faces; the submodel is then driven by the response of these elements. In some cases more than one element can encompass the location of a driven node. For example, adjacent bodies in the global model may have temporarily coincident nodes or surfaces, as depicted in Figure 10.2.1–1.

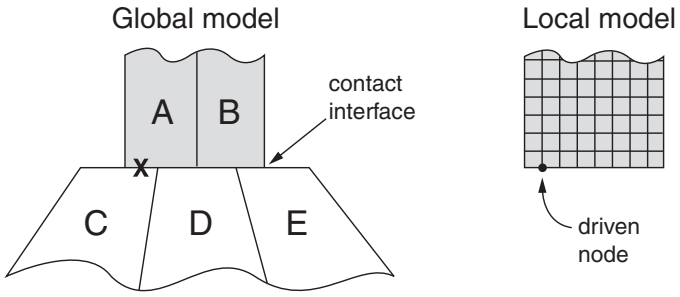


Figure 10.2.1–1 A global model with coincident surfaces in the area of the local model's driven nodes.

In this case the location of the driven node in the corresponding global model is touching both element A and element C; however, only the results from element A should drive the node in the submodel.

To preclude certain elements from driving the submodel, you have the option of specifying a global element set to limit the search to an appropriate subset of the global model.

Input File Usage: *SUBMODEL, GLOBAL ELSET=*name*

If the global model is defined in terms of an assembly of part instances, give the complete name—including the assembly and part instance names—when specifying the global element set. For example, an element set named **`top`** in part instance **`I-1`** of assembly **`Assembly-1`** must be referred to by **`Assembly-1.I-1.top`**.

If the submodel is not defined in terms of an assembly of part instances, the dots in the global element set name must be replaced by underscores: **`Assembly-1_I-1_top`**.

If the global element set is defined at the assembly level, you may provide the element set name without qualifying it with the assembly name in a submodel analysis.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: **Specify**

Minimizing file sizes

The size of the results file or the output database can be minimized for a submodeling analysis by requesting output for only those global nodes and global elements that are used to drive the submodel. To determine which global nodes and/or elements are used to drive the submodel, do the following:

1. Run a data check analysis on the global model with any combination of results file or output database file output requests. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus ("Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).
2. Run a data check analysis on the submodel.

A listing of the global nodes and/or elements that will be used to drive the submodel is output to the data file during the submodeling data check analysis.

Frequency of output

Pay special attention to the frequency at which you request output in the global model (see "Output to the data and results files," Section 4.1.2, and "Output to the output database," Section 4.1.3). It is possible to define the results file output or nodal and element output to the output database file such that the information is written at different frequencies for different nodes and elements, although that should not be done for nodes and elements involved in the interpolation to define values at driven variables since Abaqus will take values at the coarsest frequency only. To avoid this problem, write the nodal and elemental output to the output database or the results file using the same frequency for all nodes and elements involved in the interpolation and choose a frequency that will allow the history in the submodel to be reproduced accurately.

Input File Usage: To control the output frequency to the Abaqus/Standard results file, use the following option:

*NODE FILE, FREQUENCY

To control the output frequency to the Abaqus/Explicit results file, use the following option:

*FILE OUTPUT, NUMBER INTERVAL

To control the output frequency to the output database, use the following option:

*OUTPUT, FIELD, FREQUENCY

Abaqus/CAE Usage: Step module: **Output→Field Output Requests→Create**: **Frequency**

Material options

Any of the material models described in Part V, "Materials," can be used in the global and submodel analyses. The material response defined for the submodel may be different from that defined for the global model.

Elements

The dimensionality of the submodel must be the same as that of the global model: both models must be either two-dimensional or three-dimensional. The following limitations apply:

- The boundary nodes of the submodel must lie within regions of the global model where Abaqus is able to perform spatial interpolation to define the values of the driven variables. Therefore, they must lie within (or, as allowed by the exterior tolerance, near to) two- or three-dimensional geometrically defined elements in the global model. Such geometrically defined elements are:
  - first- or second-order triangles or quadrilaterals in two dimensions;
  - first- or second-order triangular or quadrilateral shells; and
  - first- or second-order tetrahedra, wedges, or bricks in three dimensions.
- The boundary nodes cannot lie in regions of the global model where there are only one-dimensional elements (beams, trusses, links, axisymmetric shells) since Abaqus does not provide the necessary interpolation of results for such elements.
- The boundary nodes cannot lie in regions of the global model where there are only user elements, substructures, springs, dashpots, cohesive elements, etc. since those element types do not allow for geometric interpolation.
- The boundary nodes cannot lie in regions of the global model where there are only axisymmetric solid elements with nonlinear, asymmetric deformation (CAXA elements). The submodeling capability is currently not supported for these elements.
- The reference node associated with generalized plane strain elements (CPEG) cannot be used as a driven boundary node in a submodeling analysis.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see “Abaqus/Standard output variable identifiers,” Section 4.2.1, and “Abaqus/Explicit output variable identifiers,” Section 4.2.2).

As described above, nodal output requests to the results file or output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

10.2.2 NODE-BASED SUBMODELING

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Submodeling: overview," Section 10.2.1
- *SUBMODEL
- *BOUNDARY
- Chapter 38, "Submodeling," of the Abaqus/CAE User's Guide

Overview

The following types of node-based submodeling are available:

- Same-to-same (e.g., solid-to-solid, shell-to-shell);
- Shell-to-solid; and
- Acoustic-to-structure.

These submodel types support the following nodal-driven variables:

- Displacement,
- Rotation,
- Temperature,
- Pore pressure, and
- Acoustic pressure.

Performing a node-based submodeling analysis

For an overview of submodeling that includes some details common to both node-based and surface-based submodeling, see "Submodeling: overview," Section 10.2.1.

Your submodel analysis is driven, either partly or completely, from the results obtained from a global model analysis. The results from the global model are interpolated onto the nodes on the appropriate parts of the boundary of the submodel (see Figure 10.2.2–1). Thus, the response at the boundary of the local region is defined by the solution for the global model. The driven nodes and any loads applied to the local region determine the solution in the submodel.

Different types of node-based submodeling

Three different techniques are available for nodal-based submodeling.

Solid-to-solid submodeling

The linear or nonlinear response of a global solid model can be used to drive the submodel response of a solid submodel. The driven variables can be displacements or temperatures.

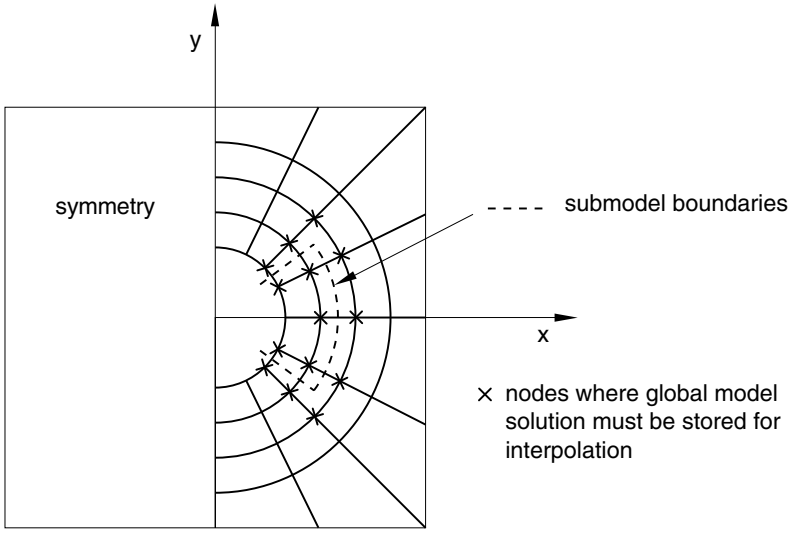


Figure 10.2.2–1 The global model.

Shell-to-solid submodeling

The linear or nonlinear response of a global shell model can be used to drive the submodel response of a solid submodel. The driven variables are displacements, which are determined from global model displacements and rotations.

Acoustic submodeling

The linear or nonlinear response of a global, structural model can be used to drive the acoustic response of a fluid region of any size if the forces exerted on the structure by the fluid are small. This is often the case for metal structures in air, building interiors, or for sound propagation from a liquid to air. In the case of a liquid and a gas, no special procedures need be followed; the pressure degrees of freedom couple straightforwardly. In the case of a structure driving a fluid, you must ensure that the degrees of freedom to be driven in the submodel exist among the global model results. Several alternatives exist. A thin layer of fluid elements, with the same properties as the submodel fluid, can be added to the global model; this element set and its nodes can then be used to drive the submodel in the usual manner. Alternatively, you can create acoustic interface elements on the surface of the submodel and drive the corresponding nodes with the structural nodes (see "Fully and sequentially coupled acoustic-structural analysis of a muffler," Section 9.1.1 of the Abaqus Example Problems Guide).

In problems where the fluid exerts large pressures on the structure, the mechanical response of the structure may be of interest. Acoustic-to-structure submodeling can be used in such problems. The submodel in these problems is a part of the structural component of the global model. The acoustic pressure obtained from solving a coupled acoustic-structural global analysis is used to drive the submodel on the surface it shares with the fluid medium. Other boundaries of the submodel may be driven using

the displacements of the structural component of the global model via solid-to-solid submodeling. The acoustic-to-structure submodel analysis solves an uncoupled structural force-displacement problem. The acoustic pressure from the global model is interpolated to the submodel driven nodes. The tributary area and the outward normal associated with the driven node are used to convert the interpolated acoustic pressure to a concentrated load acting at that location (see "Miscellaneous submodeling tests," Section 3.7.17 of the Abaqus Verification Guide).

Saving the results from the global model

The results from the global analysis must be saved at all nodes required for the interpolation of the driven variables to the boundary of the submodel (see Figure 10.2.2–1). The results (`.fil`) file or the output database (`.odb`) file can be used for this purpose.

Saving the results to the results file

In each step of the global model whose solution will be used to drive the submodel, write the nodal results for all driven variables to the results file (see "Output to the data and results files," Section 4.1.2). These results must be written in the global coordinate system of the model. The submodel can refer only to a global model results file that is from a binary compatible platform.

When the global model is run in Abaqus/Explicit and results file output is requested, the results are written to the selected results (`.sel`) file; this file needs to be converted into a results (`.fil`) file using the **convert** option (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Input File Usage: *NODE FILE
(In Abaqus/Standard GLOBAL=NO should not be used on the *NODE FILE option.)

Abaqus/CAE Usage: You cannot write output to the results file in Abaqus/CAE.

Saving the results to the output database

In each step of the global model whose solution will be used to drive the submodel, write the nodal results for all driven variables to the output database in ODB or SIM format (see "Output to the output database," Section 4.1.3). Unlike the results file, nodal output to the output database is always written in the global directions. The output database can be transferred to any platform since it is binary neutral.

Input File Usage: Use both of the following options:

*OUTPUT, FIELD
*NODE OUTPUT

Abaqus/CAE Usage: Step module: **Output→Field Output Requests→Create**

Saving results with higher precision

By default, the nodal output to the output database is written using single precision, which may not be sufficient for certain classes of problems; for example, submodels undergoing large rigid body motions (consider also surface-based submodeling in these cases—see "Surface-based submodeling,"

Section 10.2.3). For such analyses request the nodal output to the fullest possible precision (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Input File Usage: **abaqus job**=*global_model_input_file* **output_precision**=**full**

Abaqus/CAE Usage: Job module: **Create Job**: **Precision**: **Nodal output precision**: **Full**

Saving results from a global model with a physical time scale

If the global analysis in Abaqus/Standard involves a physical time scale and the results file is to be used in the submodel analysis, request that the results file output be written at the beginning of the step (the zero increment) for all steps in the global analysis (see "Output," Section 4.1.1). Abaqus will then have the complete solution history (including the solution state at the beginning of a step) from which a submodel may be driven. If the zero increment results are not requested, incorrect results will be obtained if the step time in the submodel is less than the step time of the first increment on the results file. Instead of interpolating between the results at the start of the step and the results of the first increment on the results file, Abaqus will simply use the results of the first increment as long as the submodel step time is less than the step time of the first increment on the results file. The zero increment request is not required in Abaqus/Explicit, because the results are always written to the results file at the beginning of each step. Similarly, the results will always be correctly interpolated when using the output database to transfer the results from the global model to the submodel, because the zero increment is always written to the output database.

Input File Usage: *FILE FORMAT, ZERO INCREMENT

Abaqus/CAE Usage: You cannot write output to the results file in Abaqus/CAE.

Referring to the global model results from the submodel analysis

You must define the source of the global solution results. Provide the name of the global results file or output database file (in ODB or SIM format); the file extension is optional. If the file extension is omitted, Abaqus will use in order, the results file, the ODB output database file, or the SIM database file.

Input File Usage: **abaqus job**=*submodel_input_file* **globalmodel**=*global_results_file* or *global_output_database* or *sim_database_file*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*submodel***: **Submodel**: **Read data from job**: *global_results_file* or *global_output_database*

Reading data from a SIM database file is not supported in Abaqus/CAE.

Specifying the driven nodes in the submodel

Specifying the driven nodes does not activate the driven variables: they must be activated by specifying the appropriate submodel boundary conditions.

All nodes of the submodel where variables will be driven in any step (see Figure 10.2.2–2) must be specified as driven nodes since the list of nodes cannot be extended subsequent to its initial definition (even at restart). However, variables at the nodes given do not have to be driven in all steps: the choice of which variables are driven in a particular step is made as part of a submodel boundary condition definition, as discussed later.

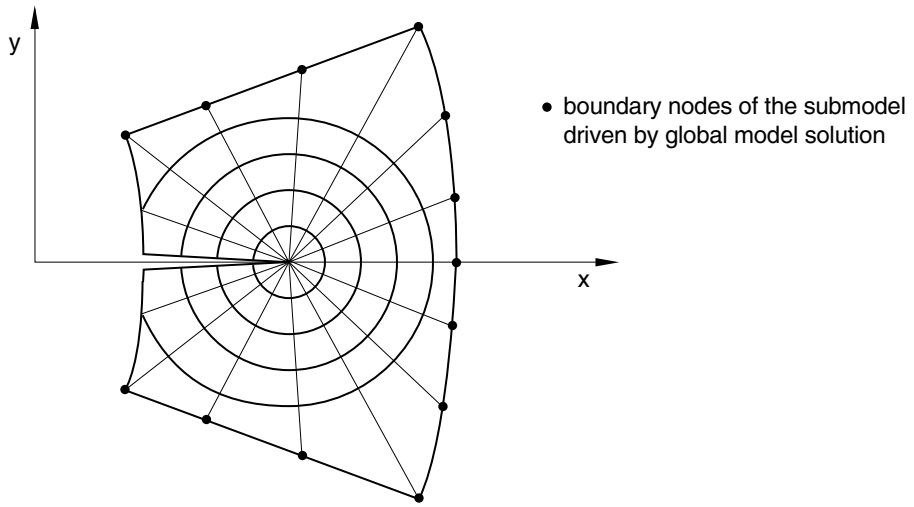


Figure 10.2.2–2 The magnified submodel.

Input File Usage: ∗SUBMODEL
list of nodes or node set labels or, for acoustic-to-structure submodeling, the name of an element-based structural surface

The ∗SUBMODEL option must be included in the model definition portion of the input file for the submodel analysis. Multiple ∗SUBMODEL options are allowed; however, in this case you must ensure that the driven nodes specified on the data line of one option are separate and distinct from the nodes specified on the data lines of all the other options.

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region

Specifying the driven nodes in shell-to-solid submodeling

In shell-to-solid submodeling, the submodel is made up of solid elements and replaces a region where conventional shell elements are used in the global model. In this case Abaqus expects that all the driven nodes on the submodel belong to solid elements and are driven from a global model region that is entirely made up of shell elements. The boundary where the submodel is driven is a set of surfaces in the submodel but is a set of lines in the shell reference surface in the global model, as shown in Figure 10.2.2–3. The dashed line $A - B - C$ on the shell model is replaced by the shaded surfaces of the solid element submodel.

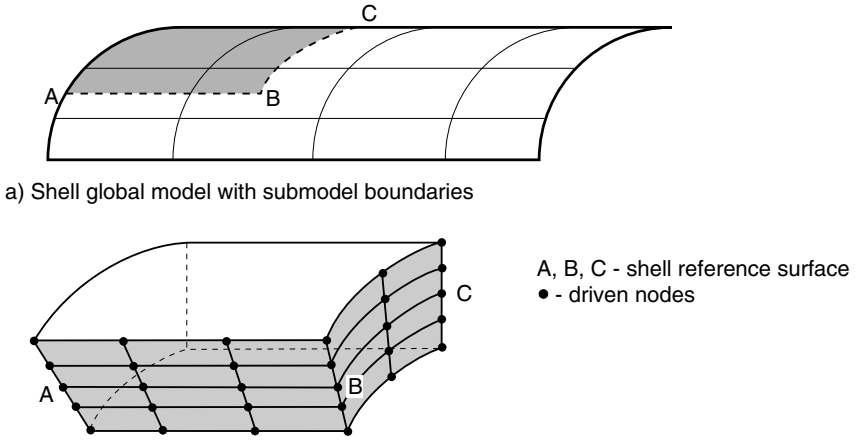


Figure 10.2.2–3 Shell-to-solid submodeling.

Whenever shell-to-solid submodeling is used, you must define the maximum shell thickness in the global model, given in the units used for the models. If a shell offset is defined in the global model, the shell thickness must be set equal to twice the maximum distance from the top or bottom shell surface to the shell reference surface.

Input File Usage: ∗SUBMODEL, SHELL TO SOLID, SHELL THICKNESS=*thickness*

If more than one ∗SUBMODEL option is used, the SHELL TO SOLID parameter must be included on every option.

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*submodel*: Submodel**: **Shell global model drives a solid submodel**
Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Shell thickness:** *thickness*

Specifying the driven nodes in acoustic-to-structure submodeling

The global analysis for acoustic-to-structure submodeling problems is performed as a coupled acoustic-structural analysis. The acoustic nodal pressures from the global analysis must be written to the results file for the acoustic mesh in contact with the structural surface of interest. In the submodel analysis acoustic pressures from the global analysis drive the user-specified structural surface of interest. The driven nodes for the submodel are the nodes lying on the specified surface. Only element-based surfaces are allowed in acoustic-to-structure submodeling.

Input File Usage: ∗SUBMODEL, ACOUSTIC TO STRUCTURE,
ABSOLUTE EXTERIOR TOLERANCE=*value*

Abaqus/CAE Usage: Acoustic-to-structure submodeling is not supported in Abaqus/CAE.

Specifying driven nodes for shells with acoustic pressures on both sides

In certain problems the acoustic pressure may act on both sides of a shell structure. Figure 10.2.2–4 shows a section of a global model consisting of a shell structure that is sandwiched between two acoustic media.

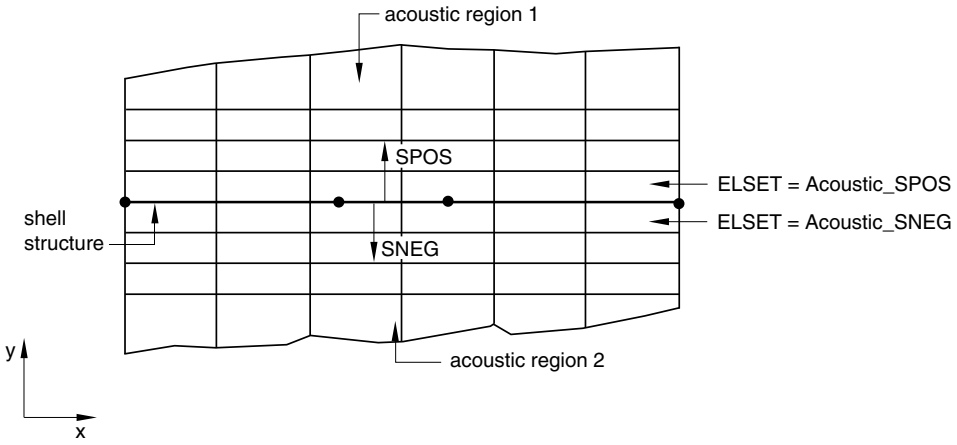


Figure 10.2.2–4 A cross-section of the acoustic-to-structure global model with acoustic regions on both sides of the shell.

Separate element sets consisting of acoustic elements on the positive and negative sides of the shell are defined, respectively. The nodal pressures for nodes attached to elements in these sets are written to the selected results file. Figure 10.2.2–5 shows the submodel that consists only of the refined shell structure.

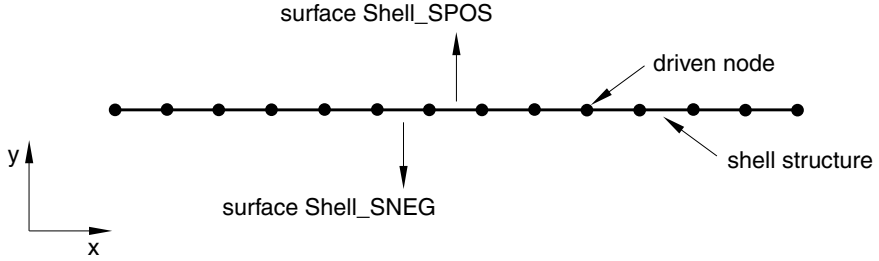


Figure 10.2.2–5 The acoustic-to-structure submodel with acoustic pressure on both sides of the shell.

Two separate surfaces are defined on the SPOS and SNEG sides, respectively. To apply the acoustic pressure from the global analysis on each side of the shell correctly, you must specify the surface name along with the corresponding acoustic element set.

Input File Usage: *SUBMODEL, ACOUSTIC TO STRUCTURE, GLOBAL
ELSET=*Acoustic_SPOS*
Shell_SPOS

*SUBMODEL, ACOUSTIC TO STRUCTURE, GLOBAL
ELSET=*Acoustic_SNEG*
Shell_SNEG

Abaqus/CAE Usage: Acoustic-to-structure submodeling is not supported in Abaqus/CAE.

Defining geometric tolerances

A geometric tolerance is used to define how far a boundary node in the submodel can lie outside the exterior surface of the global model, as that surface is interpolated in the global, undeformed finite element model. By default, nodes in the submodel must lie within a distance calculated by multiplying the average element size in the global model by 0.05. You can change the tolerance, which is useful in cases where submodel driven nodes lie to a greater extent outside the global model exterior surface. Tolerances larger than this default value, however, may result in significantly greater computation times and lower accuracy in the driven solution for driven nodes significantly outside the global model exterior surface.

You can define the geometric tolerance as a fraction of the size of the average element in the global model or as an absolute distance in the length units chosen for the model. If both tolerances are defined, Abaqus uses the tighter tolerance.

Input File Usage: Use the following option to define the geometric tolerance as an absolute distance:

*SUBMODEL, ABSOLUTE EXTERIOR TOLERANCE=*tolerance*

Use the following option to define the geometric tolerance as a fraction of the size of the average element in the global model:

*SUBMODEL, EXTERIOR TOLERANCE=*tolerance*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Exterior tolerance: absolute:** or **relative:** *tolerance*

The exterior tolerance in solid-to-solid submodeling

The exterior tolerance for a solid-to-solid submodel analysis is indicated by the shaded region in Figure 10.2.2–6. If the distance between the driven nodes and the free surface of the global model falls within the specified tolerance, the solution variables from the global model are extrapolated to the submodel.

The exterior tolerance in shell-to-shell submodeling

In a shell-to-shell submodel analysis Abaqus checks whether the driven nodes of the submodel lie sufficiently close to the reference surface of the shell elements in the global model. To simplify calculations, the closest point in the global model is calculated as the intersection of a line drawn

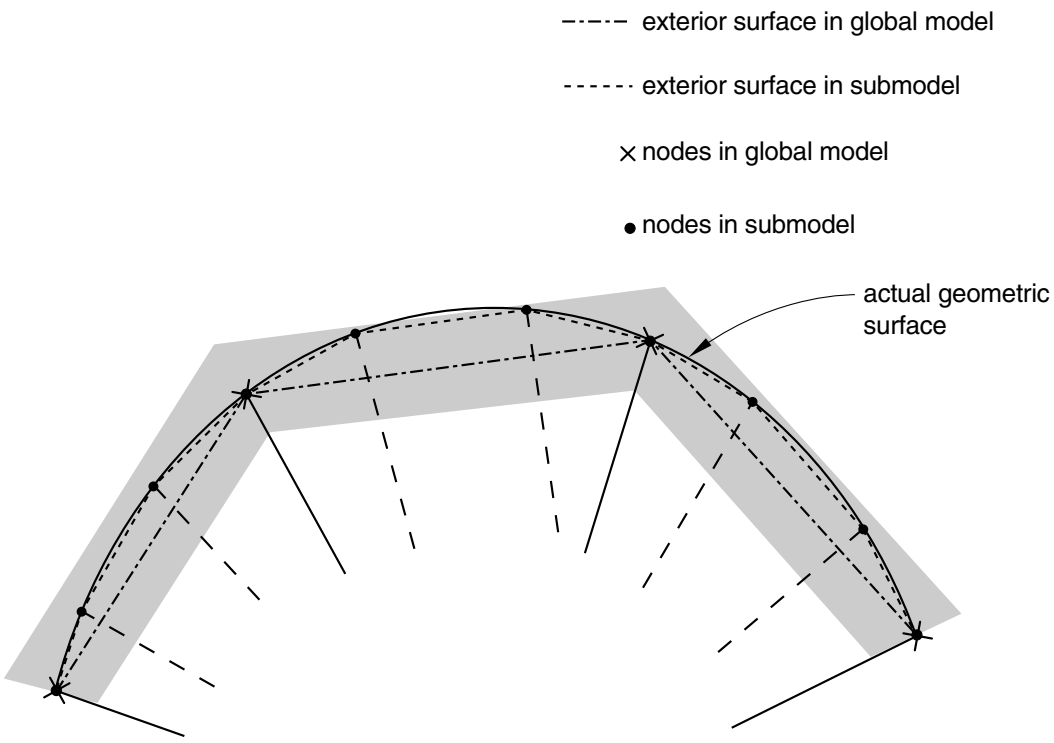


Figure 10.2.2–6 The exterior tolerance in solid-to-solid submodeling.

through the node on the submodel with the reference surface of the shell in the global model. The direction of the line is normal to a flat surface approximation to each shell element. The normal to the flat surface is the average of the normals at the nodes of the shell element. The distance checked against the specified exterior tolerance is shown in Figure 10.2.2–7.

The exterior tolerance in shell-to-solid submodeling

For shell-to-solid submodeling Abaqus uses two kinds of tolerances to determine the relationship between the submodel and the global model. First, the closest point on the shell reference surface of the global model is determined. This point, the "image node," is shown in Figure 10.2.2–8. The user-specified exterior tolerance is used to check if the image node lies within the domain of the global model. Then the distance, D, between the driven node and its image is checked; if the distance is less than half the value of the specified shell thickness plus the exterior tolerance, it is accepted. This check is only approximate if the global model has varying shell thickness or if the shell reference surface is offset from the midsurface.

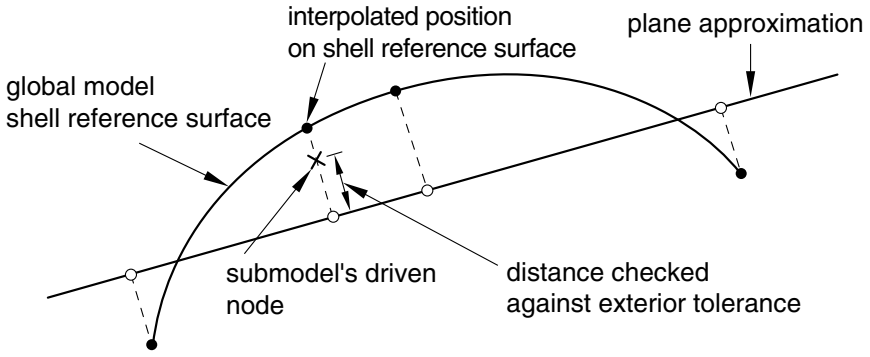


Figure 10.2.2–7 Flat surface approximation in shell-to-shell submodeling.

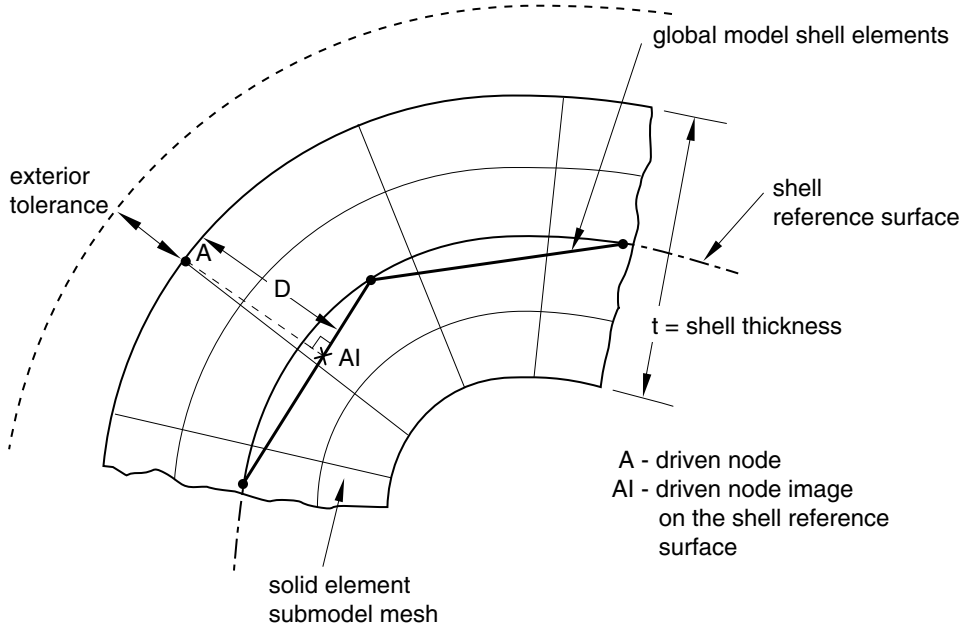


Figure 10.2.2–8 The exterior tolerance in shell-to-solid submodeling.

Permitting driven nodes to be excluded from submodeling

In some cases (such as when your submodel geometry is more detailed than the global model in regions near a free surface) you may specify driven nodes that Abaqus will find, even when accounting for the search tolerance, to be outside the region of the global model elements. By default, these cases result

in an error message. In solid-to-solid submodeling you can, however, specify that Abaqus ignore driven nodes that cannot be found. Use this option with caution and always evaluate the list of nodes that are labeled as not found. Most cases where Abaqus finds driven nodes to lie outside the global model reflect a modeling error and use of the intersection only option may lead to incorrect results in these cases.

Input File Usage: Use the following option to specify that Abaqus ignore driven nodes that cannot be found in the global model elements:

```
*SUBMODEL, INTERSECTION ONLY
list of nodes or node set labels
```

The driven nodes ignored through the use of the INTERSECTION ONLY parameter are then ignored in all subsequent submodel boundary condition references.

Defining the driven variables in the submodel

The actual driven variables are defined in any step as a submodel boundary condition. The boundary conditions are "driven variables" obtained from the results or output database file of the global analysis.

The degrees of freedom on the driven nodes of the submodel must exist at the forcing nodes of the global model. In a problem involving an acoustic fluid submodel driven by a structural global model, for example, acoustic interface elements should be created on the submodel's driven boundary with the structure.

For solid-to-solid and shell-to-shell submodeling specify the individual degrees of freedom to be driven. In most cases all components of the solution variables (displacements, rotations, temperatures, etc.) at these nodes are driven by the global solution, although you may choose to drive only some components at any of the driven nodes. For shell-to-solid submodeling the driven degrees of freedom are chosen automatically based on a user-specified zone around the shell reference surface, as explained later.

Abaqus/Explicit does not admit jumps in displacement and rotation boundary conditions (see "Prescribed displacement" in "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1); any jumps in the driven displacements and rotations will be ignored.

It is not recommended to have all the variables at all the nodes in the submodel driven by the global solution.

For acoustic-to-structure submodeling, the loads due to acoustic pressure acting at the driven nodes of the submodel are activated by specifying pressure (degree of freedom 8) along with the driven node set.

Only one submodel boundary condition can be specified in each step of the analysis.

Input File Usage: *BOUNDARY, SUBMODEL

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Degrees of freedom:** *degrees of freedom*

Specifying the step number from the global analysis

You specify the step of the global model history that is to be used for the driven variables in the current submodel analysis step. When the global solution is obtained from the results file, the zero increment is included if it was requested in the global analysis (see "Output," Section 4.1.1).

In a general analysis step or a direct-solution steady-state dynamic analysis step, Abaqus calculates the amplitudes for the driven variables as functions of time or frequency from the results of the global model.

Input File Usage: *BOUNDARY, SUBMODEL, STEP=*step*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number:** *step*

Scaling the global time period to the submodel time period

The global analysis and submodel analysis can have different time steps. You can scale the time variable of the driven nodes from the global analysis to the step time of the submodel analysis. This technique is useful when the analyses are static or quasi-static in nature; the use of this technique in dynamic analyses with significant inertial effects is not recommended. If the same step time is used in both the global model and the submodel, the time scale has no effect. The time scale cannot be specified in frequency domain analyses or in linear perturbation steps.

Abaqus will determine the values that the driven variables will follow throughout the step in the submodel analysis by using the points in time at which the global solution results or output database file was written. When the time variable of the driven nodes of the global analysis is scaled and if the step time is different from the submodel step time, the points in time of the driven variables are scaled to the submodel step time.

Input File Usage: *BOUNDARY, SUBMODEL, STEP=*step*, TIMESCALE

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Scale time period of global step to time period of submodel step**

Scaling the magnitude of driven variables

For displacement-based submodeling the magnitude values of driven variables are obtained by multiplying the displacement history as obtained from the global analysis by a scaling parameter. You can scale the driven variables by setting the scaling parameter in the definition of the submodel boundary conditions. This technique is useful in scaling the submodel boundary conditions in a multiple-step analysis without rerunning the global model. It can be used in Abaqus/Standard and Abaqus/Explicit for the same-to-same and shell-to-solid cases except for acoustic-to-structure submodeling.

Input File Usage: *BOUNDARY, SUBMODEL, STEP=*step*, SCALE=*scalarValue*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Scale:** *scale*

Modifying the set of driven variables

You can modify the submodel boundary condition to add new variables to the list of driven variables, you can remove variables from the driven variable set, and you can reintroduce them later (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). New nodes cannot be added to the total set of driven nodes defined for the submodel; this set of driven nodes is a fixed part of the model definition.

Input File Usage: Use one of the following options:

*BOUNDARY, SUBMODEL, OP=MOD
*BOUNDARY, SUBMODEL, OP=NEW

Abaqus/CAE Usage: Load module: boundary condition editor: **Degrees of freedom**

Automatically selecting the driven variables in shell-to-solid submodeling

For shell-to-solid submodeling the driven degrees of freedom at the driven nodes are chosen automatically, depending on the distance between the driven node and the global model shell reference surface. All displacement components are driven at nodes that lie on the reference surface or within a "center zone," as shown in Figure 10.2.2–9. The size of the center zone is specified as part of the submodel boundary condition definition, as described below. For nodes that lie further away from the reference surface, only the displacement components parallel to the shell reference surface are driven. At least one layer of nodes in the submodel must be within the center zone; if no nodes are found this close to the reference surface, Abaqus issues an error message.

Specifying the size of the center zone in shell-to-solid submodeling

The center zone method of prescribing driven variables usually provides a reasonable transfer of the plane stress assumption in the shell model. The width of this zone around the reference surface where all displacement components are driven may be different for various driven nodes or node sets. If you do not provide values for the center zone size, a default value of 10% of the maximum of the specified shell thicknesses is assumed.

For complicated geometries it can be advantageous to assign a different center zone size to different nodes or node sets.

You can view the driven nodes lying inside and outside the center zone in Abaqus/CAE by displaying the model boundary conditions (**View→ODB Display Options**) in the Visualization module.

Input File Usage: *BOUNDARY, SUBMODEL, STEP=*step*
nodes, *center zone size*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Center zone size:** *center zone size*

Transferring transverse shear stresses in shell-to-solid submodeling

Usually it is enough for the layer of nodes closest to the shell reference surface to lie inside the center zone. If a very fine solid mesh is used in the thickness direction and substantial transverse shear stresses

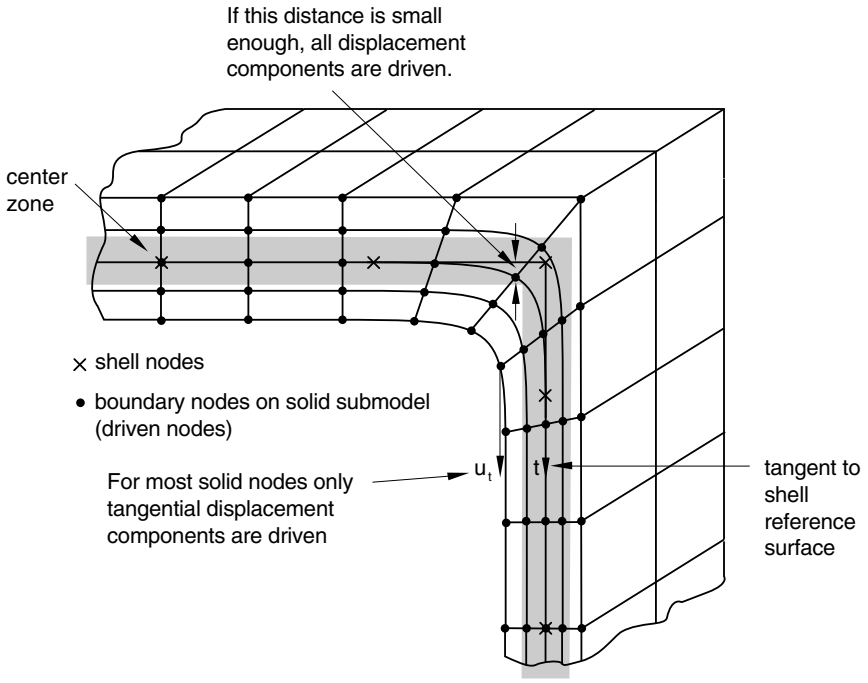


Figure 10.2.2–9 Center zone choice in shell-to-solid submodeling.

are transferred, it may be necessary to make the center zone size large enough that multiple layers of nodes lie inside the zone. However, if the transverse shear stresses at the submodel boundary are high and the submodel is highly refined in the thickness direction, high local stresses may develop since the shear force at the submodel boundary is transferred only at the driven nodes within the center zone. High transverse shear stresses occur only in regions where bending moments vary rapidly; it is better not to locate the submodel boundary in such regions. It is best to locate the submodel boundary in areas of low transverse shear stress in the global model.

Special considerations

There are several special considerations that are worth noting.

Specifying the shell thickness in shell-to-shell submodeling

For shell-to-shell submodeling the shell thickness generally is not changed between the models. You can specify different shell thicknesses if, for example, a local thickness change is being investigated; however, Abaqus does not check the validity of these differences.

Limitations in shell-to-solid submodeling

The following limitations and special cases apply to the shell-to-solid capability:

- The global model can contain both solid and shell elements; however, when the shell-to-solid capability is used, all driven nodes must lie within shell elements in the global model. If the driven boundary lies at the border between a solid and a shell region, the driven nodes must be moved a small distance away from the solid region (see Figure 10.2.2–10).
- Corners or kinks may exist in global models made of shell elements. At such corners or kinks the shell elements only approximate the distribution of the material away from the midsurface of the shell (see Figure 10.2.2–11). Because of such approximations, it is not possible to drive a submodel correctly if the driven nodes of the submodel lie within a shell thickness from a corner or a kink. If necessary, use the approach shown in Figure 10.2.2–11. A better approach is to include the corner or kink as part of the submodel and drive it from nodes well away from corners or kinks since they are a source of stress concentration and high stress gradients (see Figure 10.2.2–12).
- Temperature degrees of freedom cannot be driven in shell-to-solid submodeling.

Alternative to shell-to-solid submodeling

An alternative to shell-to-solid submodeling is the surface-based shell-to-solid coupling capability discussed in "Shell-to-solid coupling," Section 35.3.3.

Procedures

Neither the coupled thermal-electrical procedure nor any of the mode-based dynamics procedures can be used on the submodel level. In addition, submodeling cannot be used in conjunction with symmetric model generation or symmetric results transfer. Adaptive meshes should not be used in the global model. However, they can be used in the submodel analysis; Abaqus will always treat the driven nodes in the submodel as Lagrangian nodes.

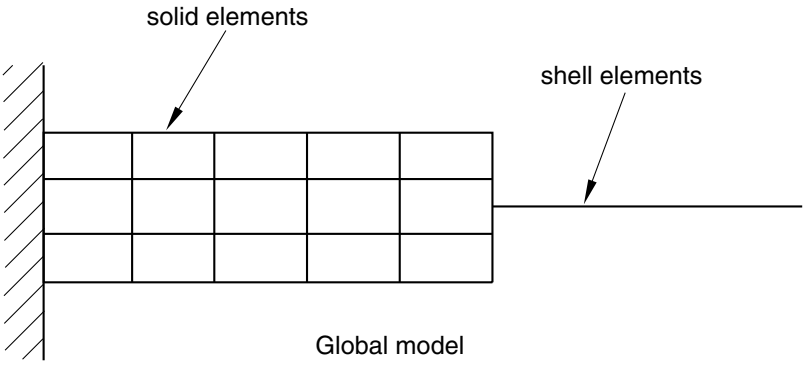
Both general (possibly nonlinear) and linear perturbation steps can be used in submodeling (see "General and linear perturbation procedures," Section 6.1.3, for a discussion of general and linear perturbation steps).

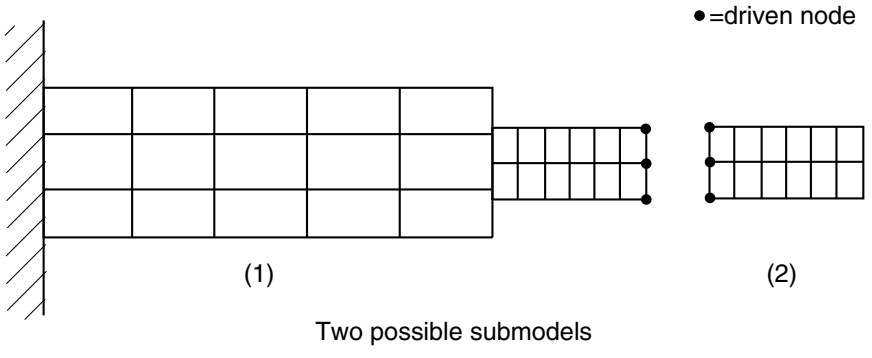
Submodeling in dynamic procedures

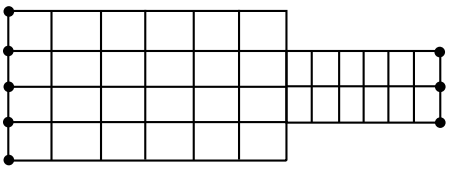
The submodeling capability can be used in the dynamic procedures using explicit integration (in Abaqus/Explicit) and in the dynamic procedures using direct integration (in Abaqus/Standard). The following combinations of procedures between the global model and the submodel can be considered: explicit dynamic, implicit dynamic, dynamic coupled thermal-stress, and coupled thermal-stress. In dynamic problems in which inertial forces are significant, the global model and the submodel need to be run for the same step time intervals.

In Abaqus/Explicit a quasi-static analysis is performed as a dynamic procedure. For this case and for the static analyses performed in Abaqus/Standard, the time step of the global model and submodel can be different. The time variable of the driven nodes from the global analysis must be scaled to the step time of the submodel analysis to match the time variable of the amplitude functions generated at the driven nodes to the step time used in the submodel.

For significantly dynamic problems in Abaqus/Explicit, a sufficiently large number of intervals need to be written to the results or output database file for the global model. Preferably the displacement results







Incorrect submodel (driven nodes in both shell and solid regions of the global model)

Figure 10.2.2–10 A limitation of shell-to-solid submodeling.

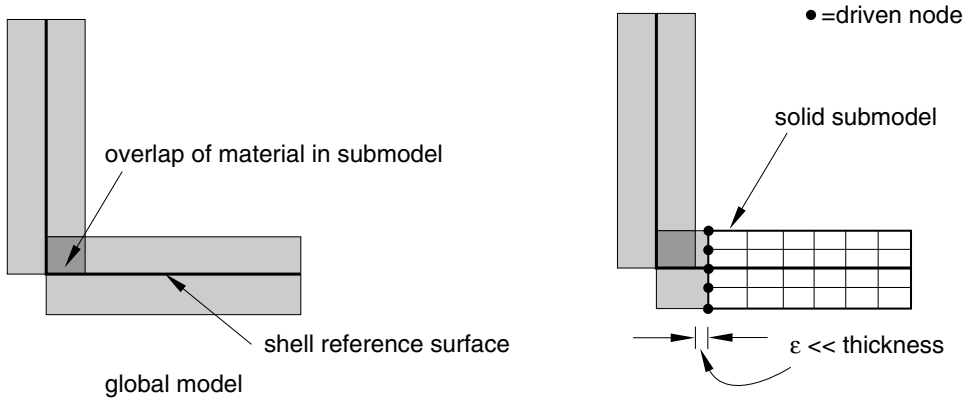


Figure 10.2.2–11 Shell-to-solid submodeling around corners.

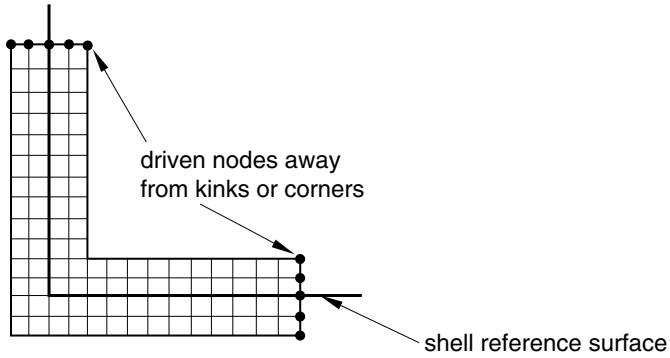


Figure 10.2.2–12 Solid submodel of a shell intersection.

for the nodes that are used to drive the submodel should be saved for each increment. This caution is necessary in particular for problems with elastic material properties to avoid possible aliasing (under sampling), which can cause solution distortion in the submodel. These requirements do not apply to quasi-static problems.

Interpreting acceleration results

When you drive a submodel boundary with global model displacement results, the displacements are interpreted as a smoothed piecewise linear function in time, similar to how you would apply a displacement boundary condition using a tabular amplitude definition (see "Using an amplitude definition with boundary conditions" in "Amplitude curves," Section 34.1.2). This smoothed function typically results in displacements and velocities at the driven nodes that agree reasonably with the global model. Acceleration results at the driven boundary, however, are generally not in good agreement with the global model as they reflect the shape of the displacement history smoothing rather than the global

model acceleration results (information that is not available from a piecewise linear global-model displacement history). The submodel acceleration results away from the submodel driven nodes are less affected by this smoothing and are typically in good agreement with the global model response.

Obtaining a solution at a particular point in time using linear perturbation analysis

In Abaqus/Standard it is possible to study the submodel's linearized response corresponding to a particular point in time in the global solution by using a static, linear perturbation procedure in the submodel analysis. You can select the increment in the global analysis step that is to be used as the basis for calculating the values for the driven variables. If you do not select an increment in a static linear perturbation step, the last increment of the selected step in the global analysis is used as the basis for calculating the values for the driven variables. You cannot select an increment in a general submodel step.

Input File Usage: *BOUNDARY, SUBMODEL, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number**: *step*, **Global increment**: *increment*

Submodeling in the frequency domain

The submodeling capability can be used in the frequency domain by using the direct-solution steady-state dynamics procedure. Mode-based steady-state dynamics cannot be used at the submodel level.

The only restriction on the specification of the frequency range in the submodel is that the minimum and maximum frequency should lie within the range of calculated frequencies in the global model. Abaqus will interpolate the solution variables from the global model in the frequency domain, as well as spatially, before applying them to the submodel. The results will be most accurate if the frequencies at which the response in the submodel is requested match the frequencies at which the response was calculated in the global model. This is particularly true in the vicinity of the eigenfrequencies of the global model.

In the global model you must write both the amplitude and the phase of the nodal displacements to the results file so that Abaqus can apply the real and imaginary parts of the solution at the driven nodes in the submodel. If you are using the output database to drive the submodel, you need to request only nodal displacement output since displacement output to the output database includes both real and imaginary parts.

Mixing general and linear perturbation steps

It is possible to mix general steps and linear perturbation steps in both the global and the submodel analyses. Abaqus allows general analysis steps to be treated as linear perturbation steps during submodeling, and vice versa.

Example: Submodeling with general and linear perturbation steps

For an example of submodeling that uses both general and linear perturbation steps, consider the following situation. The global analysis consists of a static preload—done as a general, nonlinear,

analysis step—followed by extraction of the eigenmodes of the preloaded structure, then a step of 5 seconds of modal dynamic response analysis:

```
*STEP
** Apply preload
*STATIC
 0.1, 1.0
…
** Write out results for nodes needed to
** interpolate to the submodel's boundary
*NODE FILE, NSET=DETAIL
 U
*END STEP
*STEP
** Calculate modes and frequencies
*FREQUENCY
…
** The *NODE FILE option is repeated because
** this is the first linear perturbation step
*NODE FILE, NSET=DETAIL
 U
*END STEP
*STEP
** Dynamic response of preloaded system
*MODAL DYNAMIC

 0.01, 5.0
…
*END STEP
```

We wish to study the local, possibly nonlinear, response of a part of this model that is so small that we do not need to model dynamic effects locally and can, thus, perform two steps of static analysis:

```
** Define submodel boundary (driven nodes)
*SUBMODEL
PERIM
*STEP
** Preload
*STATIC
 0.1, 1.0
*BOUNDARY, SUBMODEL, STEP=1
…
*END STEP
*STEP
```

```
** Local static response to global dynamic step
*STATIC
 0.01, 5.0
*BOUNDARY, SUBMODEL, STEP=3
…
*END STEP
```

It is perfectly acceptable that the submodel analysis requests general, possibly nonlinear, analysis for both steps, while in the global analysis the dynamic step was a linear perturbation step (modal dynamics is always a linear perturbation analysis). It is your responsibility to judge that this use of the submodeling feature is reasonable. For example, suppose that the global analysis were continued with a fourth step of general, nonlinear static response:

```
*RESTART, READ, STEP=3
** Read state at end of initial preload
** (could equally well use *RESTART, READ, STEP=1)
*STEP
** Add more preload
*STATIC
 0.2, 1.0
…
*END STEP
```

This fourth general analysis step starts with the state at the end of general analysis Step 1 because the frequency extraction and the modal dynamic steps are both linear perturbation steps. However, if we restart the submodel analysis in the same way, the solution may not be comparable with the global model solution:

```
*RESTART, READ, STEP=2
** Read state at end of step 2
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*BOUNDARY, SUBMODEL, STEP=4
…
*END STEP
```

The second step in the submodel is a general analysis step, to which the response may be nonlinear, thus changing the state of the model. A valid alternative would be to apply the Step 4 response to the submodel immediately after the first step:

```
*RESTART, READ, STEP=1
** Read state at end of preload step
*STEP
```

```
** Add more preload
*STATIC
 0.2, 1.0
*BOUNDARY, SUBMODEL, STEP=4
…
*END STEP
```

Reinterpreting solution variables in the submodel analysis

During general analysis steps Abaqus works in terms of total solution variables such as the displacements, u. In linear perturbation steps Abaqus works in terms of the displacement perturbation, Δu, about a base state, $u\big|_0$. When general analysis steps and linear perturbation steps are reinterpreted in the submodel analysis, the global analysis results are treated as defined in Table 10.2.2–1.

Table 10.2.2–1 Reinterpreting solution variables in the submodel analysis.

Global analysis step basis	Submodel step basis	Global increment specified in definition of submodel boundary condition	Driven variable basis
General	General	none	$u_s = u_g$
Linear perturbation	General	none	$u_s = u_s\big\|_0 + \Delta u_g$
General	Static, linear perturbation	i	$\Delta u_s = u_g\big\|_i - u_s\big\|_0$
Linear perturbation	Static, linear perturbation	i	$\Delta u_s = \Delta u_g\big\|_i$

In this table

u_s is the current value of a driven variable in the submodel at any time during a general, nonlinear, analysis step;

Δu_s is the value of the perturbation of a driven variable in the submodel during a linear perturbation step;

u_g and Δu_g are the corresponding values of the same (geometrically interpolated) variable in the global model;

$u_g\big|_0$ is the "base state" value of the variable during a linear perturbation step in the global analysis;

$u_s\big|_0$ is the "base state" value of the variable during a linear perturbation step in the submodel analysis;

$u_g|_i$ is the value of u_g at increment *i* of the global analysis step; and

$\Delta u_g|_i$ is the value of Δu_g at increment *i* of the global analysis step.

Mixing general and linear perturbation steps in shell-to-solid submodeling

Additional assumptions must be made for the shell-to-solid case when a general procedure on the global model drives a linear perturbation procedure on the submodel and vice versa. The assumptions depend on the geometric formulation used (linear or nonlinear) and on the procedure combination. For details and governing equations for these cases, see "Submodeling analysis," Section 2.15.1 of the Abaqus Theory Guide.

Initial conditions

The definition of initial conditions should be consistent between the global model and the submodel.

Boundary conditions

Boundary conditions (other than submodel boundary conditions) prescribed on the degrees of freedom that are driven will replace those prescribed using submodel boundary conditions. When this replacement occurs, Abaqus reports the change in the data file.

A node can be driven from the global model in some steps and have user-prescribed boundary conditions in other steps. In these cases all relevant boundary conditions must be respecified (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Any other boundary conditions that are applied in the submodel region should be imposed in the submodel analysis in the usual way. It is your responsibility to apply such prescribed boundary conditions to the submodel correctly so that they correspond to the loading of the global model.

Be careful with submodel boundary nodes that are also on planes of symmetry, where both forms of boundary conditions can be applied. It may be helpful in such cases to apply boundary conditions in a local coordinate system (see "Transformed coordinate systems," Section 2.1.5). The local coordinate system should be applied only to the boundary conditions that are intended to override the submodel boundary conditions, since the submodel boundary conditions are always output in the global coordinate directions by the global model.

Loads

Any loads that are applied in the submodel region must be imposed in the submodel analysis in the usual way. It is your responsibility to apply such loads to the submodel correctly so that they correspond to the loading of the global model. See "Applying loads: overview," Section 34.4.1, for an overview of the loads available in Abaqus.

Predefined fields

The following predefined fields can be specified in a submodeling analysis, as described in "Predefined fields," Section 34.6.1:

- Nodal temperatures can be specified. Any difference between the applied and initial temperatures will cause thermal strain if a thermal expansion coefficient is given for the material ("Thermal expansion," Section 26.1.2). The specified temperature also affects temperature-dependent material properties, if any.
- The values of user-defined field variables can be specified. These values affect only field-variable-dependent material properties, if any.

Abaqus interpolates solution variables onto the submodel driven nodes. It can also interpolate temperatures as field variables (see "Interpolating data between meshes" in "Predefined fields," Section 34.6.1, for details). Other predefined fields will not be interpolated to the nodes of the submodel; they must be available from the input data for all nodes of the submodel where they are required.

Abaqus/Standard provides multiple approaches for cases where a submodel thermal-stress analysis must be performed using temperature solutions from a global heat transfer analysis.

- Run a heat transfer analysis of the global model, and write the nodal temperatures to the results or output database file. Run a sequentially coupled thermal-stress analysis of the global model. The temperatures obtained from the results or output database file of the global heat transfer analysis are field variables in this case. If the mesh used in the thermal-stress analysis is different from the mesh in the heat transfer analysis, specify that Abaqus/Standard should interpolate the temperature field from the heat transfer analysis mesh to the thermal-stress analysis mesh. Run a thermal-stress analysis of the submodel using the results or output database file for the global thermal-stress analysis to read the driven variables (displacement field) and using the results or output database file from either the global heat transfer analysis or the global thermal-stress analysis to read the temperatures as field variables. In either case the temperature field will have to be interpolated to the current submodel nodes. If interpolation between dissimilar meshes is necessary, the global output database file must be used to read the temperatures. For details, see Figure 10.2.2–13 and Figure 10.2.2–14.

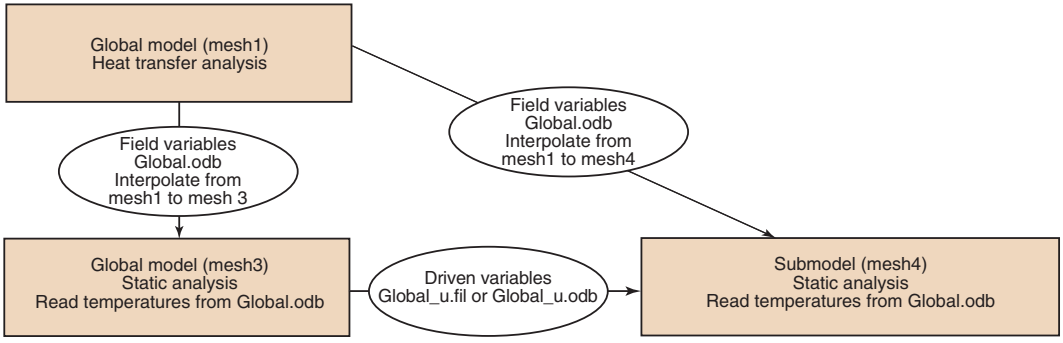


Figure 10.2.2–13 Sequentially coupled thermal-stress analysis for the global model with only a thermal-stress analysis for the submodel.

- Run a heat transfer analysis of the global model, and write the nodal temperatures to the results or output database file. Run a sequentially coupled thermal-stress analysis (the global thermal-stress

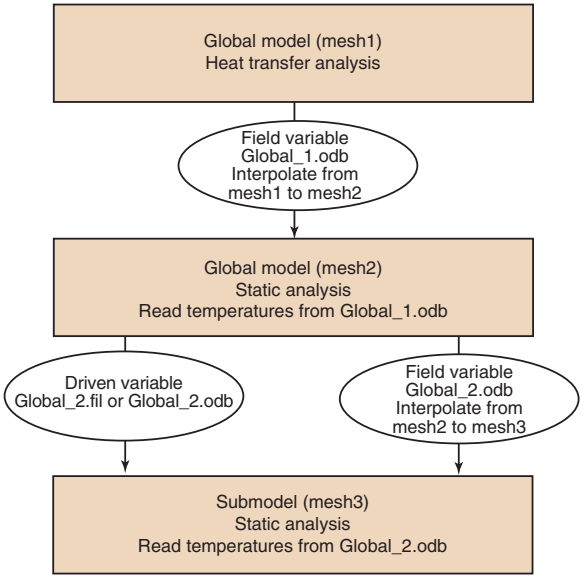


Figure 10.2.2–14 Sequentially coupled thermal-stress analysis for the global model with only a thermal-stress analysis for the submodel.

analysis) using the same mesh (mesh1) as the global heat transfer analysis and the temperatures from the results or output database file for the global heat transfer analysis. Next, run a submodel heat transfer analysis using the mesh (mesh2) that is required for the final submodel thermal-stress analysis, and write the nodal temperatures to the results or output database file. Use the temperature solution from the global heat transfer analysis to drive the solution of the submodel heat transfer analysis. Finally, run the submodel thermal-stress analysis using the temperatures (as field variables) obtained from the results or output database file for the submodel heat transfer analysis and the displacements (as driven variables) obtained from the global thermal-stress analysis. See the detailed flow chart in Figure 10.2.2–15.

Material options

Any of the material models described in Part V, "Materials," can be used in the global and submodel analyses. The material response defined for the submodel may be different from that defined for the global model.

Elements

The dimensionality of the submodel must be the same as that of the global model: both models must be either two-dimensional or three-dimensional. The following limitations apply:

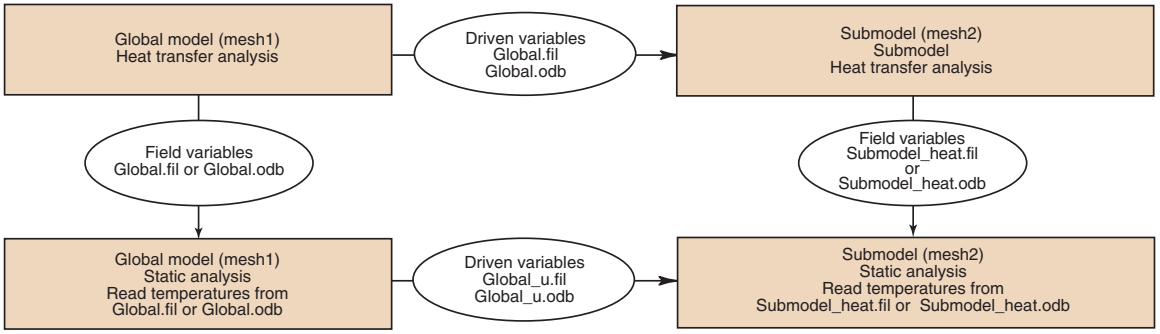


Figure 10.2.2–15 Sequentially coupled thermal-stress analysis for both the global model and submodel.

- The boundary nodes of the submodel must lie within regions of the global model where Abaqus is able to perform spatial interpolation to define the values of the driven variables. Therefore, they must lie within (or, as allowed by the exterior tolerance, near to) two- or three-dimensional geometrically defined elements in the global model. Such geometrically defined elements are:
  - first- or second-order triangles or quadrilaterals in two dimensions;
  - first- or second-order triangular or quadrilateral shells; and
  - first- or second-order tetrahedra, wedges, or bricks in three dimensions.
- When shell elements with five degrees of freedom per node (S4R5, S8R5, STRI65, etc.) are used in the global model, the rotations are not written to the results file or the output database; therefore, only the displacement degrees of freedom can be driven. This restriction suggests that submodeling should not be used with these elements or that the submodel should include a set of narrow elements around its driven edges so that the interpolated displacements at these nodes effectively transfer the rotation. Five degree of freedom shells cannot be used in shell-to-solid submodeling.
- The boundary nodes cannot lie in regions of the global model where there are only one-dimensional elements (beams, trusses, links, axisymmetric shells) since Abaqus does not provide the necessary interpolation of results for such elements.
- The boundary nodes cannot lie in regions of the global model where there are only user elements, substructures, springs, dashpots, etc. since those element types do not allow for geometric interpolation.
- The boundary nodes cannot lie in regions of the global model where there are only axisymmetric solid elements with nonlinear, asymmetric deformation (CAXA elements). The submodeling capability is currently not supported for these elements.
- The reference node associated with generalized plane strain elements (CPEG) cannot be used as a driven boundary node in a submodeling analysis.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see "Abaqus/Standard output variable identifiers," Section 4.2.1, and "Abaqus/Explicit output variable identifiers," Section 4.2.2).

As described above, nodal output requests to the results file or output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

Input file template

Global analysis:

```
*HEADING
…
*STEP
Step 1
*STATIC (or *DYNAMIC, etc.)
Data line to define step time and control incrementation
…
*NODE FILE
List of solution variables to be used to drive the submodel
*OUTPUT, FIELD
*NODE OUTPUT
List of solution variables to be used to drive the submodel
*END STEP
```

Submodel analysis:

```
*HEADING
…
*SUBMODEL, EXTERIOR TOLERANCE=tolerance
List of all nodes to be driven
**
*STEP
*STATIC (or any other allowable procedure)
Data line to define step time (must be the same as the step time in the global analysis unless the
TIMESCALE parameter is used on the *BOUNDARY option) and control incrementation
…
*BOUNDARY, SUBMODEL, STEP=1
Data lines listing nodes and degrees of freedom to be driven in this step
…
*END STEP
```

10.2.3 SURFACE-BASED SUBMODELING

Products: Abaqus/Standard

References

- "Submodeling: overview," Section 10.2.1
- *SUBMODEL
- *DSLOAD
- Chapter 38, "Submodeling," of the Abaqus/CAE User's Guide

Overview

The surface-based submodeling technique:

- may not provide the same level of accuracy as node-based submodeling;
- should be used only when the node-based technique cannot provide adequate results;
- is limited to stress-based solid-to-solid submodeling for general static procedures (see "Static stress analysis," Section 6.2.2) in Abaqus/Standard;
- applies surface tractions to submodel surfaces based on a stress field interpolated from the global model; and
- can be combined with node-based submodeling of displacements (see "Node-based submodeling," Section 10.2.2).

Performing a surface-based submodeling analysis

Your submodel analysis is driven, either partly or completely, from the results obtained from a global model analysis. The results from the global model are interpolated onto the surfaces on the appropriate parts of the boundary of the submodel. Thus, the response at the boundary of the local region is defined by the solution for the global model. The driven surfaces and any loads applied to the local region determine the solution in the submodel.

Surface-based submodeling should be used only when the node-based technique cannot provide adequate results. For a comparison of the two submodeling techniques and recommendations for their application, refer to "Submodeling: overview," Section 10.2.1.

Saving the results from the global model

The results from the global analysis must be saved at all elements required for the interpolation of the driven variables to the boundary surface of the submodel. Only the output database (in ODB or SIM format) can be used for this purpose.

In each step of the global model whose solution will be used to drive the submodel, write the stress results to the output database (see "Output to the output database," Section 4.1.3).

Input File Usage: Use both of the following options:

*OUTPUT, FIELD
*ELEMENT OUTPUT

Abaqus/CAE Usage: Step module: **Output→Field Output Requests→Create**

Referring to the global model results from the submodel analysis

You must define the source of the global solution results and provide the name of the output database file (in ODB or SIM format); the file extension is optional. If the file extension is omitted, Abaqus will use in order, the ODB output database file or the SIM database file.

Input File Usage: **abaqus job**=*submodel_input_file* **globalmodel**= *global_output_database* or *sim_database_file*

Abaqus/CAE Usage: Any module: **Model→Edit Attributes→*submodel***: **Submodel**: **Read data from job**: *global_output_database*

Reading data from a SIM database file is not supported in Abaqus/CAE.

Specifying the driven surfaces in the submodel

Specifying the driven element-based surfaces does not activate the driven surface loads: they must be activated by specifying the appropriate submodel distributed surface loads.

All surface facets of the submodel to be driven by stresses in any step must be specified as driven surfaces since the list of surfaces cannot be extended subsequent to its initial definition (even at restart). However, variables at the surfaces given do not have to be driven in all steps: the choice of which surfaces are driven in a particular step is made as part of a submodel distributed surface load definition, as discussed in "Defining the driven surface tractions in the submodel," later in this section.

Input File Usage: *SUBMODEL
list of element-based structural surfaces

The *SUBMODEL option must be included in the model definition portion of the input file for the submodel analysis. Multiple *SUBMODEL options are allowed; however, in this case you must ensure that the driven surfaces specified on the data line of one option are separate and distinct from the other surfaces specified on the data lines of all the other options.

Abaqus/CAE Usage: Load module: **Create Load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: **Driving region**: select region

Defining geometric tolerances

A geometric tolerance is used to define how far driven element-based surface nodes in the submodel can lie outside the exterior surface of the global model, as that surface is interpolated in the global, undeformed finite element model. By default, surface nodes in the submodel must lie within a distance calculated by multiplying the average element size in the global model by 0.05. You can change the tolerance, which is useful in cases where submodel driven surfaces lie to a greater extent outside

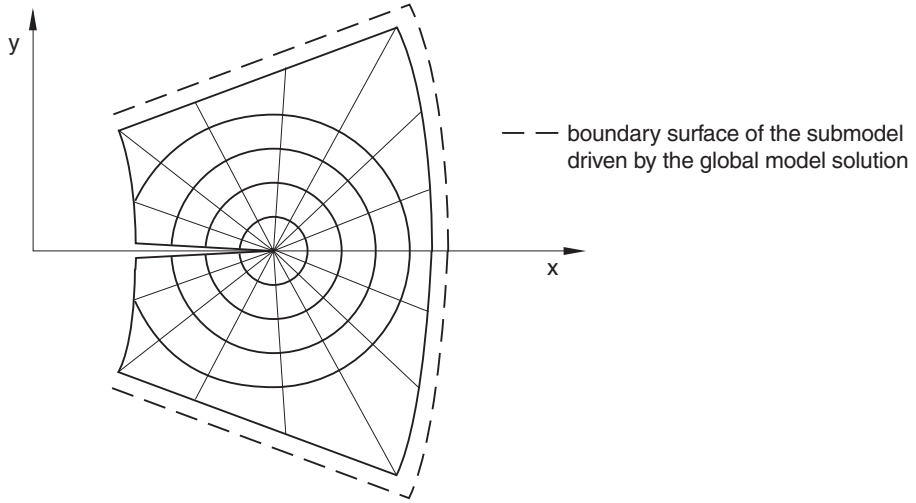


Figure 10.2.3–1 The magnified submodel.

the global model exterior surface. Tolerances larger than this default value, however, can result in significantly greater computation times and lower accuracy in the driven solution for driven surface regions significantly outside the global model exterior surface.

You can define the geometric tolerance as a fraction of the size of the average element in the global model or as an absolute distance in the length units chosen for the model. If both tolerances are defined, Abaqus uses the tighter tolerance.

Input File Usage: Use the following option to define the geometric tolerance as an absolute distance:

∗SUBMODEL, TYPE=SURFACE, ABSOLUTE EXTERIOR TOLERANCE=*tolerance*

Use the following option to define the geometric tolerance as a fraction of the size of the average element in the global model:

∗SUBMODEL, TYPE=SURFACE, EXTERIOR TOLERANCE=*tolerance*

Abaqus/CAE Usage: Load module: **Create Load** choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Exterior tolerance: absolute:** or **relative:** *tolerance*

The exterior tolerance in solid-to-solid submodeling

The exterior tolerance for a solid-to-solid submodel analysis is indicated by the shaded region in Figure 10.2.3–2. If the distance between the driven surface nodes and the free surface of the global model falls within the specified tolerance, the solution variables from the global model are extrapolated to the submodel.

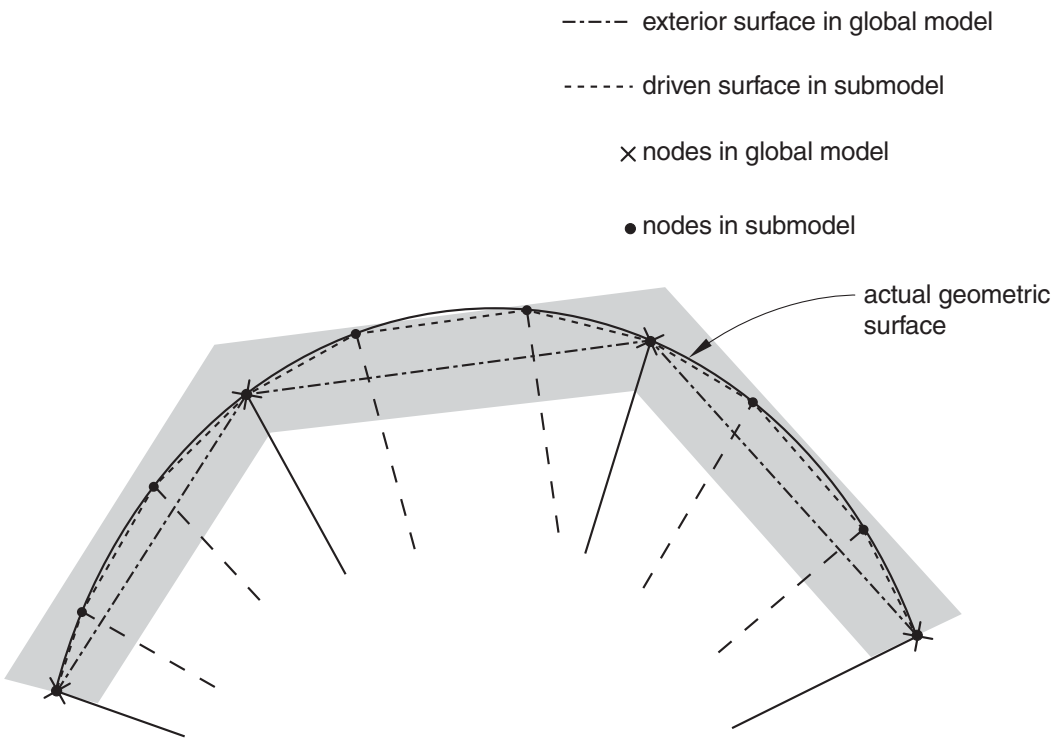


Figure 10.2.3–2 The exterior tolerance in surface-based submodeling.

Defining the driven surface tractions in the submodel

The actual driven surface tractions are defined in any step as submodel distributed surface loads. The stresses resulting in these tractions are "driven variables" obtained from the output database file of the global analysis.

All stress components from the global model elements that will drive the submodel boundary surface must have been written to the output database. They will be used to create traction, shear, and normal stresses at integration points of driven surfaces (as non-uniform distributed surface loads). All applicable stress components are calculated and applied to the surface integration points at each time increment.

Input File Usage: *DSLOAD, SUBMODEL

Abaqus/CAE Usage: Load module: **Create load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**

Specifying the step number from the global analysis

You specify the step of the global model history that is to be used for the driven variables in the current submodel analysis step.

Input File Usage: *DSLOAD, SUBMODEL, STEP=*step*

Abaqus/CAE Usage: Load module: **Create load**: choose **Other** for the **Category** and **Submodel** for the **Types for Selected Step**: select region: **Global step number:** *step*

Modifying the set of driven surface tractions

You can modify the submodel distributed surface load definitions from step to step to change the global step reference, you can remove surface load definitions, and you can reintroduce them later (see "Applying loads: overview," Section 34.4.1). New surfaces cannot be added to the total set of driven surface defined for the submodel; this set of driven surfaces is a fixed part of the model definition.

Input File Usage: Use one of the following options:

*DSLOAD, SUBMODEL, OP=MOD
*DSLOAD, SUBMODEL, OP=NEW

Guidelines for obtaining adequate solution accuracy

Unlike node-based submodeling, surface-based submodeling can in many cases provide incorrect or misleading submodel results. This risk follows from the methods used to interpolate stresses from the global model to the submodel:

- The global model material point stresses are smoothed and associated with the global model nodes.
- These global model node-located stresses are then interpolated to the submodel surface integration points and applied as tractions.

This process is generally nonconservative, resulting in a submodel traction field that is not equivalent to the global model stress field in an equilibrium sense.

Modeling guidelines

You can improve accuracy and achieve reasonable submodel solutions by observing the following guidelines:

- Design your models so that your submodel surface intersects the global model in regions of relatively low stress gradients.
- Design your models so that your submodel surface intersects the global model in regions of uniform element size. A warning message is provided in the data (`.dat`) file in cases where significant nonuniform element size distributions are seen.

Checking your results

To understand whether your modeling approach results in a reasonably accurate solution, the following guidelines are recommended:

- Compare the stress distributions on the submodel-driven surfaces with the stress distributions in the global model. You can view the stress distributions in the global model by using tools such as cutting planes and path plots in the Visualization module of Abaqus/CAE. The degree to which the global model's stress distributions agree with those in the submodel-driven surface is generally an indication of the level of accuracy of your submodel solution.
- When using inertia relief in the submodel for cases where submodeling does not remove all rigid body modes, compare the inertia relief forces to the prevailing force level in your submodel. If the inertia relief force is large compared to the prevailing force level, your submodel results may be inaccurate.

Special considerations

There are several special considerations that are worth noting.

Handling of rigid-body modes

When you use surface-based submodeling exclusively to drive your submodel response, your displacement solution will not be unique; you will generally encounter rigid-body modes and accompanying numerical issues. You can address these rigid-body modes by

- providing sufficient node-based submodel displacement boundary condition definitions in the submodel analysis,
- providing sufficient boundary condition definitions in the submodel analysis, or
- providing an inertia relief load definition in the submodel analysis (see "Inertia relief," Section 11.1.1).

You can combine these definitions, as necessary and appropriate to your model, to address all rigid body modes.

Cases of finite rotation

Global model stress results are stored in the output database in the global coordinate system. Submodel tractions are calculated from these stresses and the current configuration surface normal in the submodel. Hence, when your global model result involves significant finite rotation, your submodel results will generally be inaccurate unless you provide sufficient node-based submodel displacement boundary condition definitions to impart similar rigid-body rotations to the submodel; exclusive use of surface-based submodeling definitions is not adequate to provide these rigid-body motions. You may also experience convergence difficulties in the submodel when it is not properly rotated.

Inelastic behavior

When surface-based submodeling is used to drive a submodel region with an inelastic material definition, you may encounter rigid-body modes and accompanying numerical issues. For example, numerical issues will prevent convergence if the submodel material definition includes plasticity and the submodel loading results in a shear band formation beyond the material hardening definition, such

that unconstrained motion can occur (i.e., if the submodel loads exceed the limit load capacity). In these cases node-based submodeling should be used.

Procedures

Only the static procedure is allowed. Both general (possibly nonlinear) and linear perturbation steps can be used in submodeling (see "General and linear perturbation procedures," Section 6.1.3, for a discussion of general and linear perturbation steps).

Obtaining a solution at a particular point in time using linear perturbation analysis

In Abaqus/Standard it is possible to study the submodel's linearized response corresponding to a particular point in time in the global solution by using a static, linear perturbation procedure in the submodel analysis. You can select the increment in the global analysis step that is to be used as the basis for calculating the values for the driven variables. If you do not select an increment in a static linear perturbation step, the last increment of the selected step in the global analysis is used as the basis for calculating the values for the driven variables. You cannot select an increment in a general submodel step.

Input File Usage: *DSLOAD, SUBMODEL, STEP=*step*, INC=*increment*

Abaqus/CAE Usage: Selection of a specific global model increment is not supported in Abaqus/CAE.

Mixing general and linear perturbation steps

It is possible to mix general steps and linear perturbation steps in both the global and the submodel analyses. Abaqus allows general analysis steps to be treated as linear perturbation steps during submodeling, and vice versa.

Example: Submodeling with general and linear perturbation steps

For an example of submodeling that uses both general and linear perturbation steps, consider the following situation. The global analysis consists of a static preload—done as a general, nonlinear, analysis step—followed by extraction of the eigenmodes of the preloaded structure, then a step of 5 seconds of modal dynamic response analysis:

```
*STEP
** Apply preload
*STATIC
 0.1, 1.0
…
** Write out stress results for elements needed to
** interpolate to the submodel's surfaces
*ELEMENT OUTPUT, ELSET=DETAIL
 S
*END STEP
*STEP
```

```
** Calculate modes and frequencies
*FREQUENCY
…
** The *ELEMENT OUTPUT option is repeated because
** this is the first linear perturbation step
*ELEMENT OUTPUT, ELSET=DETAIL
 U
*END STEP
*STEP
** Dynamic response of preloaded system
*MODAL DYNAMIC
 0.01, 5.0
…
*END STEP
```

We wish to study the local, possibly nonlinear, response of a part of this model that is so small that we do not need to model dynamic effects locally and can, thus, perform two steps of static analysis:

```
** Define submodel surfaces (driven surfaces)
*SUBMODEL,TYPE=SURFACE
PERIM
*STEP
** Preload
*STATIC
 0.1, 1.0
*DSLOAD, SUBMODEL, STEP=1
…
*END STEP
*STEP
** Local static response to global dynamic step
*STATIC
 0.01, 5.0
*DSLOAD, SUBMODEL, STEP=3
…
*END STEP
```

It is perfectly acceptable that the submodel analysis requests general, possibly nonlinear, analysis for both steps, while in the global analysis the dynamic step was a linear perturbation step (modal dynamics is always a linear perturbation analysis). It is your responsibility to judge that this use of the submodeling feature is reasonable. For example, suppose that the global analysis were continued with a fourth step of general, nonlinear static response:

```
*RESTART, READ, STEP=3
** Read state at end of initial preload
```

```
** (could equally well use *RESTART, READ, STEP=1)
*STEP
** Add more preload
*STATIC
 0.2, 1.0
…
*END STEP
```

This fourth general analysis step starts with the state at the end of general analysis Step 1 because the frequency extraction and the modal dynamic steps are both linear perturbation steps. However, if we restart the submodel analysis in the same way, the solution may not be comparable with the global model solution:

```
*RESTART, READ, STEP=2
** Read state at end of step 2
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*DSLOAD, SUBMODEL, STEP=4
…
*END STEP
```

The second step in the submodel is a general analysis step, to which the response may be nonlinear, thus changing the state of the model. A valid alternative would be to apply the Step 4 response to the submodel immediately after the first step:

```
*RESTART, READ, STEP=1
** Read state at end of preload step
*STEP
** Add more preload
*STATIC
 0.2, 1.0
*DSLOAD, SUBMODEL, STEP=4
…
*END STEP
```

Loads

Any loads that are applied in the submodel region of the global analysis must be imposed in the submodel analysis in the usual way. It is your responsibility to apply such loads to the submodel correctly so that they correspond to the loading of the global model. See “Applying loads: overview,” Section 34.4.1, for an overview of the loads available in Abaqus.

Output

Any of the output normally available within a particular procedure is also available during a submodeling analysis (see "Abaqus/Standard output variable identifiers," Section 4.2.1, and "Abaqus/Explicit output variable identifiers," Section 4.2.2).

As described above, element stress output requests to the output database file must be used in the global analysis to save the values of the driven variables at the submodel boundary.

Input file template

Global analysis:

```
*HEADING
…
*STEP
Step 1
*STATIC (or *STATIC, etc.)
Data line to define step time and control incrementation
…
*ELEMENT OUTPUT
S
*OUTPUT, FIELD
*ELEMENT OUTPUT
S
*END STEP
```

Submodel analysis:

```
*HEADING
…
*SUBMODEL,TYPE=SURFACE, EXTERIOR TOLERANCE=tolerance
List of all surfaces to be driven
**
*STEP
*STATIC (or any other allowable procedure)
Data line to define step time and control incrementation.
…*DSLOAD, SUBMODEL, STEP=1
Data lines listing surfaces to be driven in this step
…
*END STEP
```

10.3 Generating matrices

- “Generating matrices,” Section 10.3.1
- “Generating thermal matrices,” Section 10.3.2

10.3.1 GENERATING MATRICES

Product: Abaqus/Standard

References

- "Defining matrices," Section 2.11.1
- "Element matrix assembly utility," Section 3.2.28
- "Defining an analysis," Section 6.1.2
- *MATRIX GENERATE
- *MATRIX CHECK
- *MATRIX OUTPUT
- *MATRIX INPUT
- *CLOAD

Overview

Matrix generation:

- is a linear perturbation procedure;
- allows for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the stiffness, mass, viscous damping, structural damping, and load vectors in a model;
- generally creates matrices identical to those used in a subspace-based steady-state dynamic procedure (see "Subspace-based steady-state dynamic analysis," Section 6.3.9).
- includes initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the analysis;
- writes matrix data to a binary **`.sim`** file that can be read as input by Abaqus;
- can output matrix data to text files that can be read as input in other analyses by Abaqus or other simulation software; and
- can check the quality of the generated global stiffness and mass matrices.

Introduction

A linearized finite element model can be summarized in terms of matrices representing the stiffness, mass, damping, and loads in the model. Using these matrices, you can exchange model data between other users, vendors, or software packages without exchanging mesh or material data. Matrix representations of a model prevent the transfer of proprietary information and minimize the need for data manipulation.

The matrix generation procedure is a linear perturbation step (see "General and linear perturbation procedures," Section 6.1.3) that accounts for all current boundary conditions, loads, and material

response in a model. You can also specify new boundary conditions, loads, and predefined fields within the matrix generation step. The generated matrices are input to a matrix usage model.

The matrix generation procedure uses SIM, which is a high-performance database available in Abaqus. The generated matrices are stored in a file named *jobname*`_X`*n*`.sim`, where *jobname* is the name of the input file or analysis job and *n* is the number of the Abaqus step that generates the matrices.

Specifying the matrix type

You can generate matrices representing the following model features:

- stiffness,
- mass,
- viscous damping,
- structural damping, and
- loads.

The load matrix contains integrated nodal load vectors (right-hand sides) for the load cases defined in the matrix generation step. Load cases can be made up of any combination of loadings—distributed loads, concentrated nodal loads, thermal expansion, and load cases defined for any substructures that may be used as part of the model.

Input File Usage: Use the following option to generate the stiffness matrix:

*MATRIX GENERATE, STIFFNESS

Use the following option to generate the mass matrix:

*MATRIX GENERATE, MASS

Use the following option to generate the viscous damping matrix:

*MATRIX GENERATE, VISCOUS DAMPING

Use the following option to generate the structural damping matrix:

*MATRIX GENERATE, STRUCTURAL DAMPING

Use the following option to generate the load matrix:

*MATRIX GENERATE, LOAD

Generating element matrices

By default, the matrix generation procedure generates global matrices for a model in assembled form. The generated global matrices are assembled from the local element matrices and include contributions from matrix input data. Abaqus/Standard offers an option to generate global matrices in element-by-element form. Instead of global (assembled) matrices, local element matrices are generated. If you choose to generate local element matrices for a model containing matrix input data, Abaqus/Standard calculates and stores only element matrices; the matrix input data are ignored.

Input File Usage: *MATRIX GENERATE, ELEMENT BY ELEMENT

Generating matrices for a part of the model

You can generate matrices for a part of the model defined by an element set, the source contributions, or the physical fields.

Defining the element set from which to generate matrices

By default, the matrix generation procedure generates matrices for a whole model. Abaqus/Standard can generate matrices for a part of the model defined by an element set.

Input File Usage: *MATRIX GENERATE, ELSET=*element set name*

Defining the source contributions from which to generate matrices

By default, the matrix generation procedure generates matrices from the finite elements and from the matrix input data used in the finite element model. Abaqus/Standard can generate matrices for a part of the model defined by the finite elements only or by the matrix input contributions only.

Input File Usage: Use the following option to generate matrices from the finite elements and from the matrix input data:

*MATRIX GENERATE, SOURCE=ALL (default)

Use the following option to generate matrices including only contributions from the finite elements:

*MATRIX GENERATE, SOURCE=ELEMENTS

Use the following option to generate matrices including only contributions from the matrix input data:

*MATRIX GENERATE, SOURCE=MATRIX INPUT

Defining the physical fields from which to generate matrices

You can also define a part of the model for the matrix generation by the physical field. The matrix generation procedure can generate matrices for mechanical and acoustic fields. By default, the matrix generation procedure generates matrices for the structural and acoustic parts of the model. You can generate matrices for only the structural part or for only the acoustic part of the model. Abaqus/Standard does not support generated matrices containing acoustic degrees of freedom in the matrix usage model (see “Defining matrices,” Section 2.11.1).

Input File Usage: Use the following option to generate matrices for the structural and acoustic parts of the model:

*MATRIX GENERATE, FIELD=ALL (default)

Use the following option to generate matrices for only the structural part of the model:

*MATRIX GENERATE, FIELD=MECHANICAL

Use the following option to generate matrices for only the acoustic part of the model:

*MATRIX GENERATE, FIELD=ACOUSTIC

Evaluating frequency-dependent material properties

When frequency-dependent material properties are specified in the model definition, Abaqus/Standard offers the option of choosing the frequency at which these properties are evaluated for use in global matrix generation. If you do not choose the frequency, Abaqus/Standard evaluates the matrices at zero frequency and does not consider the contributions from frequency-domain viscoelasticity.

Input File Usage: *MATRIX GENERATE, PROPERTY EVALUATION=*frequency*

Applying multipoint constraints

If multipoint constraints are used in the model, by default, the generated matrices include only entries for the independent degrees of freedom. You can skip applying the multipoint constraints in the generated model.

Input File Usage: Use the following option to generate the matrices with applied multipoint constraints:

*MATRIX GENERATE, MPC=YES (default)

Use the following option to skip the multipoint constraints during the matrix generation:

*MATRIX GENERATE, MPC=NO

Specifying public nodes

An Abaqus/Standard model may contain user-defined nodes and internal nodes. Internal nodes are nodes with internal degrees of freedom associated with them (for example, Lagrange multipliers and generalized displacements) that are created internally by Abaqus/Standard.

You can use the matrix generation procedure to specify some of the user-defined nodes as "public nodes." These nodes will be visible in the matrix usage model. The remaining user-defined nodes in the matrix data are designated as internal nodes and are effectively hidden in the matrix usage model (see "Internal nodes in matrix data" in "Defining matrices," Section 2.11.1). By default, all user-defined nodes in the matrix data are public nodes.

By specifying public nodes, you can reduce the number of user-defined nodes in the matrix usage analysis, which simplifies the remapping process (see "Remapping user-defined nodes in assembled matrices" in "Defining matrices," Section 2.11.1). For example, you may want to identify the nodes for attaching a subcomponent (matrix) to the matrix usage model or the nodes for the output of results as public nodes.

Input File Usage: *MATRIX GENERATE, PUBLIC NODES=*node set name*

Checking generated matrices

In a matrix generation analysis, you can check the quality of the generated global stiffness and mass matrices. The matrix check generates six "artificial" rigid body modes and projects the matrices onto the rigid body modal subspace. It is expected that the projected 6×6 stiffness matrix (also known as the rigid body energy matrix) is close to zero in the absence of the boundary conditions and constraints. To perform the stiffness matrix quality check, the boundary conditions and multipoint constraints can be suppressed in the matrix generation procedure. The total inertia statistics for the model are extracted from the projected 6×6 rigid body mass matrix. You can specify the center of rotation for creating the artificial rotational rigid body modes and calculating the global inertia tensor. If the matrix quality check is requested, the check results are printed in the data (`.dat`) file.

Input File Usage: Use the following option to request the matrix check:

∗MATRIX CHECK

Use the following option to specify the center of rotation of the coordinate frame to be used for the matrix check:

∗MATRIX CHECK, REFERENCE NODE=*node_label*

Initial conditions

Matrix generation is a linear perturbation procedure. Therefore, the initial state for the matrix generation step is the state of the model at the end of the last general analysis step. The generated matrix includes initial stress and load stiffness effects due to preloads and initial conditions if nonlinear geometric effects are included in the analysis.

Boundary conditions

Boundary conditions can be defined or modified in a matrix generation step. For more information on defining boundary conditions, see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1. Any boundary conditions that are defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Loads

All types of loads can be applied in the load cases for a matrix generation step. Real and imaginary parts of the load vectors will be generated for the complex loads. For more information on applying loads, see "Applying loads: overview," Section 34.4.1. Any loads that are defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Predefined fields

All types of predefined fields can be specified in a matrix generation procedure. For more information on specifying predefined fields, see "Predefined fields," Section 34.6.1. Any predefined fields that are

defined in a matrix generation step will not be used in subsequent general analysis steps (unless they are respecified).

Material options

All types of materials that are available in Abaqus/Standard can be used in a matrix generation procedure.

Elements

All types of elements that are available in Abaqus/Standard can be used in the matrix generation procedure.

Generating matrices for models containing solid continuum infinite elements

Solid continuum infinite elements (CIN-type elements) have different formulations in static and dynamic analyses. Therefore, when generating matrices for a model containing solid continuum infinite elements, you must specify whether to use the static or dynamic formulation.

Input File Usage: Use the following option to select the static formulation for solid infinite elements:

∗MATRIX GENERATE, SOLID INFINITE FORMULATION=STATIC

Use the following option to select the dynamic formulation for solid infinite elements:

∗MATRIX GENERATE, SOLID INFINITE FORMULATION=DYNAMIC

Output

In a matrix generation analysis, you can output the stiffness, mass, viscous damping, structural damping, and load matrices to text files. Several formats are available for the matrix output, as discussed below. Matrices are copied from the **`.sim`** file and output in text files that use the following naming conventions:

jobname_matrixN`.mtx`

where *jobname* is the name of the input file or analysis job, *matrix* is a four-letter identifier indicating the matrix type (as outlined in Table 10.3.1–1), and *N* is the number associated with the Abaqus analysis step generating the matrices. For example, if a stiffness matrix generation is performed in the third step of an analysis job named **`VehicleFrame`**, the matrix is output to a file named **`VehicleFrame_STIF3.mtx`**.

Alternatively, the matrices can be output into a Nastran DMIG-s format file that uses the following naming convention:

jobname`_X`*N*`.bdf`

where *jobname* is the name of the input file or analysis job and *N* is the number associated with the Abaqus analysis step generating the matrices. For example, if the matrix generation is performed

Table 10.3.1–1 Identifiers used in the generated matrix file name.

Identifier	Matrix Type
STIF	Stiffness matrix
MASS	Mass matrix
DMPV	Viscous damping matrix
DMPS	Structural damping matrix
LOAD	Load matrix

in the third step of an analysis job named **VehicleFrame**, the matrices are output to a file named **VehicleFrame_X3.bdf**.

You can use the matrix assembly utility ("Element matrix assembly utility," Section 3.2.28) to assemble element matrices in the SIM document and/or write assembled matrices to text files.

Input File Usage: Use the following option to output the stiffness matrix:

*MATRIX OUTPUT, STIFFNESS

Use the following option to output the mass matrix:

*MATRIX OUTPUT, MASS

Use the following option to output the viscous damping matrix:

*MATRIX OUTPUT, VISCOUS DAMPING

Use the following option to output the structural damping matrix:

*MATRIX OUTPUT, STRUCTURAL DAMPING

Use the following option to output the load matrix:

*MATRIX OUTPUT, LOAD

Matrix input text format

This default text format creates matrix files consistent with the format used by the matrix definition technique in Abaqus/Standard (see "Defining matrices," Section 2.11.1). This format does not convert any of the internal Abaqus node labels. A negative number or zero can be used as a label for an internal node.

Input File Usage: *MATRIX OUTPUT, FORMAT=MATRIX INPUT

Format of the operator matrix

The assembled sparse matrix operator data are written to the text file as a series of comma-separated lists. Each row in the file represents a single matrix entry; a row is written as a comma-separated list with the following elements:

1. Row node label
2. Degree of freedom for row node

3. Column node label
4. Degree of freedom for column node
5. Matrix entry

Format of the load matrix

Nonzero entries in load matrices, which represent right-hand-side vector data, are written to the text file as a comma-separated list with the following elements:

1. Node label
2. Degree of freedom
3. Right-hand-side vector entry

The format of the load vectors and heading labels is based on the Abaqus keyword interface, which allows the generated loads to be easily applied in other Abaqus analyses. Each load vector uses the following headings to indicate the real and imaginary portions of the load:

```
*CLOAD, REAL
*CLOAD, IMAGINARY
```

If the matrix generation step has multiple load cases, the load matrices for each load case are wrapped by the following labels in the generated text file:

```
*LOAD CASE
…
*END LOAD CASE
```

Including generated matrix data and generated loads in another Abaqus model

The generated sparse matrix data and generated loads that are output in matrix input text format can be included in another Abaqus model.

Input File Usage: Use the following options:

∗MATRIX INPUT
∗INCLUDE, INPUT=*matrix or load file*

Labeling text format

You can generate text files in which the matrix is formatted according to the standard labeling format. Internal Abaqus node labels are converted into large positive numbers that are acceptable for Abaqus matrix input data. This is the only difference between the labeling text format and the default matrix input text format. All nodes of the matrix generated in the labeling text format are treated as user-defined nodes if the matrix is used in an Abaqus/Standard analysis.

Input File Usage: ∗MATRIX OUTPUT, FORMAT=LABELS

Coordinate text format

You can generate text files in which the matrix is formatted according to the common mathematical coordinate format. This format is commonly used in mathematics programs such as MATLAB.

Each row in a coordinate-formatted file corresponds to a matrix entry; a row is written as a comma-separated list with the following elements:

1. Row number
2. Column number
3. Matrix entry

For load matrices, which represent right-hand-side vector data, each row in the text file is written with the following elements:

1. Equation (row) number
2. Right-hand-side vector entry

Commented load case options are written to the output file to indicate the load cases.

Input File Usage: *MATRIX OUTPUT, FORMAT=COORDINATE

DMIG format

You can output the generated matrices into a Nastran DMIG-s format (**`.bdf`**) file. See the Nastran documentation for details.

Input File Usage: *MATRIX OUTPUT, FORMAT=DMIG

Outputting matrices in element-by-element form

If matrices are generated in element-by-element form, you can write them in element-by-element form. When you generate text files using the matrix input or labeling format, each row in the file represents a single matrix entry; a row is written as a comma-separated list with the following elements:

1. Element label
2. Row node label
3. Degree of freedom for row node
4. Column node label
5. Degree of freedom for column node
6. Matrix entry

For load matrices, which represent right-hand-side vector data, each row in the text file is written with the following elements:

1. Element label
2. Node label
3. Degree of freedom
4. Right-hand-side vector entry

Coordinate format is also available for element-by-element global matrix generation. Each row in a coordinate-formatted file corresponds to a matrix entry; a row is written as a comma-separated list with the following elements:

1. Element number
2. Row number
3. Column number
4. Matrix entry

For load matrices each row in the text file is written with the following elements:

1. Element number
2. Equation (row) number
3. Right-hand-side vector entry

Input file template

```
*HEADING
…
**
*STEP
Options to define the preloading history for the model.
*END STEP
**
*STEP
*MATRIX GENERATE, STIFFNESS, MASS, VISCOUS DAMPING,
STRUCTURAL DAMPING, LOAD
*MATRIX OUTPUT, STIFFNESS, MASS, VISCOUS DAMPING,
STRUCTURAL DAMPING, LOAD, FORMAT=MATRIX INPUT
*BOUNDARY
Options to define the boundary conditions for the matrix generation step.
**
*LOAD CASE, NAME=LC1
Options to define the loading for the first load case.
*END LOAD CASE
**
*LOAD CASE, NAME=LC2
Options to define the loading for the second load case.
*END LOAD CASE
Any number of load cases can be defined.
*END STEP
```

10.3.2 GENERATING THERMAL MATRICES

Product: Abaqus/Standard

References

- "Element matrix assembly utility," Section 3.2.28
- *ELEMENT OPERATOR OUTPUT

Overview

Thermal matrix generation:

- allows for the mathematical abstraction of model data such as mesh and material information by generating global or element matrices representing the thermal conductivity, heat capacity, and load at specific times;
- writes matrix data to a binary SIM document that can be further processed by Abaqus; and
- can be used only as part of an uncoupled heat transfer analysis.

Introduction

A linearized heat transfer finite element model can be summarized in terms of the thermal load vector and thermal matrices that represent the heat capacity and the thermal conductivity. This mathematical abstraction serves various purposes. For example, you can use these matrices to exchange model data with other users, vendors, or software packages without exchanging mesh or material information. You can also use these matrices in techniques such as model order reduction. This abstraction can also be extended to transient nonlinear problems, which can be treated as a series of piecewise linear models constructed from thermal matrix data at discrete times.

Thermal matrix generation occurs in a heat transfer analysis and accounts for all the current boundary conditions, loads, and material response in the model. The generated matrices are stored in a SIM document named *jobname*`THERM`*n*`.sim`, where *jobname* is the name of the input file or analysis job and *n* is the number of the Abaqus heat transfer step that generates the matrices.

Defining matrix types

The continuous time description of the spatially discretized heat transfer equation (see "Uncoupled heat transfer analysis," Section 2.11.1 of the Abaqus Theory Guide) is

$$\int_V N^N \rho \dot{U}\, dV = -\int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV + \int_V N^N r\, dV + \int_{S_q} N^N q\, dS,$$

where θ is the temperature field, N^N are the finite element interpolation functions, ρ is the material density, $\dot{U}$ is the material time derivative of the internal energy, $\mathbf{k}$ is the (possibly anisotropic) conductivity matrix, r is the prescribed heat flux per unit volume, V is the volume of the domain, and S_q is the surface on which heat flux per unit area q is either directly prescribed or specified through film and radiation conditions.

The external flux vector $\mathbf{P}$ is defined as

$$P^N = \int_V N^N r \, dV + \int_{S_q} N^N q \, dS.$$

The internal flux vector $\mathbf{I}$ is defined as

$$I^N = - \int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV.$$

The net flux vector $\mathbf{F}$ is defined as the sum of the internal flux vector $\mathbf{I}$ and the external flux vector $\mathbf{P}$. The heat capacity matrix $\mathbf{C}$ is defined as

$$C^{NM} = \int_V N^N \rho \frac{\partial U}{\partial \theta} N^M dV.$$

The thermal conductivity matrix $\mathbf{K}$ is defined as

$$K^{NM} = - \frac{\partial}{\partial \theta^M} \left[- \int_V \frac{\partial N^N}{\partial \mathbf{x}} \cdot \mathbf{k} \cdot \frac{\partial \theta}{\partial \mathbf{x}} dV + \int_V N^N r \, dV + \int_{S_q} N^N q \, dS \right].$$

That is, the thermal conductivity matrix is the negated derivative of the net flux vector with respect to the nodal temperature vector $\boldsymbol{\theta}$ and, hence, includes the effect of temperature-dependent flux conditions such as film and radiation.

Specifying the matrix type

You can generate thermal matrices representing the following model features:

- heat capacity
- thermal conductivity
- loads

The thermal conductivity matrix has an unsymmetric contribution if the thermal conductivity property is temperature dependent. This term is taken into account only if the unsymmetric solver has been activated in the step definition (see "Defining an analysis," Section 6.1.2).

The load matrix contains either the nodal external flux vector or the net flux vector corresponding to the loading defined in the heat transfer step.

Input File Usage: Use the following option to generate the heat capacity matrix:

*ELEMENT OPERATOR OUTPUT, DAMPING

Use the following option to generate the thermal conductivity matrix:

*ELEMENT OPERATOR OUTPUT, STIFFNESS

Use the following option to generate the external flux vector:

*ELEMENT OPERATOR OUTPUT, LOAD, LOADTYPE=EXTERNAL

Use the following option to generate the net flux vector:

*ELEMENT OPERATOR OUTPUT, LOAD, LOADTYPE=NET

Generating matrices for a part of the model

By default, thermal matrices are generated for all supported element types in the model. You can request that Abaqus/Standard generate matrices for a part of the model defined by an element set.

Input File Usage: *ELEMENT OPERATOR OUTPUT, ELSET=*element set name*

Specifying the frequency of matrix generation

By default, thermal matrices are generated for every increment in the step in which it is requested. You can request that Abaqus/Standard generate matrices at a specified frequency.

Input File Usage: *ELEMENT OPERATOR OUTPUT, FREQUENCY=*output frequency*

Generating assembled matrices

By default, thermal matrices are written to the SIM document in element-by-element form. You can write assembled matrices to the SIM document, which is recommended when thermal matrix output is requested for large element sets or at frequent intervals.

Input File Usage: *ELEMENT OPERATOR OUTPUT, ASSEMBLE

Initial conditions

Thermal matrix generation occurs in a general analysis procedure. Therefore, the generated matrices include the effect of initial conditions in transient analyses.

Boundary conditions

Prescribed temperature boundary conditions are not imposed on the generated thermal matrices and load vector.

Loads

All types of loads supported in an uncoupled heat transfer analysis can be used for thermal matrix generation. For more information on applying loads, see “Applying loads: overview,” Section 34.4.1. Load types that are functions of temperature (such as film and radiation) contribute additional “load stiffness” terms to the thermal conductivity matrix.

Predefined fields

All types of predefined fields can be specified for thermal matrix generation. For more information on specifying predefined fields, see "Predefined fields," Section 34.6.1.

Material options

All types of materials that are available in Abaqus/Standard for uncoupled heat transfer can be used for thermal matrix generation.

Elements

Only continuum diffusive heat transfer elements and thermal contact elements are supported for thermal matrix generation. Thermal matrices are written to the SIM document only for supported elements.

Output

The generated matrices are written to the output SIM document in either element-by-element or assembled form. For efficiency, only nonzero matrix entries are stored in the SIM document. If the matrix is symmetric, only the nonzero entries in the upper triangular portion of the matrix are stored. You can use the matrix assembly utility ("Element matrix assembly utility," Section 3.2.28) to assemble element matrices in the SIM document and/or write assembled matrices to text files.

Limitations

Constraints that are implemented using the degree-of-freedom elimination technique (such as tie constraints) are not processed for thermal matrix output. In addition, cavity radiation effects are not considered for thermal matrix output.

Input file template

```
*HEADING
...
**
*STEP
Options to define an uncoupled heat transfer analysis.
...
*BOUNDARY
Options to define the boundary conditions for the heat transfer step.
**
*CFLUX and/or *DFLUX and/or *DSFLUX
Data lines to define thermal loading
*FILM and/or *SFILM and/or *RADIATE and/or *SRADIATE
Data lines to define convective film and radiation conditions
```

```
**
*ELEMENT OPERATOR OUTPUT, ASSEMBLE, STIFFNESS, DAMPING,
LOAD, LOADTYPE=EXTERNAL, FREQUENCY=1
**
```
Options to define the output requests for the heat transfer step.
```
**
*END STEP
```

10.4 Symmetric model generation, results transfer, and analysis of cyclic symmetry models

- "Symmetric model generation," Section 10.4.1
- "Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh," Section 10.4.2
- "Analysis of models that exhibit cyclic symmetry," Section 10.4.3

10.4.1 SYMMETRIC MODEL GENERATION

Product: Abaqus/Standard

Reference

- *SYMMETRIC MODEL GENERATION

Overview

A three-dimensional model can be created in Abaqus/Standard by:

- revolving an axisymmetric model about its axis of revolution;
- revolving a single three-dimensional sector about its axis of symmetry; or
- combining two parts of a symmetric three-dimensional model, where one of the parts is the original model and the other part is obtained by reflecting the original model through either a symmetry line or a symmetry plane.

Abaqus/Standard also provides for the transfer of the solution obtained in the original analysis onto the new model (see "Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh," Section 10.4.2).

Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be used to generate a new model.

Model generation

The symmetric model generation capability can be used to create a three-dimensional model by revolving an axisymmetric model about its axis of revolution, by revolving a single three-dimensional sector about its axis of symmetry, or by combining two parts of a symmetric model, where one part is the original model and the other part is the original model reflected through a line or a plane. The original model must have been saved to a restart file. The symmetric model generation capability is not available for models defined in terms of an assembly of part instances. Therefore, an element set name or a node set name containing quotation marks is not supported.

An entire three-dimensional model—including nodes, elements, section definitions, material and orientation definitions, rebar, and contact pair definitions—is generated from the original model. Symmetric model generation from a model with general contact is not allowed. You must redefine most types of kinematic constraints ("Kinematic constraints: overview," Section 35.1.1). However, surface-based constraints ("Mesh tie constraints," Section 35.3.1) and embedded element constraints ("Embedded elements," Section 35.4.1) defined in the original model will be generated automatically in the new three-dimensional model. Changes made to the model as part of the history data—element or contact pair removal/reactivation ("Element and contact pair removal and reactivation," Section 11.2.1) or changes to friction properties ("Changing friction properties during an Abaqus/Standard analysis" in "Frictional behavior," Section 37.1.5)—will not be transferred to the new model. Such changes will

have to be redefined in the history data of the new model. All element and node sets defined in the original model will be used in the new model. These sets will contain all of the new elements and nodes that originated from the original sets.

Additional nodes, elements, contact surfaces, etc. can also be defined to create parts of the model that were not specified in the original model. You must ensure that the numbering of these nodes and elements does not conflict with those used by the symmetric model generation capability. You can control the node and element numbering in the new model (as described below for each type of revolved model) so that you can define additional parts of the model without the risk of conflicting element and node labels. The smallest node/element number used in defining additional parts of the new model should be greater than the largest node/element number generated by the symmetric model generation capability.

Eliminating duplicate nodes

Duplicate nodes will be generated in certain situations. Such nodes can be eliminated to ensure that the mesh is connected properly. Duplicate nodes can be generated on the axis of revolution of a revolved model, on the connection planes between sectors of a periodic model, and on the connection plane between the two parts of a reflected model. You can specify the tolerance distance, *d*, to be used in the search for duplicate nodes. The default distance is 1.0% of the average element dimension. In some cases a tolerance distance that is smaller than the default value needs to be specified if, for example, the distance between two nodes on one of the connection planes in the original sector of a periodic model is smaller than the default tolerance distance. Closely spaced nodes elsewhere in the model, such as between interface surfaces or on parts of the model that are generated with any of the other model definition options, will not be eliminated.

Input File Usage: Use one of the following options to specify the tolerance to be used in the search for duplicate nodes:

∗SYMMETRIC MODEL GENERATION, PERIODIC, TOLERANCE=*d*
∗SYMMETRIC MODEL GENERATION, REVOLVE, TOLERANCE=*d*
∗SYMMETRIC MODEL GENERATION, REFLECT, TOLERANCE=*d*

Writing the new model definition to an external file

You can specify the name of an external file (without an extension) to which the data for the new model definition will be written. The extension **`.axi`** will be added to the file name provided. The file can be edited to modify or to extend the model generated by Abaqus/Standard.

Input File Usage: Use one of the following options:

∗SYMMETRIC MODEL GENERATION, PERIODIC, FILE NAME=*name*
∗SYMMETRIC MODEL GENERATION, REVOLVE, FILE NAME=*name*
∗SYMMETRIC MODEL GENERATION, REFLECT, FILE NAME=*name*

Identifying the restart files

The symmetric model generation capability uses the restart (**`.res`**), analysis database (**`.stt`** and **`.mdl`**), and part (**`.prt`**) files from the old model to generate the new model. The name of the restart files from the old model must be specified when the new analysis is executed by using the **oldjob** parameter

in the command for running Abaqus or by answering a request made by the command procedure (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Verifying the new model

It is recommended that you verify the new model carefully before an analysis is performed. The symmetric model generation capability requires only information stored in the restart file during a data check run to generate the new model, which allows you to verify the new model before the analysis of the original model is performed. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Revolving an axisymmetric cross-section

You can create a three-dimensional model by revolving the cross-section of a two-dimensional axisymmetric model about a symmetry axis starting at a prescribed reference plane, $\theta = 0.0°$. Both the symmetry axis and reference plane of the new three-dimensional model can be oriented in any direction with respect to the global coordinate system (see Figure 10.4.1–1). A nonuniform discretization in the circumferential direction can be specified.

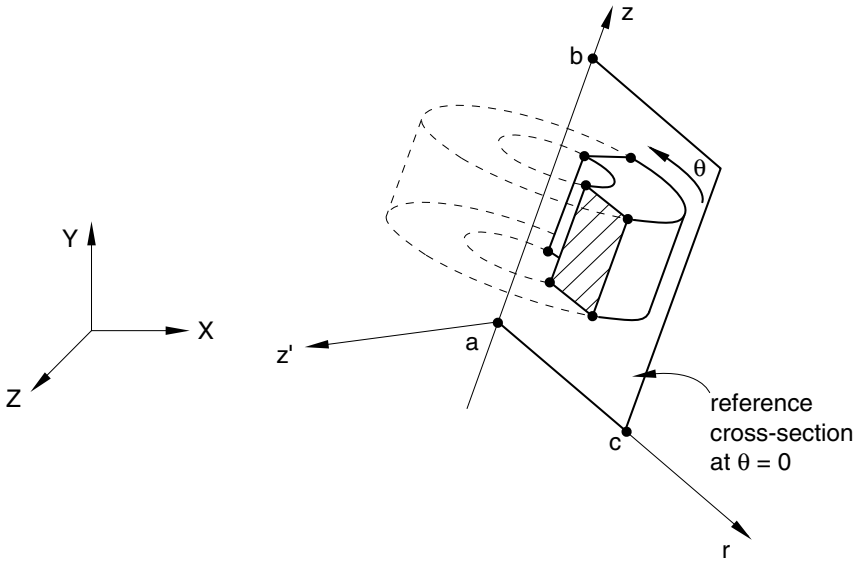


Figure 10.4.1–1 Revolving an axisymmetric cross-section.

Specify the coordinates of points *a*, *b*, and *c* shown in Figure 10.4.1–1, followed by a list that defines the discretization in the circumferential direction containing the segment angle, number of elements per segment, and the bias ratio of the segment. Several segment angles, each with a different number of

element subdivisions and a different bias ratio, can be used to define the complete discretization around the circumference of the revolved model. The endpoint of a cross-section revolved through 360.0° will always be connected to the origin of revolution, $\theta = 0.0°$, regardless of the value specified for the duplicate node tolerance.

Input File Usage: *SYMMETRIC MODEL GENERATION, REVOLVE

Local orientation system

A local cylindrical orientation system is always used for element output of stress, strain, etc. A default local orientation definition is provided if the material in the original axisymmetric model does not contain an orientation definition. This default orientation is defined with the polar axis of the system along the axis of revolution, with an additional 90.0° rotation about the local 1-direction so that the local axes are 1=radial, 2=axial, and 3=circumferential. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. This orientation system is always provided, even if an orientation is specified in the original axisymmetric model. However, if the results of the axisymmetric analysis are mapped onto the new three-dimensional model (see "Transferring results from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh," Section 10.4.2) and an orientation definition is associated with the material in the original model, the original orientation revolved about the axis of symmetry replaces this default orientation definition.

Controlling the new node and element numbering

You can define the increments in numbers between each node and element around the circumference of the three-dimensional model. The numbering starts at the reference cross-section $\theta = 0.0°$. The reference cross-section uses the same numbering as the original axisymmetric model. The defaults are the largest node and element numbers used in the original axisymmetric model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels. Each offset value should be greater than or equal to the maximum node or element label, respectively, used in the original model. When specifying the offset value, care must be taken that the generated node or element does not exceed the maximum value allowed, which is 999,999,999.

Input File Usage: *SYMMETRIC MODEL GENERATION, REVOLVE, NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Correspondence between axisymmetric and three-dimensional elements

The element type used in the original two-dimensional model determines the element type in the new three-dimensional model. You can specify whether the new element should be either a general three-dimensional element or a cylindrical element. General and cylindrical elements can be used in the same model.

Input File Usage: *SYMMETRIC MODEL GENERATION, REVOLVE
coordinates of points a and b
coordinates of point c

segment angle, number of elements per segment, bias ratio,
CYLINDRICAL or GENERAL
For example, the following input specifies 4 cylindrical elements in a 300° segment and 10 general elements in a 60° segment:

```
*SYMMETRIC MODEL GENERATION, REVOLVE
a_x, a_y, a_z, b_x, b_y, b_z
c_x, c_y, c_z
300.0, 4, 1.0, CYLINDRICAL
60.0, 10, 1.0, GENERAL
```

Regular axisymmetric elements (CAX), axisymmetric elements with twist (CGAX), shell elements, membrane elements, rigid elements, and surface elements can be used in the two-dimensional model; however, nonlinear axisymmetric elements (CAXA) cannot be used. A two-dimensional model that contains incompatible mode elements; first-order, reduced-integration, continuum elements; shell elements; or rigid elements cannot be used to generate cylindrical elements. The correspondence between the axisymmetric element type and the equivalent three-dimensional element type (general or cylindrical) is shown in Table 10.4.1–1.

Table 10.4.1–1 Correspondence between axisymmetric and three-dimensional (general and cylindrical) element types.

Axisymmetric element	General three-dimensional element	Cylindrical element
ACAX3	AC3D6	
CAX3	C3D6	CCL9
CAX3H	C3D6H	CCL9H
CGAX3	C3D6	CCL9
CGAX3H	C3D6H	CCL9H
CGAX3T	C3D6T	
DCAX3	DC3D6	
ACAX4	AC3D8	
CAX4	C3D8	CCL12
CAX4H	C3D8H	CCL12H
CAX4I	C3D8I	
CAX4R	C3D8R	
CAX4RH	C3D8RH	
CGAX4	C3D8	CCL12

Axisymmetric element	General three-dimensional element	Cylindrical element
CGAX4H	C3D8H	CCL12H
CGAX4R	C3D8R	
CGAX4RH	C3D8RH	
CAX4T	C3D8T	
CAX4RT	C3D8RT	
CAX4HT	C3D8HT	
CAX4RHT	C3D8RHT	
CGAX4T	C3D8T	
CGAX4RT	C3D8RT	
CGAX4HT	C3D8HT	
CGAX4RHT	C3D8RHT	
DCAX4	DC3D8	
DCCAX4	DCC3D8	
DCCAX4D	DCC3D8D	
ACAX6	AC3D15	
CAX6	C3D15	CCL18
CAX6H	C3D15H	CCL18H
CGAX6	C3D15	CCL18
CGAX6H	C3D15H	CCL18H
DCAX6	DC3D15	
ACAX8	AC3D20	
CAX8	C3D20	CCL24
CAX8H	C3D20H	CCL24H
CAX8R	C3D20R	CCL24R
CAX8RH	C3D20RH	CCL24RH
CGAX8	C3D20	CCL24
CGAX8H	C3D20H	CCL24H
CGAX8R	C3D20R	CCL24R

Axisymmetric element	General three-dimensional element	Cylindrical element
CGAX8RH	C3D20RH	CCL24RH
CAX8T	C3D20T	
CAX8RT	C3D20RT	
CAX8HT	C3D20HT	
CAX8RHT	C3D20RHT	
CGAX8T	C3D20T	
CGAX8RT	C3D20RT	
CGAX8HT	C3D20HT	
CGAX8RHT	C3D20RHT	
DCAX8	DC3D20	
SAX1	S4R	
DSAX1	DS4	
SAX2	S8R	
DSAX2	DS8	
MAX1	M3D4R	MCL6
MGAX1	M3D4R	MCL6
MAX2	M3D8R	MCL9
MGAX2	M3D8R	MCL9
RAX2	R3D4	
SFMAX1	SFM3D4R	SFMCL6
SFMGAX1	SFM3D4R	SFMCL6
SFMAX2	SFM3D8R	SFMCL9
SFMGAX2	SFM3D8R	SFMCL9

Limitations

- First- and second-order elements cannot be used together in the axisymmetric model.
- Nonaxisymmetric elements such as springs, dashpots, beams, and trusses will be ignored in the model generation.

- Only surface-based contact pairs can be revolved. Models using general contact cannot be revolved. Contact conditions modeled with contact elements will be ignored in the model generation.
- A two-dimensional model that includes incompatible mode elements; first-order, reduced-integration, continuum elements; shell elements; or rigid elements cannot be used to generate cylindrical elements.
- Rebar with nonuniform spacing in the radial direction of an axisymmetric element cannot be revolved.
- Most types of kinematic constraints cannot be revolved. However, surface-based constraints ("Mesh tie constraints," Section 35.3.1) and embedded element constraints ("Embedded elements," Section 35.4.1) defined in the original model will be generated automatically in the new three-dimensional model.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be revolved.

Revolving a three-dimensional sector to create a periodic model

You can create a three-dimensional periodic model by revolving a single three-dimensional sector about a symmetry axis. Each generated sector in the periodic model can span the same angle in the circumferential direction, such as in a vented disc, or alternatively, can have a variable angle, such as in a treaded tire. In both cases, each sector always has the same geometry and mesh. Both the symmetry axis and the original three-dimensional sector can be oriented in any direction with respect to the global coordinate system (see Figure 10.4.1–2). Mismatched surface meshes can be used between sectors. Both open (the structure has end edges) or closed loop periodic structures can be generated. If a closed loop periodic structure needs to be created, the sum of the segment angles over all the sectors must be equal to 360°.

Defining a periodic model with a constant angle

To define a periodic model with a constant angle, you must specify the coordinates of points a and b shown in Figure 10.4.1–2 to define the symmetry axis. You then define the segment angle, θ (in degrees), of the original sector and the number of three-dimensional repetitive sectors, N, including the original sector, in the generated periodic model.

Input File Usage: *SYMMETRIC MODEL GENERATION, PERIODIC=CONSTANT
coordinates of points a and b
θ, N

Defining a periodic model with a variable angle

To define a periodic model with a variable angle, the surfaces on both sides of the original sector must be completely planar. You specify the coordinates of points a and b shown in Figure 10.4.1–2 to define the symmetry axis. You then define the segment angle, θ (in degrees), of the original sector and the number of three-dimensional repetitive sectors, N, including the original sector, in the generated periodic model. Next, you specify an additional number of three-dimensional sectors to be generated, M, and the angular

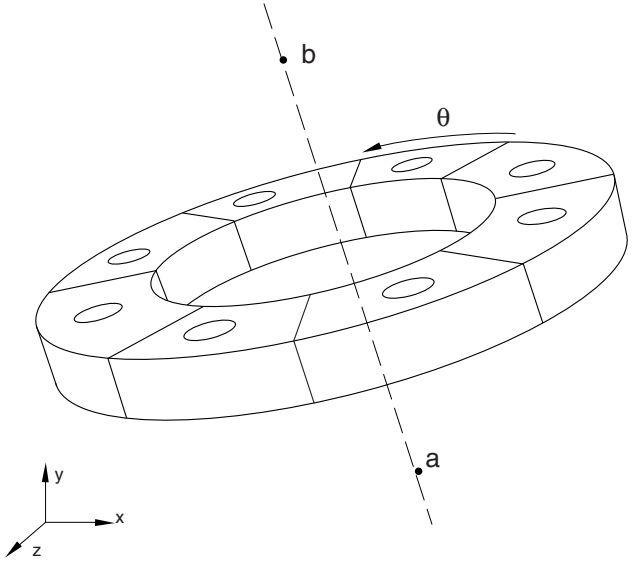


Figure 10.4.1–2 Revolving a three-dimensional sector to form a periodic model.

scaling factor, *f*, in the circumferential direction with respect to the original sector to be applied to these additional sectors. You can define pairs of additional sectors and scaling factors as needed.

Input File Usage: *SYMMETRIC MODEL GENERATION, PERIODIC=VARIABLE
coordinates of points a and b
θ, N
M_1, f_1
M_2, f_2
Etc.

For example, the following input creates a 210° three-dimensional model with 7 sectors with the angles of 20°, 20°, 30°, 30°, 30°, 40°, and 40°, respectively:

```
*SYMMETRIC MODEL GENERATION, PERIODIC=VARIABLE
a_x, a_y, a_z, b_x, b_y, b_z
20.0,2
3,1.5
2,2.0
```

Applying constraints to symmetric surfaces with mismatched meshes

If the symmetric surfaces in the original sector have precisely matched meshes, as shown in Figure 10.4.1–3, any duplicate nodes that are generated will be eliminated automatically to ensure that the mesh is connected properly between the neighboring sectors when revolving the original sector about the symmetry axis to create a periodic model.

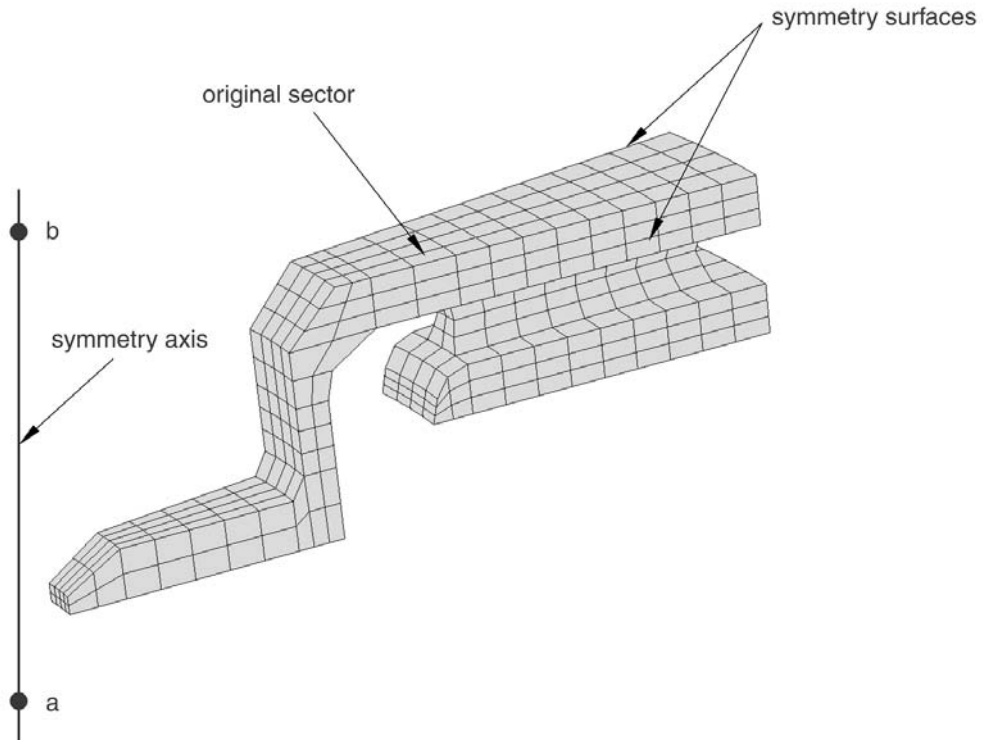


Figure 10.4.1–3 Surfaces with precisely matching meshes on the original sector.

In all other cases you must define one or more pairs of corresponding surfaces on each side of the original sector (see "Surfaces: overview," Section 2.3.1) in the original model and specify the pairs of corresponding surfaces in the symmetric model generation definition.

Optionally, you can also specify the tolerance distance within which nodes on one surface of a sector must lie from the corresponding surface of the neighboring sector to be constrained. Nodes on the surface of the sector that are further away from the corresponding surface of the neighboring sector than this distance are not constrained. The default value for the tolerance distance is 5% or 10% of the typical element size in the surfaces of the original sector, depending on whether node-to-surface or surface-to-surface type constraints are used, respectively.

You can also specify whether surface-to-surface (default) or node-to-surface constraints should be used. Constraints between the automatically generated neighboring pairs of corresponding surfaces are then applied with an automatically generated surface-based tie constraint ("Mesh tie constraints," Section 35.3.1) when revolving the original sector about the symmetry axis to create a periodic model. The first surface of each specified pair is the slave surface, and all degrees of freedom of the nodes in the surface will be eliminated by internally generated multi-point constraints.

Input File Usage: Use the following options in the original model:

∗SURFACE, NAME=*master*
∗SURFACE, NAME=*slave*

Use the following option in the new model with a constant angle for each sector:

∗SYMMETRIC MODEL GENERATION, PERIODIC=CONSTANT
a_x, a_y, a_z, b_x, b_y, b_z
θ, N
slave, master, tolerance distance, SURFACE or NODE

Use the following option in the new model with a variable angle for each sector:

∗SYMMETRIC MODEL GENERATION, PERIODIC=VARIABLE
a_x, a_y, a_z, b_x, b_y, b_z
θ, N
M, f
slave, master, tolerance distance, SURFACE or NODE

Local orientation system

A local cylindrical orientation system is always used for element output of stress, strain, etc. If an orientation is specified in the original three-dimensional sector (see "Orientations," Section 2.2.5), the orientation system in the new model is defined by revolving the original orientation system about the symmetry axis. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. If the material in the original three-dimensional sector does not contain an orientation definition, a default local orientation definition is provided. This default orientation is defined by revolving the global coordinate system in the original model about the axis of symmetry in the new model.

Controlling the new node and element numbering

You can define the increments in numbers between each node and element around the circumference of the three-dimensional model. The numbering starts at the original three-dimensional repetitive sector. The original three-dimensional repetitive sector uses the same numbering as the original model. The defaults are the largest node and element numbers used in the original model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels. Each offset value should be greater than or equal to the maximum node or element label, respectively, used in the original model. When specifying the offset value, care must be taken that the generated node or element does not exceed the maximum value allowed, which is 999,999,999.

Input File Usage: ∗SYMMETRIC MODEL GENERATION, PERIODIC,
NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Limitations

- Only surface-based contact pairs can be revolved. Models using general contact cannot be revolved. Contact conditions modeled with contact elements will be ignored in the model generation.

- Most types of kinematic constraints cannot be revolved. However, surface-based constraints ("Mesh tie constraints," Section 35.3.1) and embedded element constraints ("Embedded elements," Section 35.4.1) defined in the original model will be generated automatically in the new three-dimensional model. One exception is that surface-based ties for enforcing cyclic symmetric constraints are not revolved.
- Surface-based distributed coupling constraints—e.g., couplings ("Coupling constraints," Section 35.3.2), shell-to-solid couplings ("Shell-to-solid coupling," Section 35.3.3), and fasteners ("Mesh-independent fasteners," Section 35.3.4)—cannot be revolved and must be redefined.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be revolved. Beam and frame elements cannot be revolved.

Reflecting a partial three-dimensional model

You can create a three-dimensional model by combining two parts of a symmetric three-dimensional model. One of the parts is the original model, and the other part is obtained by reflecting the original model through a symmetry line (Figure 10.4.1–4) or plane (Figure 10.4.1–5).

Specify the coordinates of points *a*, *b*, and (if required) *c* shown in Figure 10.4.1–4 and Figure 10.4.1–5.

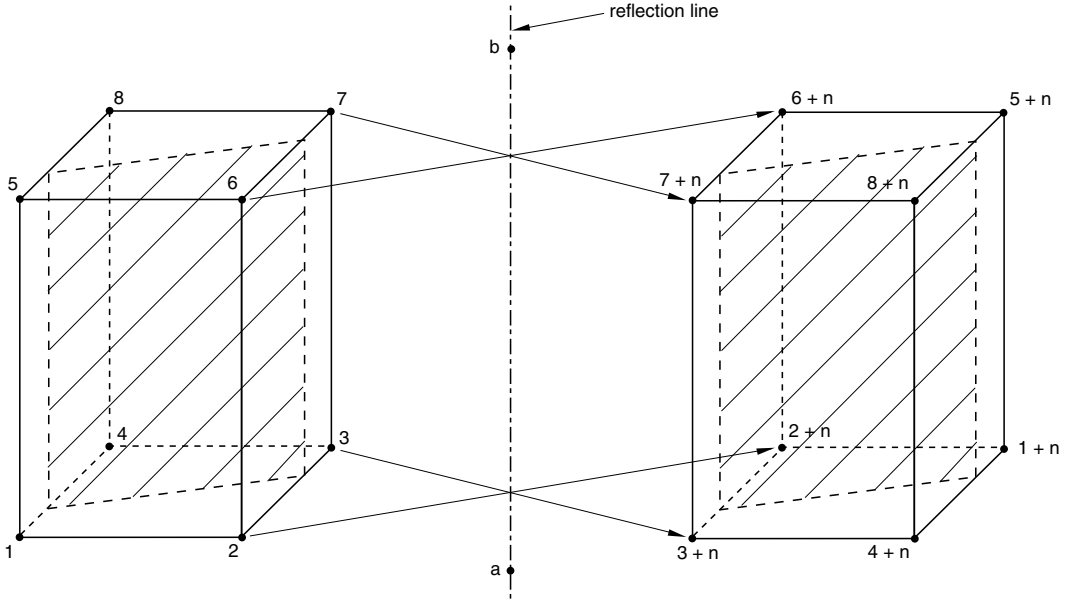


Figure 10.4.1–4 Reflecting a three-dimensional model through line ab with node offset n.

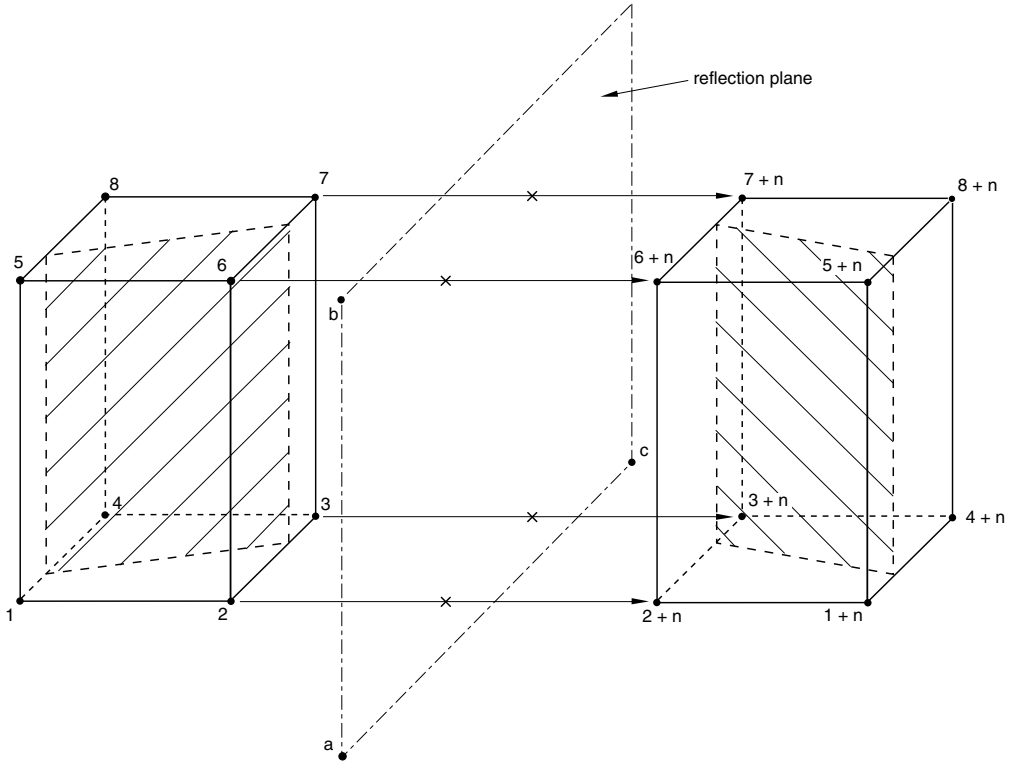


Figure 10.4.1–5 Reflecting a three-dimensional model through a plane abc with node offset n.

Input File Usage: Use one of the following options:

∗SYMMETRIC MODEL GENERATION, REFLECT=LINE
∗SYMMETRIC MODEL GENERATION, REFLECT=PLANE

Controlling the new node and element numbering

You can specify constants that must be added to the original node and element numbers for numbering the reflected part of the three-dimensional model. The defaults are the maximum node and element numbers used in the original model. Control over the numbering allows you to define additional parts of the model without the risk of conflicting element and node labels.

Input File Usage: ∗SYMMETRIC MODEL GENERATION, REFLECT,
NODE OFFSET=*offset*, ELEMENT OFFSET=*offset*

Limitations

- Only surface-based contact pairs can be reflected. Models using general contact cannot be reflected. Contact conditions modeled with contact elements will be ignored in the model generation.
- You must ensure that master surfaces remain continuous after reflection. A discontinuous surface is created when the surface in the original model does not intersect the connection plane between the two parts of the symmetric structure.
- Rigid surfaces cannot be reflected. The rigid surface definition of the original model is simply repeated in the new model. You must, therefore, specify the complete rigid surface in the original model.
- Most types of kinematic constraints cannot be reflected. However, surface-based constraints ("Mesh tie constraints," Section 35.3.1) and embedded element constraints ("Embedded elements," Section 35.4.1) defined in the original model will be generated automatically in the new three-dimensional model.
- Only stress/displacement, heat transfer, coupled temperature-displacement, and acoustic elements can be reflected.
- Nonaxisymmetric elements such as springs, dashpots, beams, and trusses cannot be reflected.

10.4.2 TRANSFERRING RESULTS FROM A SYMMETRIC MESH OR A PARTIAL THREE-DIMENSIONAL MESH TO A FULL THREE-DIMENSIONAL MESH

Product: Abaqus/Standard

References

- "Symmetric model generation," Section 10.4.1
- *SYMMETRIC RESULTS TRANSFER

Overview

Symmetric results transfer:

- reduces the analysis cost of structures that may first undergo symmetric deformation followed by nonsymmetric deformation later during the loading history;
- can be used to transfer the solution of an axisymmetric model to a three-dimensional model;
- can be used to transfer the solution of the symmetric part of a three-dimensional model to a full three-dimensional model;
- must be used in conjunction with the symmetric model generation capability (see "Symmetric model generation," Section 10.4.1); and
- can be used only to transfer the solution of a stress/displacement, heat transfer, coupled temperature-displacement, or coupled acoustic-structural analysis to a new model.

Transferring the solution from a symmetric mesh or a partial three-dimensional mesh to a full three-dimensional mesh

The symmetric results transfer capability can be used to transfer the solution of an axisymmetric model to a three-dimensional model or to transfer the solution of the symmetric part of a three-dimensional model to a full three-dimensional model. The symmetric model generation capability described in "Symmetric model generation," Section 10.4.1, must be used to generate the three-dimensional model.

The symmetric results transfer capability is not available for models defined in terms of an assembly of part instances.

The solution that is transferred to the new model consists of the deformed configuration and corresponding material state, which includes strains and all state variables. The nodes are imported with their original coordinates. This solution becomes the initial or base state in the new analysis.

Specifying the time at which the solution obtained in the original model must be read

You specify the time at which the solution obtained in the original model must be read. The required step and increment or iteration must have been written to the restart files during the original analysis.

Input File Usage: Use the following option if the solution is transferred from any analysis other than a direct cyclic procedure:

*SYMMETRIC RESULTS TRANSFER, STEP=*step*, INC=*increment*

Use the following option if the solution is transferred from a previous direct cyclic analysis:

*SYMMETRIC RESULTS TRANSFER, STEP=*step*, ITERATION=*iteration*

Obtaining equilibrium

You must ensure that the model is in equilibrium at the beginning of the analysis. It is recommended that an initial step definition be included using boundary conditions and loading that match the state of the model from which the results are transferred. An initial time increment equal to the total time should be used for this step to allow Abaqus/Standard to try and achieve the equilibrium in one increment. If needed, Abaqus/Standard can resolve the stress unbalance linearly over the step such that more than one increment is used. You can choose to have the stress unbalance resolved in the first increment of the step instead.

Input File Usage: Use the following option to have Abaqus/Standard resolve the stress unbalance linearly over the step:

*SYMMETRIC RESULTS TRANSFER, UNBALANCED STRESS=RAMP

Use the following option to have Abaqus/Standard resolve the stress unbalance in the first increment of the step:

*SYMMETRIC RESULTS TRANSFER, UNBALANCED STRESS=STEP

Identifying the restart files

The symmetric results transfer capability uses the restart (**`.res`**), analysis database (**`.stt`** and **`.mdl`**), part (**`.prt`**), and output database (**`.odb`**) files from the old analysis to transfer the solution data to the new mesh. The name of the restart files from the old analysis must be specified when the new analysis is executed by using the **oldjob** parameter in the command for running Abaqus or by answering a request made by the command procedure (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Verifying the new model

It is recommended that you verify that the new model is generated correctly before results are transferred or any analysis is performed. The model generation capability requires only information stored in the restart files during a data check run to generate the new model, which allows you to verify the new model before the analysis of the original model is performed. A data check analysis is performed by using the **datacheck** parameter in the command for running Abaqus (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2).

Once the model has been verified, the analysis of the original model can be performed and the results can be transferred to the new model.

The transferred solution can be written to the results files by requesting output at the beginning of a step (the zero increment; see "Output," Section 4.1.1). This solution can also be viewed in Abaqus/CAE.

Orientation system

When results are transferred from an axisymmetric model to a three-dimensional model, a local cylindrical orientation system is used for element output of stress, strain, etc. A default local orientation definition ("Orientations," Section 2.2.5) is provided if the material in the original axisymmetric model does not contain an orientation definition. This default orientation is defined with the polar axis of the system along the axis of revolution with an additional 90° rotation about the local 1-direction so that the local axes are 1=radial, 2=axial, and 3=circumferential. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. It is assumed that the material properties are specified in this system. If, on the other hand, an orientation definition is associated with the material in the original model, the orientation in the new three-dimensional model will be that orientation definition revolved about the axis of symmetry.

When results are transferred from a partial three-dimensional model to a full three-dimensional model by reflecting the partial three-dimensional model, a local material orientation is created in the full three-dimensional model based on the corresponding orientation definition in the partial three-dimensional model. However, if the material does not contain an orientation definition in the partial three-dimensional model and the partial three-dimensional model is not created by revolving an axisymmetric model, no local orientation definition is created in the full three-dimensional model. The full three-dimensional model uses a global coordinate system.

When results are transferred from a three-dimensional sector to a periodic three-dimensional model by revolving the three-dimensional sector about its symmetry axis, a local cylindrical orientation system is always used for element output of stress, strain, etc. If an orientation is specified in the original three-dimensional sector, the orientation system in the new model is defined by revolving the original orientation system about the symmetry axis. If shells or membranes are used, the projections of the local 2- and 3-axes onto the surface of the shell or membrane are taken as the local directions on the surface. If the material in the original three-dimensional sector does not contain an orientation definition, a default local orientation definition is provided. This default orientation is defined by revolving the global coordinate system in the original model about the axis of symmetry in the new model.

Coordinate system at nodes

The displacement and rotational components obtained from the original model are first transformed into a global, rectangular Cartesian axis system before the results are transferred. If local coordinate directions are required in the new model, a nodal transformation ("Transformed coordinate systems," Section 2.1.5) must be specified in the new model to define this coordinate system.

Limitations

The following limitations exist for result transfer from an axisymmetric model to a 3D model:

- Result transfer is not available from 8-node reduced-integration axisymmetric elements (CAX8R and CAX8RH) to the corresponding 20-node brick elements (C3D20R and C3D20RH) when the elements are underlying the slave surface in a contact pair.
- SAX2 is a finite-strain shell, while S8R is a small-strain shell. Do not use this combination when deformations are large in the original analysis.

The following limitation exists for result transfer from a symmetric 3D model to a full 3D model:

- Result transfer is not supported for shells with five degrees of freedom per node (STRI65, S8R5, and S9R5).

Initial conditions

Initial conditions (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) can be specified on all nodes and elements, including the part of the model generated using symmetric model generation (see "Symmetric model generation," Section 10.4.1). However, in most cases the symmetric results transfer capability will overwrite the initial condition values with the solution obtained from the original model. An exception is initial temperatures and field variables. Initial temperatures and field variables are overwritten only when temperatures and field variables are specified in the original model. If only part of the original model contains a specification for temperatures or field variables, the remaining part of the model is assumed to have initial conditions with a magnitude of zero. This distribution of the field will be transferred to the new model. If temperatures and/or field variables are not defined anywhere in the original model, the initial conditions specified in the new model are applied.

Boundary conditions

All boundary conditions must be redefined; the symmetric result transfer capability ignores the boundary conditions specified in the original model. You must ensure that the model is in equilibrium at the beginning of the analysis; therefore, an initial step definition should be included using boundary conditions and loading that match the state of the model from which the results are transferred.

Loads

All loads must be redefined; the symmetric result transfer capability ignores the loads specified in the original model. You must ensure that the model is in equilibrium at the beginning of the analysis; therefore, an initial step definition should be included using boundary conditions and loading that match the state of the model from which the results are transferred.

Material options

All of the material definitions defined in the original model will be transferred to the new model.

Elements

Any element or contact pair removal/reactivation definition (see "Element and contact pair removal and reactivation," Section 11.2.1) that was active in the original model should be respecified.

Output

All of the standard output variables available for stress/displacement elements can be used with the symmetric results transfer capability.

The solution that is transferred to the new model can be written to the results (`.fil`) file by requesting output at the beginning of a step (the zero increment; see "Output," Section 4.1.1). It can also be displayed in Abaqus/CAE.

10.4.3 ANALYSIS OF MODELS THAT EXHIBIT CYCLIC SYMMETRY

Products: Abaqus/Standard Abaqus/CAE

References

- "Natural frequency extraction," Section 6.3.5
- "Mode-based steady-state dynamic analysis," Section 6.3.8
- *CYCLIC SYMMETRY MODEL
- *SELECT CYCLIC SYMMETRY MODES
- *SURFACE
- *TIE
- "Defining cyclic symmetry," Section 15.13.19 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The cyclic symmetry analysis technique in Abaqus/Standard:

- makes it possible to analyze the behavior of a 360° structure with cyclic symmetry based on a model of a repetitive sector;
- can determine the response to cyclic symmetric loading in static, quasi-static, and heat transfer analyses;
- can calculate all eigenfrequencies and eigenmodes of the 360° structure with the block Lanczos eigenfrequency extraction procedure;
- can determine the response to loading corresponding to a given cyclic symmetry mode in modal-based steady-state dynamic analysis; and
- does not require that matched meshes be used on the symmetry surfaces.

Introduction

Structures that exhibit cyclic symmetry provide the analyst with an opportunity to model an entire 360° structure at considerably reduced computational expense by analyzing only a single repetitive sector of the model. Typically, this is the smallest sector that can be identified, although this is not necessary. For example, if a structure consists of 16 repetitive sectors it is possible to use a 45° model containing two repetitive sectors. The sectors are numbered in the counterclockwise direction to the axis of cyclic symmetry (as described further below). Of course this is less efficient than using a 22.5° model with one sector. There is no restriction that the meshes on the two symmetry surfaces of a repetitive sector match in any way.

There are two basic cases that must be considered in such an analysis: a model that has a cyclic symmetric initial state and a cyclic symmetric response, and a model with a cyclic symmetric initial state

but a nonsymmetric response. The cyclic symmetry capability in Abaqus/Standard provides for linear and nonlinear analysis of cyclic symmetric structures with cyclic symmetric response. The condition that the structure be cyclic symmetric holds throughout the analysis, so in a loading step it is not possible to have any nonsymmetric deformation in the structure at any time. Therefore, only cyclic symmetric loads can be applied for this situation.

Analysis of cyclic symmetric structures that exhibit nonsymmetric response requires additional consideration. Such an analysis can be performed only in a linear perturbation step, since the nonsymmetric deformation invalidates the assumption of a cyclic symmetric "base state" for any subsequent step in a general nonlinear analysis. The full response of an entire cyclic symmetric structure, such as the structure illustrated in Figure 10.4.3–1, can be represented as a linear combination of several independent basic responses, each of which corresponds to some k-fold cyclic symmetry mode.

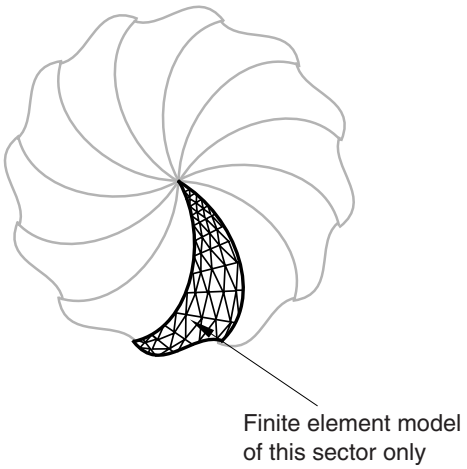


Figure 10.4.3–1 Cyclic symmetric structure.

The cyclic symmetry mode number, which is sometimes also referred to as the "nodal diameter," indicates the number of waves along the circumference in a basic response. Figure 10.4.3–2, Figure 10.4.3–3, and Figure 10.4.3–4 illustrate basic responses corresponding to the 0-, 1-, and 2-fold modes (nodal diameters 0, 1, and 2) in a cyclic symmetric structure containing four repetitive sectors. A full linear perturbation analysis can be performed by solving a sequence of corresponding linear analyses for a symmetric single sector. Cyclic symmetric boundary conditions (associated with various cyclic symmetry modes) on the single sector give rise to Hermitian stiffness and mass matrices (complex matrices with symmetric real parts and skew-symmetric imaginary parts). The kth linear analysis in the sequence is performed using symmetry conditions that correspond to the k-fold cyclic symmetry mode of the structural response. For a structure exhibiting N-fold cyclic symmetry, only $N/2 + 1$ (N even) or $(N + 1)/2$ (N odd) such analyses are required. This results in a solution for the response of the entire structure at a relatively low computational expense.

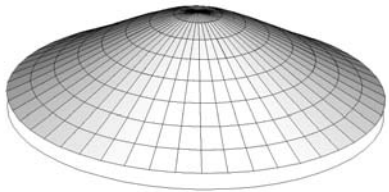


Figure 10.4.3–2 Response corresponding to the 0-fold cyclic symmetry mode.

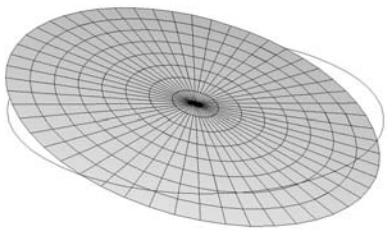


Figure 10.4.3–3 Response corresponding to the 1-fold cyclic symmetry mode.

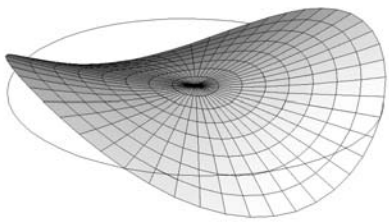


Figure 10.4.3–4 Response corresponding to the 2-fold cyclic symmetry mode.

To perform a general linear analysis of a cyclic symmetric structure, the external forces should be represented as a linear combination of basic loads, each of which corresponds to a symmetry mode and excites a structural response corresponding to the same mode. In static analysis a capability to define loads on any mode other than the 0-fold mode has not yet been implemented. As the response of the 0-fold mode preserves cyclic symmetry, analysis of this type of structure can be done in a general nonlinear step, as well as in a linear perturbation step (as described above). For the same reason, such a step can be used as a preload step for a cyclic symmetric linear perturbation step.

Extraction of a nonsymmetrical response for a cyclic symmetric structure is currently available only for eigenfrequency extraction analysis ("Natural frequency extraction," Section 6.3.5) using the block Lanczos method and for frequency domain, modal-based steady-state dynamic analysis ("Mode-based steady-state dynamic analysis," Section 6.3.8). Natural frequencies corresponding to both symmetric and nonsymmetric eigenmodes can be extracted for a specific cyclic symmetry mode, for a group of cyclic symmetry modes, or for all cyclic symmetry modes. They can be used within the subsequent steady-state dynamic analysis. The eigenmodes onto which the solution is projected are chosen as described in "Selecting the modes and specifying damping" in "Mode-based steady-state dynamic analysis," Section 6.3.8.

In a steady-state modal-based dynamic analysis, concentrated, distributed, and surface loads can be defined as projected onto a specific cyclic symmetry mode. Within the same steady-state dynamics step all applied loads have to be given as projected onto the same cyclic symmetry mode. This limitation implies that the specified cyclic symmetry mode must be the same for all loads within the given steady-state dynamics step.

Defining a cyclic symmetric model

Define the mesh for a single sector of the model, the so called "datum sector." Specify the number of sectors, n, in the 360° model. Define the axis of symmetry by specifying the coordinates (in the global coordinate system) of two points lying on the axis. The axis direction is from the first point to the second point, and the sectors are numbered counterclockwise around the axis, with the datum sector as sector number 1. For a two-dimensional model only a single point needs to be given on the axis. The axis direction is assumed to be in the positive z-direction; hence, the sectors are numbered counterclockwise in the x–y plane.

Input File Usage: ∗CYCLIC SYMMETRY MODEL, N=*n*

In a model defined in terms of an assembly of part instances, the ∗CYCLIC SYMMETRY MODEL option must appear within the model definition (see "Defining an assembly," Section 2.10.1).

Abaqus/CAE Usage: Interaction module: **Interaction→Create**: **Cyclic symmetry**: **Total number of sectors**: *n*

Applying cyclic symmetry constraints

To apply the cyclic symmetry constraints, you must define one or more pairs of corresponding surfaces on each side of the datum sector (see "Surfaces: overview," Section 2.3.1). You can then apply the cyclic

symmetry constraints between the pairs of corresponding surfaces using a cyclic symmetry surface-based tie constraint (see "Defining tied contact in Abaqus/Standard," Section 36.3.7). The first surface of each pair specified in the tie constraint definition is the slave surface, and all degrees of freedom of the nodes in the surface will be eliminated by internally generated multi-point constraints. The second surface of each pair is a master surface. If more than one pair of slave/master surfaces is defined, the rotation direction from the slave surface to the master surface must be the same for all pairs (i.e., clockwise or counterclockwise).

Input File Usage: Use the following options to apply a cyclic symmetry constraint between two surfaces:

∗SURFACE, NAME=*master*
∗SURFACE, NAME=*slave*
∗TIE, CYCLIC SYMMETRY, NAME=*cyclic*
slave, master

Abaqus/CAE Usage: Interaction module: **Interaction→Create**: **Cyclic symmetry**: click **Surface** in the prompt area

Using mismatched surface meshes

In the case of mismatched surface meshes, as shown in Figure 10.4.3–5, the finer mesh should typically be the slave surface. Mismatched meshes may cause some local inaccuracies in the stress field. The magnitude of the inaccuracies depends on the degree of mismatch between the meshes as well as on the element type used: the inaccuracies are typically most pronounced for second-order (modified) tetrahedral elements. Hence, if mismatched surface meshes are used, it is recommended that the sector boundaries be chosen in areas where accuracy of the local stress field is not critical.

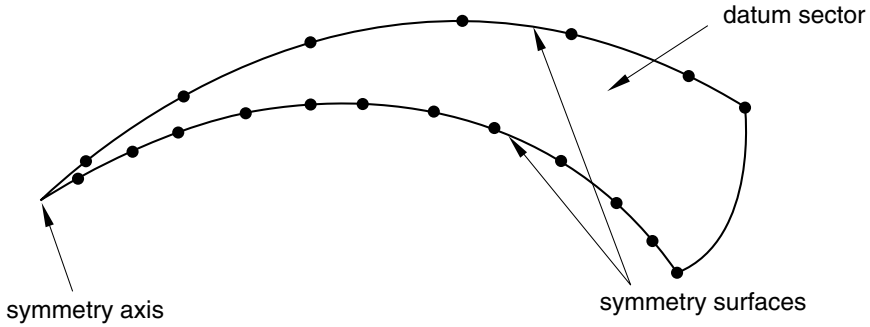


Figure 10.4.3–5 Cyclic symmetry surfaces with mismatched nodes.

For shells the cyclic symmetry condition has to be applied to the nodes on the edges of the shell elements. Currently cyclic symmetry is not supported for element-based surfaces defined on the edges of shells. Therefore, if mismatched meshes are used for shell elements, an element-based surface should be defined on the top or bottom of the shell elements adjacent to the edges that form the master surface. A node-based surface can be defined on the edge that forms the slave surface.

Applying node-to-node cyclic symmetry constraints

In the case of matched meshes, either surface can be chosen as the slave surface. If the surfaces have matched meshes, as shown in Figure 10.4.3–6, it is possible to use a node-based master surface to obtain node-to-node cyclic symmetry constraints. The advantage of this is that Abaqus/Standard will adjust the positions of the nodes on the slave surface so that they precisely match the positions of the nodes on the master surface. This yields the most accurate results and minimizes the computational cost. In this case the slave surface will typically be chosen as a node-based surface as well, although computationally it does not matter since in either case a strict node-to-node constraint is applied.

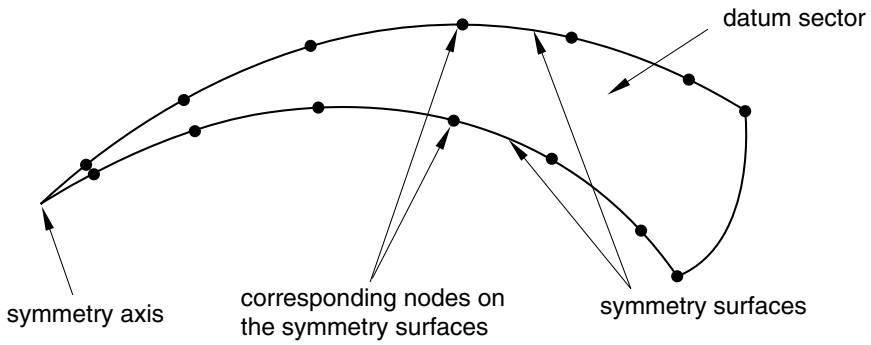


Figure 10.4.3–6 Cyclic symmetry surfaces with node-to-node matching.

For discrete members (such as trusses or beams) the cyclic symmetry condition can be enforced only using node-based surfaces.

Input File Usage: Use the following options to apply a cyclic symmetry constraint between two node-based surfaces:

```
*SURFACE, TYPE=NODE, NAME=master
*SURFACE, TYPE=NODE, NAME=slave
*TIE, CYCLIC SYMMETRY, NAME=cyclic
slave, master
```

Abaqus/CAE Usage: Interaction module: **Interaction→Create**: **Cyclic symmetry**: click **Node Region** in the prompt area

Applying cyclic symmetry conditions on the symmetry axis

If a node is located on the symmetry axis, special cyclic symmetry constraints must be applied for the 0-fold and 1-fold cyclic symmetry modes; whereas all degrees of freedom must be constrained for the other cyclic symmetry modes. For the 0-fold cyclic symmetry mode the degrees of freedom in the plane orthogonal to the symmetry axis are constrained; for the 1-fold cyclic symmetry mode the degrees of freedom along the symmetry axis are constrained. Abaqus/Standard will create these constraints

automatically as long as the node is included in the definition of the slave surface, the master surface, or both the slave and master surfaces.

Obtaining all eigenfrequencies of a cyclic symmetric structure

The natural frequencies and corresponding eigenmodes of a cyclic symmetric structure can be extracted using the eigenfrequency extraction procedure with the Lanczos eigensolver (see "Natural frequency extraction," Section 6.3.5). No additional information is required for the eigenfrequency extraction procedure. All the natural frequencies are sorted in the conventional (ascending) order. For each natural frequency the cyclic symmetry mode number is reported.

The eigenmodes are written in the order corresponding to natural frequencies to the data (`.dat`), results (`.fil`), and output database (`.odb`) files for the user-specified datum sector only. These modes can be expanded in Abaqus/CAE to the entire structure depending on the cyclic symmetry mode number.

There are two different types of eigenmodes: single and paired. The eigenmodes for 0-fold cyclic symmetry are always single. For even N the eigenmodes for the $N/2$-fold cyclic symmetry are also single. The eigenmodes for the remaining $N/2 - 1$ (even N) or $(N - 1)/2$ (odd N) cyclic symmetry modes are paired. The natural frequencies corresponding to the paired eigenmodes are equal and always appear together in the table of the natural frequencies in the data file. The expansion of the eigenmodes with k-fold cyclic symmetry ($k < N/2$) to the sector $i = 2, ..., N$ can be done in the following manner:

$$\begin{bmatrix} u_k^i \\ v_k^i \end{bmatrix} = [T_k^i] \begin{bmatrix} u_k \\ v_k \end{bmatrix},$$

where

$$[T_k^i] = \begin{bmatrix} cos\Psi_k(i-1) & sin\Psi_k(i-1) \\ -sin\Psi_k(i-1) & cos\Psi_k(i-1) \end{bmatrix}.$$

Here (u, v) and (u^i, v^i) are paired eigenmodes corresponding to double natural frequencies on the first (datum) sector and on the ith sectors, respectively; and $\Psi_k = 2\pi k/N$.

From the expressions above it is clear that eigenmodes with 0-fold cyclic symmetry are always symmetric; i.e., $u_0^i = u_0$. Similarly, for even N the eigenmodes with $(N/2)$-fold cyclic symmetry are single, since $u_{(N/2)}^i = (-1)^{(i-1)} u_{(N/2)}$.

Selecting the cyclic symmetry modes

You can select the cyclic symmetry modes for which the eigenfrequency analysis will be performed by specifying the lowest cyclic symmetry mode to be used in the analysis, *nmin*, and the highest cyclic symmetry mode to be used in the analysis, *nmax*. By default, *nmin* is 0. By default, *nmax* is $N/2$ (even N) or $(N - 1)/2$ (odd N). The value of *nmin* cannot be greater than the value of *nmax*, and the value of *nmax* cannot be greater than the default value. If you do not select the cyclic symmetry modes, all possible cyclic symmetry modes are considered in the analysis. You can choose to use only the even cyclic symmetry modes.

Input File Usage: Use the following option to specify the cyclic symmetry modes:

*SELECT CYCLIC SYMMETRY MODES, NMIN=*nmin*, NMAX=*nmax*

Use the following option to request only the even cyclic symmetry modes:

*SELECT CYCLIC SYMMETRY MODES, EVEN

Abaqus/CAE Usage: Use the following option to specify the cyclic symmetry modes:

Interaction module: **Interaction→Create**: **Cyclic symmetry**: toggle on **Specified range** and specify the **Lowest nodal diameter** and **Highest nodal diameter**

You cannot request only the even cyclic symmetry modes in Abaqus/CAE.

Selecting the cyclic symmetry mode for a steady-state dynamic step

Only a single cyclic mode can be excited in a steady-state dynamic step. You specify the cyclic symmetry mode associated with the loading in the load definition.

Input File Usage: Use one of the following options:

*CLOAD, CYCLIC MODE=*k*, REAL or IMAGINARY
*DLOAD, CYCLIC MODE=*k*, REAL or IMAGINARY
*DSLOAD, CYCLIC MODE=*k*, REAL or IMAGINARY

Abaqus/CAE Usage: Interaction module: **Interaction→Create**: **Cyclic symmetry**: **Excited nodal diameter**

Comparison of the cyclic symmetry analysis technique and MPC type CYCLSYM

MPC type CYCLSYM ("General multi-point constraints," Section 35.2.2) provides a subset of the functionality provided by the cyclic symmetry analysis capability. For an eigenvalue analysis MPC type CYCLSYM will allow extraction of the symmetric (0-fold) modes only. The cyclic symmetry analysis capability allows the use of surfaces ("Surfaces: overview," Section 2.3.1) to define the symmetry surfaces for the model, which enables the use of mismatched meshes on the symmetry surfaces, whereas MPC type CYCLSYM can be applied only on a node-to-node basis.

Limitations

The following limitations exist:

- A continuation capability is not available for the cyclic symmetry eigenvalue extraction procedure. Each eigenvalue extraction step will not reuse any eigenmodes obtained in the previous eigenvalue extraction steps.
- The specified cyclic symmetry mode must be the same for all loads defined within a given steady-state dynamic step.
- Base motion is not implemented for cyclic symmetry models.

- Cyclic symmetry conditions are applied to the mechanical degrees of freedom in stress/displacement analysis and temperature degrees of freedom in heat transfer analysis. Cyclic symmetry conditions are not applied to acoustic pressure, pore pressure, and electrical degrees of freedom.
- Cavity radiation cannot be used in cyclic symmetric models.

Initial conditions

All applied initial conditions must be cyclic symmetric.

Boundary conditions

Only cyclic symmetric boundary conditions can be applied. Boundary conditions cannot be applied to the nodes on the slave cyclic symmetry surface.

Loads

In static analysis only cyclic symmetric loads can be applied. Coriolis loads cannot be applied, and the effect of the Coriolis load stiffness is not considered in the frequency analysis.

In modal-based steady-state dynamic analysis the loads are defined on the datum sector for a specific cyclic symmetry mode, which is indicated in the loading definition. For the k-fold cyclic symmetry mode ($0 < k < N/2$) the complex loads f^i and g^i (corresponding to real and imaginary components, respectively) on the sector $i = 1, 2, ..., N$ are obtained in the following manner:

$$f^i = Fcos\Psi_k(i-1) + Gsin\Psi_k(i-1),$$

$$g^i = Gcos\Psi_k(i-1) - Fsin\Psi_k(i-1),$$

where $\Psi_k = 2\pi k/N$ and F and G are real and imaginary components of loads specified for the datum sector, respectively. For the 0-fold cyclic symmetry mode ($k = 0$) this type of loading corresponds to a cyclic symmetric load pattern with $f^i = F$ and $g^i = G$. For $k > 0$ this type of loading is generated when a spatially constant load pattern is applied to a rotating structure (or when a constant load pattern rotates around the structure). For the $N/2$-fold mode the complex loads on the sector i are: $f^i = (-1)^{(i-1)}F$ and $g^i = (-1)^{(i-1)}G$.

Predefined fields

Only cyclic symmetric predefined fields can be applied. Hence, the predefined fields should have the same values at each side of the datum sector.

Material options

No specific restrictions apply to material models for cyclic symmetry models of general procedures. For the frequency analysis procedure, see the remarks in “Natural frequency extraction,” Section 6.3.5.

Elements

Axisymmetric elements should not be used in cyclic symmetry models.

Output

Nodal displacements and element output variables such as stress, strain, and section force are only available for the datum sector. The mass listed in the data file is computed for the whole model.

In the eigenvalue extraction procedure the following special conditions apply:

- If displacement eigenvector normalization is chosen (the default), the largest displacement entry in each eigenvector on the datum sector is unity. If mass eigenvector normalization is chosen, the eigenvectors are normalized so that the generalized mass computed on the datum sector is unity. See "Natural frequency extraction," Section 6.3.5, for details.
- The eigenvalue numbers, cyclic symmetry mode numbers, and corresponding frequencies (in both radians/time and cycles/time) are listed in the data file, along with the generalized masses, composite modal damping factors, participation factors, and modal effective masses. The generalized masses are calculated on the datum sector; composite modal damping factors, participation factors, and modal effective masses are calculated for the entire model.
- You can restrict output to the results and data files by selecting the modes for which output is desired (see "Output to the data and results files," Section 4.1.2).
- With Abaqus/CAE static displacements and eigenmodes can be displayed for any sector. The results of steady-state, modal-based dynamic analysis can also be animated for any number of sectors, including the entire model.

Input file template

```
*HEADING
...
**
*CYCLIC SYMMETRY MODEL, N=integer
N denotes the number of sectors in the entire 360° model.
...
**
*SURFACE, NAME=name, TYPE=ELEMENT
*SURFACE, NAME=name, TYPE=NODE
Surface description for the slave and master nodes that will be referenced in the *TIE option.
...
**
*TIE, CYCLIC SYMMETRY
Indicates the internal MPCs that tie the master and slave surfaces
using the cyclic symmetry condition in the cyclic symmetry models only.
Data lines to specify surface names that will be tied with this option.
...
```

```
**
*STEP (,NLGEOM)
```
If NLGEOM is used, initial stress and preload stiffness effects will be included in subsequent linear perturbation steps, including the frequency extraction step
```
*STATIC
...
*DLOAD
```
Data lines to specify element or element set, load type, value, (direction).
```
...
**
*END STEP
*STEP
*FREQUENCY, EIGENSOLVER=LANCZOS
…
*SELECT CYCLIC SYMMETRY MODES, NMAX=integer, NMIN=integer, EVEN
…
**
*END STEP
*STEP
*STEADY STATE DYNAMICS
…
*SELECT EIGENMODES
```
Use this option to specify the list of eigenmodes used in the response.
```
*MODAL DAMPING
```
Data lines to specify damping coefficients associated with eigenmodes.
```
…
*CLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
```
Data lines to specify node or node set, degree of freedom, value
```
*DLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
```
Data lines to specify element or element set, load type, value, (direction)
```
…
*DSLOAD, CYCLIC MODE=integer, REAL or IMAGINARY
```
Data lines to specify element or element set, load type, value, (direction)
```
…
**
*END STEP
```

10.5 Periodic media analysis

- “Periodic media analysis,” Section 10.5.1

10.5.1 PERIODIC MEDIA ANALYSIS

Product: Abaqus/Explicit

References

- *PERIODIC MEDIA
- *MEDIA TRANSPORT

Overview

The periodic media analysis technique in Abaqus/Explicit:

- is a Lagrangian technique that offers an Eulerian-like view into a moving structure;
- can be used to effectively model systems that are repetitive in nature, such as manufacturing processes involving conveyor belts or continuous forming operations;
- leads to significant analysis time speedup when compared to traditional modeling techniques that may require excessively large meshes; and
- requires topologically identical meshed parts to create the model, which can be accomplished via the parts and instances modeling paradigm.

Introduction

Quite often industrial processes that need to be analyzed involve sections that repeat in a simple pattern and move through a process zone. A prominent example is a conveyor belt with regularly spaced packages, as illustrated schematically in Figure 10.5.1–1 and exemplified with a finite element mesh in Figure 10.5.1–2. Continuous forming operations such as metal rolling are also good examples because the deforming material can be broken up into an arbitrary number of identical sections.

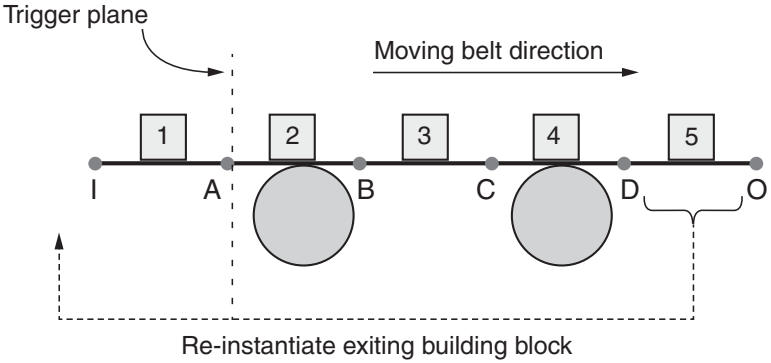


Figure 10.5.1–1 Schematic representation of periodic media.

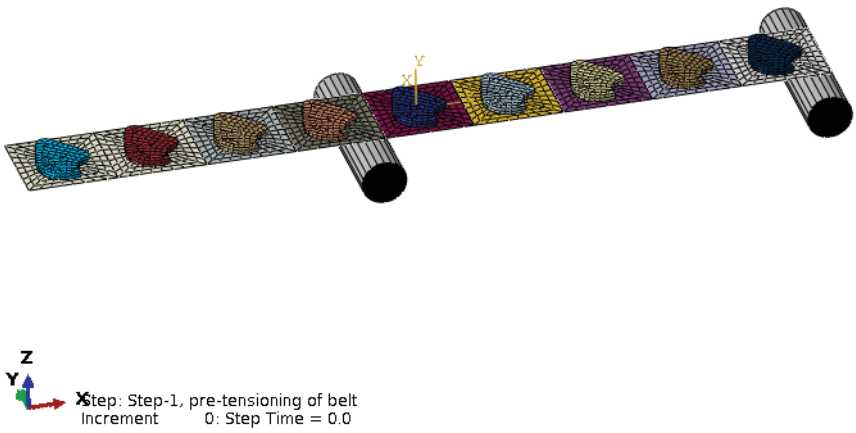


Figure 10.5.1–2 Conveyor belt with packages on top.

For the sake of clarity we will use the conveyor belt example throughout this discussion to illustrate many of the concepts associated with the periodic media analysis technique. Figure 10.5.1–1 shows a conceptual decomposition of the conveyor belt; in reality, the belt is a continuous entity.

Conceptually, the overall model can be decomposed into blocks (topologically identical meshed structures) that are connected together and span the process zone. You create a part that defines a "building block" (the meshed structure that is repeated to model the entire periodic media) and then construct the whole model via a chain of appropriately positioned instances. The periodic media analysis technique provides a simple way to automatically connect these instances together at the front and back ends of adjacent blocks. This technique also provides a convenient way to define loads and boundary conditions that represent the physical system at the unconnected ends of the first and last blocks in the chain. The first block of the chain is referred to as the inlet, and the last block is referred to as the outlet. Finally, when the periodic media moves through the process zone, blocks from the outlet are automatically shuffled to the inlet. The blocks (meshed structures) defined with this technique can interact via contact with other modeling features that are not periodic in nature, such as the rollers depicted in Figure 10.5.1–1.

At the core of the periodic media analysis technique lies the concept of shuffling blocks from the outlet back to the inlet. A dedicated algorithm is used to detect when the inlet has moved too far into the process zone and to shuffle a block from the outlet directly to the inlet. The dashed arrow in Figure 10.5.1–1 illustrates the shuffling process. To ensure a smooth transition, the necessary nodal and element state data from the inlet block are stored at the beginning of the current step. When shuffling occurs, the stored nodal and element state data are mapped to the new inlet block and any inlet/outlet loads or boundary conditions are transferred to the newly exposed block ends.

Thus, the periodic media analysis technique offers a convenient way for an Eulerian-like view into the moving repetitive structure. For example, you may be interested in assessing the package dynamics on the belt at a location somewhere between the rollers in both transient and steady-state conditions.

You define several blocks around that location, you define contact with the rollers as necessary, and you provide appropriate inlet and outlet loading conditions. The periodic media analysis technique provides a convenient and economical way to create and analyze this system. By re-using elements that have left the process zone via this shuffling process, you can avoid the large meshes at the inlet end required for purely Lagrangian simulations.

Constructing a periodic media model

The first step in constructing a periodic media model is to identify the portion of the model that constitutes the building block of the repetitive structure. In Figure 10.5.1–2 one square belt patch together with one asymmetrically shaped package on top constitute such a building block. If you string together several blocks, the entire belt with packages can be modeled as shown.

Defining a building block

The following requirements must be observed when defining each building block:

- an unsorted element set must be defined to include all elements in the building block, and
- an unsorted node set must be defined to include all nodes in the building block.

To ensure the proper transfer of information as the periodic media advances, these unsorted sets must be topologically identical between all blocks. The easiest way to achieve this requirement is to use the parts and instances modeling paradigm. You define one part corresponding to the building block and define unsorted element and node sets as discussed above. You then instantiate the part as many times as needed with the appropriate translations and rotations to generate the periodic media mesh. Constraints such as ties, couplings, and rigid bodies are allowed within a building block. You must ensure that these constraints are defined in a topologically identical fashion in all blocks.

The periodic media analysis technique connects together these otherwise unconnected blocks to create a continuous model. If structural elements (e.g., shells) are used in the connecting regions of the blocks, the nodes on the edges of these regions are connected to the adjacent regions. If continuum elements are used, the nodes on the faces of these regions are connected. For these constraints to be constructed reliably, the following additional requirements must be observed:

- the nodal arrangements at the front and back connecting ends of blocks must be topologically identical,
- the front and back end nodes of adjacent blocks must be coincident,
- the nodal arrangements at the front and back end of the initial inlet block must have coordinates that differ only by a rigid body translation, and
- two node-based surfaces created using unsorted node sets at the front and back end of each block must be defined.

The node-based surfaces are used to automatically generate node-to-node tie constraints between adjacent blocks such that the whole assembly behaves as a continuous entity.

Input File Usage: Use the following option to define the sequence of blocks using unsorted sets and surfaces as described above:

```
*PERIODIC MEDIA, NAME=name
 elset1, nodeset1, frontsurf1, backsurf1
 elset2, nodeset2, frontsurf2, backsurf2
 ...
 elsetn, nodesetn, frontsurfn, backsurfn
```

Each data line provides the set and surface names associated with a given block.

Applying loads and boundary conditions at media ends

In the schematic belt shown in Figure 10.5.1–1, you usually need to apply loads or boundary conditions at both ends of the assembly. At the inlet point I it is often useful to apply a pre-tension load that keeps the belt taut, while at the outlet point O the belt velocity is usually prescribed. As the belt advances and exiting blocks are being shuffled from the outlet to the inlet, the nodes requiring the boundary conditions will change. Therefore, these boundary conditions and loads cannot be prescribed directly at nodes belonging to the block.

The periodic media analysis technique allows for the application of such loading features via two control nodes that are associated with the current inlet and outlet node-based surfaces. The control nodes are similar to reference nodes used in other features (such as kinematic couplings) and impose automatically defined rigid body–like constraints on the nodes at the extreme ends of the assembly. You apply loads and boundary conditions at these control nodes. A rigid body–like constraint is also imposed on the front end nodes of the inlet block, but no loads or boundary conditions can be applied. When exiting blocks are being shuffled back to the inlet, the control points will enforce these rigid body–like constraints on the new extreme end surfaces and remove the rigid body–like constraints from the previous locations. The process is automatic and fully managed by the periodic media analysis technique.

Input File Usage: Use the following option to define control nodes for the inlet and outlet conditions:

```
*PERIODIC MEDIA, INLET CONTROL NODE=node,
 OUTLET CONTROL NODE=node
```

Defining the process zone

When the inlet block moves completely into the process zone, the outlet block is shuffled back to the inlet, as the dashed arrow indicates in Figure 10.5.1–1. A trigger plane controls the precise timing for when the shuffling occurs. When the nodes located at the current inlet point I cross the trigger plane, the shuffling process is launched. The trigger plane is defined using the coordinates of a (usually) stationary node and the z-axis of a user-defined orientation. The local z-axis direction points from the inlet toward the process zone.

Input File Usage: Use the following option to define the trigger plane via a trigger node and orientation:

```
*PERIODIC MEDIA, TRIGGER NODE=node,
 ORIENTATION=orientation
```

Activating a periodic media

The shuffling process can be activated on a step-by-step basis. By default, the shuffling process is inactive. In many cases the configuration of the periodic media in the operating condition can be determined only via simulation. This allows any number of analysis steps to be carried out prior to activating the shuffling process.

The example illustrated in Figure 10.5.1–2 and in "Media transport," Section 3.24.1 of the Abaqus Verification Guide, shows a conveyor belt transporting asymmetrical packages placed initially at regular intervals. In its operating condition the belt will be tensioned. You can pre-stretch the belt assembly in either Abaqus/Standard or Abaqus/Explicit. If the pre-stretch analysis is conducted in Abaqus/Standard, all ties between adjacent blocks as well as boundary conditions at the inlet and outlet ends nodes need to be defined explicitly as the periodic media analysis technique is available only in Abaqus/Explicit. If the pre-stretching step is conducted in Abaqus/Explicit, the shuffling process should remain inactive during the pre-stretching step. A boundary condition should be applied at the front end of the inlet block to keep the inlet block aligned with the direction of the movement in the pre-stretching step, and this boundary condition should be removed after the shuffling process is activated in the following step.

Input File Usage: Use the following option to activate or deactivate the periodic media shuffling process:

```
*MEDIA TRANSPORT
periodic_media_name1, ACTIVE
periodic_media_name2, INACTIVE
...
```

Modeling tips

The periodic media analysis technique is a powerful feature; however, you must exercise good engineering judgement when using it. The following comments and recommendations will help you avoid common pitfalls when using this technique:

- The block shuffling process is inherently noisy as chunks of elements are detached at one end and reattached at the other. Although the process uses appropriate material and kinematic states, small shocks are inherent to the process. A small amount of mass proportional damping is recommended to dampen out this excitation.
- The combination of boundary conditions at the inlet control node and any loads applied in the process zone should ensure that the inlet block moves across the trigger plane without a change in direction. In the conveyor belt example, a good modeling practice would be to place a fixed guide roller at least two blocks away from the trigger plane.
- For more complex geometries (such as belts that change direction between rollers or package wrapping analyses when the belt is the wrapping material itself), it may be necessary to start with a straight sequence of blocks and move the belt rollers (which are not part of the periodic media definition) into the desired locations. Contact interaction between the belt and the rollers would

deform the belt in the desired configuration. This additional analysis step can greatly simplify the definition of the initial mesh.

- Sometimes it may be necessary to model the process of threading a belt wrapping through rollers, just as in physical reality at the start of a manufacturing process. If this leading segment is followed by periodic blocks that include actual packages, you can attach the periodic media mesh to a regular mesh to execute the threading. The periodic media part of the mesh can then be imported into a separate model without the leading mesh, and the analysis of the periodic media consisting only of the wrapper and packages can be executed.

Initial conditions

Initial conditions can be specified at all nodes in the periodic media mesh. Velocity initial boundary conditions can be used to minimize the solution time needed to reach a steady-state operating condition. In cases where pre-stretching is required, importing from the prior analysis rather than performing a multistep analysis allows for initial conditions to be applied to the stretched configuration. Since periodic media definitions are not imported, they must be respecified in every analysis in which they are required.

Boundary conditions

The inlet and outlet control nodes are the only two nodes associated with a periodic media definition at which boundary conditions can be specified when it is active. Furthermore, only velocity boundary conditions are permitted. You must not specify boundary conditions at any other node associated with the periodic media mesh when it is active. While the periodic media is active and if a steady-state solution is sought, these boundary conditions should be kept constant in both direction and magnitude to mitigate solution noise.

Loads

Only concentrated loads can be applied to the inlet and outlet control nodes to either drive or stretch the periodic media. While the periodic media is active, these loads should be kept constant in both direction and magnitude. Gravity loads can be applied as desired. Other distributed loads can also be specified; however, you must keep in mind that the loads will travel with the blocks as they are shuffled.

Material options

All available material models are supported.

Limitations

Periodic media analyses are subject to the following limitations:

- Only membranes, shells, trusses, continuum elements, and rigid elements are allowed within blocks. Rebar layers can also be used, if applicable.
- No explicitly defined constraints are allowed between nodes belonging to different blocks.
- Mass scaling must be defined in the same fashion for all blocks.

The periodic media should not be involved in

- general contact that defines thermal contact properties or coupled Eulerian-Lagrangian contact or
- contact defined via the contact pair algorithm.

Input file template

The following example illustrates a model with two periodic media defined:

```
*HEADING
…
*PERIODIC MEDIA, NAME=belt1, INLET CONTROL NODE=10,
OUTLET CONTROL NODE=110, ORIENTATION=ori1, TRIGGER NODE=210
elset1, nodeset1, frontedgesurf1, backedgesurf1
elset2, nodeset1, frontedgesurf2, backedgesurf2
elset3, nodeset1, frontedgesurf3, backedgesurf3
*PERIODIC MEDIA, NAME=belt2, INLET CONTROL NODE=11,
OUTLET CONTROL NODE=111, ORIENTATION=ori2, TRIGGER NODE=211
elset1, nodeset1, frontedgesurf1, backedgesurf1
elset2, nodeset1, frontedgesurf2, backedgesurf2
elset3, nodeset1, frontedgesurf3, backedgesurf3
*STEP
*DYNAMIC, EXPLICIT
*MEDIA TRANSPORT
belt1, ACTIVE
belt2, INACTIVE
*END STEP
```

10.6 Meshed beam cross-sections

- “Meshed beam cross-sections,” Section 10.6.1

10.6.1 MESHED BEAM CROSS-SECTIONS

Products: Abaqus/Standard Abaqus/Explicit

References

- *BEAM GENERAL SECTION
- *BEAM SECTION GENERATE
- *SECTION ORIGIN
- *SECTION POINTS

Overview

Meshed cross-sections:

- allow for the description of a beam cross-section including multiple materials and complex geometry;
- are meshed in Abaqus/Standard with two-dimensional warping elements, which have an out-of-plane warping displacement as the only degree of freedom;
- generate beam cross-section properties that can be used in a subsequent beam element analysis in either Abaqus/Standard or Abaqus/Explicit;
- allow only isotropic linear elastic material behavior ("Defining isotropic elasticity" in "Linear elastic behavior," Section 22.2.1) or orthotropic linear elastic material behavior for warping elements ("Defining orthotropic elasticity for warping elements" in "Linear elastic behavior," Section 22.2.1); and
- allow stress and strain postprocessing on the beam element model or the two-dimensional warping element model.

Introduction

The response of some structures is beam-like, yet the beam cross-section geometry or multi-material makeup of the cross-section do not permit the use of a predefined library beam cross-section. In these cases a meshed cross-section can be used to model the beam cross-section and to generate beam cross-section properties appropriate for subsequent use in a Timoshenko beam analysis. The beam properties are generated assuming a thick-walled (solid) cross-section with unconstrained out-of-plane warping, so open-section beam elements cannot use the beam cross-section properties generated from the meshed section (see "Beam modeling: overview," Section 29.3.1). The generated beam cross-section properties include axial, bending, torsional, and transverse shear stiffnesses; mass, rotary inertia, and damping properties; and the centroid and shear center of the cross-section. In addition, the equivalent beam cross-section properties include information on stress recovery, such as the warping function and its derivatives.

A typical example of a structure that requires a meshed cross-section is the hull of a ship for whipping analysis, where the ship's hull has a multi-cell and multi-material construction. Other examples include an airfoil-shaped rotor blade or wing, a layered composite I-beam (with fibers running along the length of the beam axis or perpendicular to it), etc.

Modeling approach

As shown in Figure 10.6.1–1, a meshed cross-section allows for a complex description of a beam cross-section: one which may include an arbitrary shape, multiple materials, multiple cells, and non-structural mass. The basic idea is to create a two-dimensional finite element model of the beam cross-section. The meshed cross-section is used in Abaqus/Standard to numerically calculate the properties required to characterize the structural response of the cross-section in a subsequent beam element analysis. The two-dimensional Abaqus/Standard analysis writes the cross-sectional properties to an input-file-ready text file (**jobname.bsp**). In the subsequent Abaqus/Standard or Abaqus/Explicit beam element analysis the beam elements requiring the meshed cross-section properties include the text file **jobname.bsp** as the general beam section data. Once the beam element analysis is complete, the Visualization module of Abaqus/CAE is used to visualize results at preselected points along the beam length or to examine detailed stress and strain results displayed directly on the two-dimensional meshed cross-section.

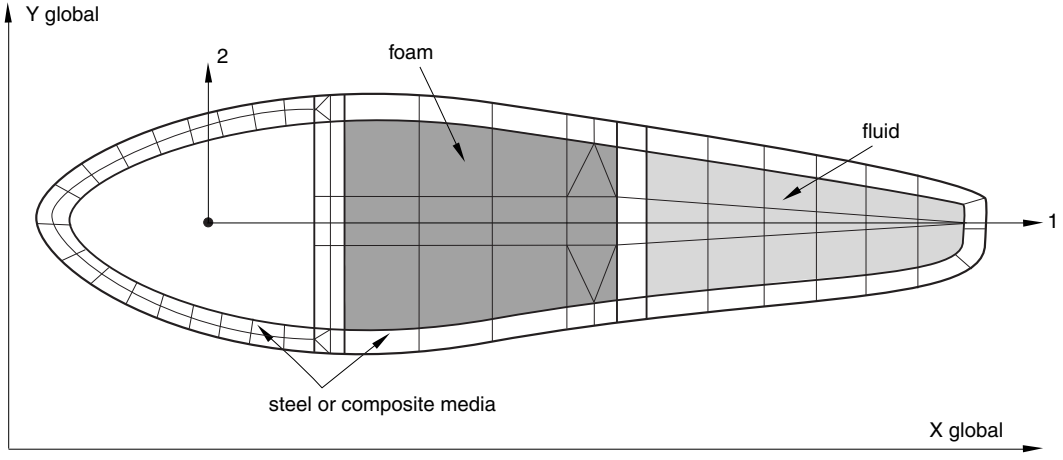


Figure 10.6.1–1 An example of a meshed section profile.

In summary, the procedure for analyzing and postprocessing a beam analysis using a meshed cross-section is as follows:

1. Mesh and analyze a two-dimensional Abaqus/Standard model of the beam cross-section.
2. Use the generated cross-sectional properties in an Abaqus/Standard or Abaqus/Explicit beam analysis.

3. Using the beam analysis results, postprocess from the beam model or the two-dimensional cross-section model.

Meshing and analyzing a two-dimensional model of the beam cross-section

The cross-section is meshed using special-purpose two-dimensional elements: WARP2D3 (3-node triangular) and WARP2D4 (4-node quadrilateral). These elements have one degree of freedom per node representing the value of the out-of-plane warping function (see "Warping element library," Section 28.4.2) and use a solid section definition; no section data are required. Adjacent elements in the cross-sectional mesh must share common nodes; mesh refinement using multi-point constraints is not allowed.

Each element in the cross-sectional mesh can refer to a different elastic material, using either isotropic linear elastic material behavior (see "Defining isotropic elasticity" in "Linear elastic behavior," Section 22.2.1) or orthotropic linear elastic material behavior for warping elements (see "Defining orthotropic elasticity for warping elements" in "Linear elastic behavior," Section 22.2.1). Alternatively, density ("Density," Section 21.2.1) can be the only material property specified, which is useful for modeling non-structural masses such as fuel in a tank.

The model is then analyzed by using the beam section property generation procedure within the step definition. This cross-section analysis will numerically calculate geometric, stiffness, and inertial properties of the section, including the warping function and shear center (see "Meshed beam cross-sections," Section 3.5.6 of the Abaqus Theory Guide) and will write the calculated properties to the **`jobname.bsp`** text file. The contents of this text file, which can be used in a subsequent Abaqus/Standard or Abaqus/Explicit beam analysis, are described in detail below.

Input File Usage: Use the following option to generate beam section properties for a meshed cross-section:

*BEAM SECTION GENERATE

Defining the origin of the cross-section

By default, the origin of the cross-section is the origin of the coordinate system used to define the mesh. You can override this default and input the coordinates of the origin directly or specify that the origin coincides with the shear center or centroid of the cross-section. A nondefault origin is particularly useful when the beam node to be used in the actual analysis does not coincide with the origin of the two-dimensional coordinate system.

Input File Usage: Use both of the following options to input the coordinates of the origin directly:

*BEAM SECTION GENERATE
*SECTION ORIGIN

Use both of the following options to locate the origin at either the centroid or shear center:

*BEAM SECTION GENERATE
*SECTION ORIGIN, ORIGIN=CENTROID or SHEAR CENTER

Requesting output at particular integration points

Output to the output database can be recovered during the actual analysis at particular integration points on the cross-section. Requesting output at a large number of cross-sectional points may degrade performance.

Input File Usage: Use both of the following options to request output at particular integration points:

*BEAM SECTION GENERATE
*SECTION POINTS

Contents of the *jobname*`.bsp` text file

After the analysis to generate the cross-sectional properties completes, the *jobname*`.bsp` text file contains the following lines of data:

(EA), $(EI)_{11}$, $(EI)_{12}$, $(EI)_{22}$, (GJ)
(ρA), $(\rho I)_{11}$, $(\rho I)_{12}$, $(\rho I)_{22}$, x_{1cm}, x_{2cm}
***TRANSVERSE SHEAR STIFFNESS**
$(GA)_{11}$, $(GA)_{22}$, $(GA)_{12}$
***CENTROID**
x_{1c}, x_{2c}
***SHEAR CENTER**
x_{1s}, x_{2s}
***DAMPING, ALPHA=**(C_α)**, BETA=**(C_β)**, COMPOSITE=**(C_c)

The first two lines of data in the *jobname*`.bsp` text file correspond to the section property data for an arbitrarily shaped solid general beam cross-section meshed with warping elements (see "Defining linear section behavior for meshed cross-sections" in "Using a general beam section to define the section behavior," Section 29.3.7).

If you requested output at particular integration points in the two-dimensional cross-section model generation, the *jobname*`.bsp` text file contains the following additional lines:

***SECTION POINTS**
section point label, *2D element number*, *integration point number*
E, G_1, G_2, α, $-(x_1 - x_{1c})$, $(x_2 - x_{2c})$, $(\frac{\partial \Psi}{\partial x_1} - (x_2 - x_{2s}))$, $(\frac{\partial \Psi}{\partial x_2} + (x_1 - x_{1s})$
...

where the set of two data lines is repeated for as many section points as requested.

The cross-sectional property information written to the *jobname*`.bsp` text file will be read into the general beam section definition in the subsequent beam analysis as described below.

Using the generated cross-section properties in a beam analysis

As discussed above, the section properties calculated and stored in the **`jobname.bsp`** text file can be used in an actual beam analysis to define cross-sections for beam elements. The data stored

in **jobname.bsp** correspond to the section property data for an arbitrarily shaped solid general beam cross-section meshed with warping elements (see "Defining linear section behavior for meshed cross-sections" in "Using a general beam section to define the section behavior," Section 29.3.7). Consequently, a simple method of inserting these data is to include the **jobname.bsp** text file in the beam analysis.

Input File Usage: Use the following options to generate section properties in a beam analysis:

∗BEAM GENERAL SECTION, SECTION=MESHED
N_x, N_y, N_z (direction cosines for n_1)
∗INCLUDE, INPUT=*jobname*.**bsp**

Postprocessing from the beam model or the two-dimensional cross-section model

A tickmark contour plot can be used to visualize stress and strain output along the length of the beam model. All stress and strain components requested for the two-dimensional cross-section model generation will be available. Contour plots of stress and strain on the two-dimensional cross-section are also available. The section geometry is read from the output database generated by the two-dimensional cross-section analysis, while the generalized section results are read from the output database generated by the beam analysis.

Initial conditions

Initial conditions are not meaningful when generating beam section properties and are ignored.

Boundary conditions

Boundary conditions are not meaningful when generating beam section properties and are ignored.

Loads

Loads are not meaningful when generating beam section properties and are ignored.

Predefined fields

Temperature and field variables are not allowed for meshed sections.

Material options

Only the following material behaviors are allowed for meshed sections:

- isotropic linear elasticity ("Defining isotropic elasticity" in "Linear elastic behavior," Section 22.2.1)
- orthotropic linear elasticity for warping elements ("Defining orthotropic elasticity for warping elements" in "Linear elastic behavior," Section 22.2.1)
- density ("Density," Section 21.2.1)

Elements

Warping elements must be used to mesh the two-dimensional cross-section. See "Warping elements," Section 28.4.1, for details.

Output

Element output is calculated during the actual beam analysis at the integration points on the meshed cross-section that are selected in the property generation analysis as described above. Output from the property generation analysis is available only on the output database. The Visualization module of Abaqus/CAE can be used to generate contour plots of element output on the cross-section, which requires the output databases from both the section property generation analysis (the cross-section model) and the actual beam analysis. For more information, see the example Python script in "Viewing the analysis of a meshed beam cross-section," Section 9.10.10 of the Abaqus Scripting User's Guide.

Input file template

Generating the cross-section properties in an Abaqus/Standard analysis

```
*HEADING
Meshed cross section
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=WARP2D3, ELSET=TRI
...
*ELEMENT, TYPE=WARP2D4, ELSET=QUAD
...
*SOLID SECTION, MATERIAL=COMPOSITE, ELSET=TRI
*MATERIAL,NAME=COMPOSITE
*ELASTIC, TYPE=TRACTION
E, G1, G2
*DENSITY
...
*SOLID SECTION, MATERIAL=MASS_ONLY, ELSET=QUAD
*MATERIAL, NAME=MASS_ONLY
*DENSITY
...
*STEP
*BEAM SECTION GENERATE
*SECTION ORIGIN
X, Y
```

```
*SECTION POINTS
section point label, element number, integration point number
*END STEP
```

Using the generated cross-section properties in a subsequent Abaqus/Standard or Abaqus/Explicit beam analysis

```
*HEADING
 Beam analysis
...
*NODE, NSET=NALL
...
*ELEMENT, TYPE=B31, ELSET=BEAM1
...
*BEAM GENERAL SECTION, SECTION=MESHED
N_x, N_y, N_z (direction cosines for n_1)
*INCLUDE, INPUT=jobname.bsp
...
*STEP
*DYNAMIC
...
*BOUNDARY
...
*CLOAD
...
*OUTPUT
*ELEMENT OUTPUT
...
*END STEP
```

10.7 Modeling discontinuities as an enriched feature using the extended finite element method

- "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1

10.7.1 MODELING DISCONTINUITIES AS AN ENRICHED FEATURE USING THE EXTENDED FINITE ELEMENT METHOD

Products: Abaqus/Standard Abaqus/CAE Abaqus/Viewer

References

- *ENRICHMENT
- *ENRICHMENT ACTIVATION
- "Using the extended finite element method to model fracture mechanics," Section 31.3 of the Abaqus/CAE User's Guide

Overview

Modeling discontinuities, such as cracks, as an enriched feature:

- is commonly referred to as the extended finite element method (XFEM);
- is an extension of the conventional finite element method based on the concept of partition of unity;
- allows the presence of discontinuities in an element by enriching degrees of freedom with special displacement functions;
- enables the modeling of discontinuities in the fluid pressure field as well as fluid flow within the cracked element surfaces as in hydraulically driven fracture;
- does not require the mesh to match the geometry of the discontinuities;
- is a very attractive and effective way to simulate initiation and propagation of a discrete crack along an arbitrary, solution-dependent path without the requirement of remeshing in the bulk materials;
- can be simultaneously used with the surface-based cohesive behavior approach (see "Surface-based cohesive behavior," Section 37.1.10) or the Virtual Crack Closure Technique (see "Crack propagation analysis," Section 11.4.3), which are best suited for modeling interfacial delamination;
- can be performed using the static procedure (see "Static stress analysis," Section 6.2.2), the implicit dynamic procedure (see "Implicit dynamic analysis using direct integration," Section 6.3.2), the low-cycle fatigue analysis using the direct cyclic approach (see "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7), the geostatic stress field procedure (see "Geostatic stress state," Section 6.8.2), or coupled pore fluid diffusion/stress analysis (see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1);
- can also be used to perform contour integral evaluations for an arbitrary stationary surface crack without the need to refine the mesh around the crack tip;
- allows contact interaction of cracked element surfaces based on a small-sliding formulation;
- allows the application of distributed pressure loads to the cracked element surfaces;
- allows the output of some surface variables on the cracked element surfaces;
- allows both material and geometrical nonlinearity; and

- is available only for first-order stress/displacement solid continuum elements, first-order displacement/pore pressure solid continuum elements, and second-order stress/displacement tetrahedron elements.

Modeling approach

Modeling stationary discontinuities, such as a crack, with the conventional finite element method requires that the mesh conforms to the geometric discontinuities. Therefore, considerable mesh refinement is needed in the neighborhood of the crack tip to capture the singular asymptotic fields adequately. Modeling a growing crack is even more cumbersome because the mesh must be updated continuously to match the geometry of the discontinuity as the crack progresses.

The extended finite element method (XFEM) alleviates the shortcomings associated with meshing crack surfaces. The extended finite element method was first introduced by Belytschko and Black (1999). It is an extension of the conventional finite element method based on the concept of partition of unity by Melenk and Babuska (1996), which allows local enrichment functions to be easily incorporated into a finite element approximation. The presence of discontinuities is ensured by the special enriched functions in conjunction with additional degrees of freedom. However, the finite element framework and its properties such as sparsity and symmetry are retained.

Introducing nodal enrichment functions

For the purpose of fracture analysis, the enrichment functions typically consist of the near-tip asymptotic functions that capture the singularity around the crack tip and a discontinuous function that represents the jump in displacement across the crack surfaces. The approximation for a displacement vector function $\mathbf{u}$ with the partition of unity enrichment is

$$\mathbf{u} = \sum_{I=1}^{N} N_I(x)[\mathbf{u}_I + H(x)\mathbf{a}_I + \sum_{\alpha=1}^{4} F_\alpha(x)\mathbf{b}_I^\alpha],$$

where $N_I(x)$ are the usual nodal shape functions; the first term on the right-hand side of the above equation, $\mathbf{u}_I$, is the usual nodal displacement vector associated with the continuous part of the finite element solution; the second term is the product of the nodal enriched degree of freedom vector, $\mathbf{a}_I$, and the associated discontinuous jump function $H(x)$ across the crack surfaces; and the third term is the product of the nodal enriched degree of freedom vector, $\mathbf{b}_I^\alpha$, and the associated elastic asymptotic crack-tip functions, $F_\alpha(x)$. The first term on the right-hand side is applicable to all the nodes in the model; the second term is valid for nodes whose shape function support is cut by the crack interior; and the third term is used only for nodes whose shape function support is cut by the crack tip.

Figure 10.7.1–1 illustrates the discontinuous jump function across the crack surfaces, $H(x)$, which is given by

$$H(\mathrm{x}) = \begin{cases} 1 & \text{if } (\mathbf{x} - \mathbf{x}^*).\mathbf{n} \geq 0, \\ -1 & \text{otherwise,} \end{cases}$$

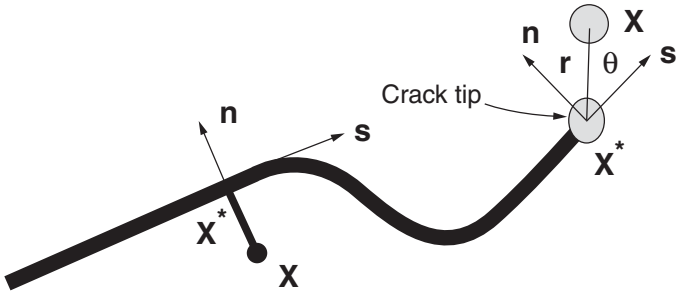


Figure 10.7.1–1 Illustration of normal and tangential coordinates for a smooth crack.

where $\mathbf{x}$ is a sample (Gauss) point, $\mathbf{x}^*$ is the point on the crack closest to $\mathbf{x}$, and $\mathbf{n}$ is the unit outward normal to the crack at $\mathbf{x}^*$.

Figure 10.7.1–1 illustrates the asymptotic crack tip functions in an isotropic elastic material, $F_\alpha(x)$, which are given by

$$F_\alpha(x) = [\sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\theta\sin\frac{\theta}{2}, \sqrt{r}\sin\theta\cos\frac{\theta}{2}],$$

where (r, θ) is a polar coordinate system with its origin at the crack tip and $\theta = 0$ is tangent to the crack at the tip.

These functions span the asymptotic crack-tip function of elasto-statics, and $\sqrt{r}\sin\frac{\theta}{2}$ takes into account the discontinuity across the crack face. The use of asymptotic crack-tip functions is not restricted to crack modeling in an isotropic elastic material. The same approach can be used to represent a crack along a bimaterial interface, impinged on the bimaterial interface, or in an elastic-plastic power law hardening material. However, in each of these three cases different forms of asymptotic crack-tip functions are required depending on the crack location and the extent of the inelastic material deformation. The different forms for the asymptotic crack-tip functions are discussed by Sukumar (2004), Sukumar and Prevost (2003), and Elguedj (2006), respectively.

Accurately modeling the crack-tip singularity requires constantly keeping track of where the crack propagates and is cumbersome because the degree of crack singularity depends on the location of the crack in a non-isotropic material. Therefore, we consider the asymptotic singularity functions only when modeling stationary cracks in Abaqus/Standard. Moving cracks are modeled using one of the two alternative approaches described below.

Modeling moving cracks with the cohesive segments method and phantom nodes

One alternative approach within the framework of XFEM is based on traction-separation cohesive behavior. This approach is used in Abaqus/Standard to simulate crack initiation and propagation. This is a very general interaction modeling capability, which can be used for modeling brittle or ductile fracture. The other crack initiation and propagation capabilities available in Abaqus/Standard are based on cohesive elements ("Defining the constitutive response of cohesive elements using a traction-separation description," Section 32.5.6) or on surface-based cohesive behavior ("Surface-based

cohesive behavior," Section 37.1.10). Unlike these methods, which require that the cohesive surfaces align with element boundaries and the cracks propagate along a set of predefined paths, the XFEM-based cohesive segments method can be used to simulate crack initiation and propagation along an arbitrary, solution-dependent path in the bulk materials, since the crack propagation is not tied to the element boundaries in a mesh. In this case the near-tip asymptotic singularity is not needed, and only the displacement jump across a cracked element is considered. Therefore, the crack has to propagate across an entire element at a time to avoid the need to model the stress singularity.

Phantom nodes, which are superposed on the original real nodes, are introduced to represent the discontinuity of the cracked elements, as illustrated in Figure 10.7.1–2. When the element is intact, each phantom node is completely constrained to its corresponding real node. When the element is cut through by a crack, the cracked element splits into two parts. Each part is formed by a combination of some real and phantom nodes depending on the orientation of the crack. Each phantom node and its corresponding real node are no longer tied together and can move apart.

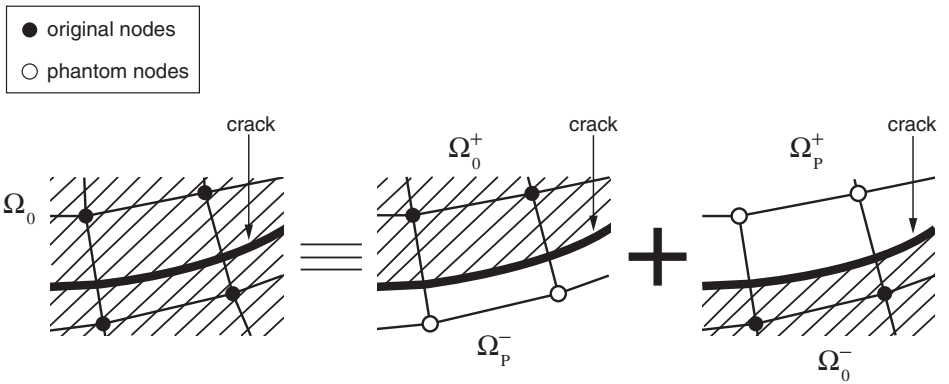


Figure 10.7.1–2 The principle of the phantom node method.

The magnitude of the separation is governed by the cohesive law until the cohesive strength of the cracked element is zero, after which the phantom and the real nodes move independently. To have a set of full interpolation bases, the part of the cracked element that belongs in the real domain, Ω_0, is extended to the phantom domain, Ω_p. Then the displacement in the real domain, Ω_0, can be interpolated by using the degrees of freedom for the nodes in the phantom domain, Ω_p. The jump in the displacement field is realized by simply integrating only over the area from the side of the real nodes up to the crack; i.e., Ω_0^+ and Ω_0^-. This method provides an effective and attractive engineering approach and has been used for simulation of the initiation and growth of multiple cracks in solids by Song (2006) and Remmers (2008). It has been proven to exhibit almost no mesh dependence if the mesh is sufficiently refined.

Modeling hydraulically driven fracture

The cohesive segments method in conjunction with phantom nodes discussed above can also be extended to model hydraulically driven fracture. In this case additional phantom nodes with pore pressure degrees

of freedom are introduced on the edges of each enriched element to model the fluid flow within the cracked element surfaces in conjunction with the phantom nodes that are superposed on the original real nodes to represent the discontinuities of displacement and fluid pressure in a cracked element. The phantom node at each element edge is not activated until the edge is intersected by a crack. The flow patterns of the pore fluid in the cracked elements are shown in Figure 10.7.1–3. The fluid is assumed to be incompressible. The fluid flow continuity, which accounts for both tangential and normal flow within and across the cracked element surfaces as well as the rate of opening of the cracked element surfaces, is maintained. The fluid pressure on the cracked element surfaces contributes to the traction-separation behavior of the cohesive segments in the enriched elements, which enables the modeling of hydraulically driven fracture.

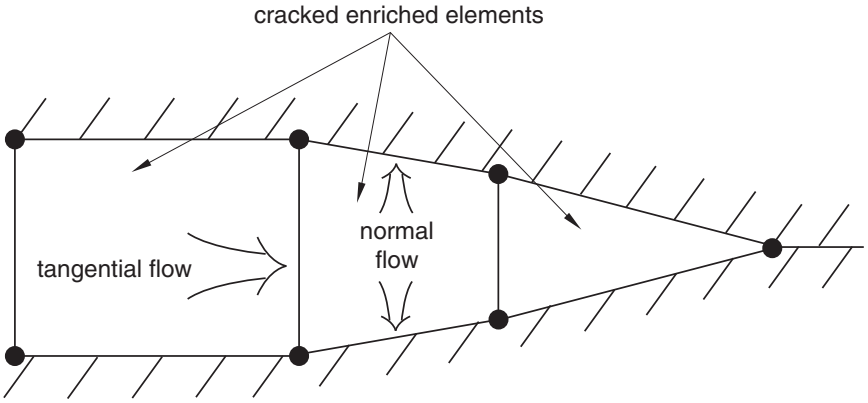


Figure 10.7.1–3 Flow within a cracked element.

Modeling moving cracks based on the principles of linear elastic fracture mechanics (LEFM) and phantom nodes

Another alternative approach to modeling moving cracks within the framework of XFEM is based on the principles of linear elastic fracture mechanics (LEFM). Therefore, it is more appropriate for problems in which brittle crack propagation occurs. Similar to the XFEM-based cohesive segments method described above, the near-tip asymptotic singularity is not considered, and only the displacement jump across a cracked element is considered. Therefore, the crack has to propagate across an entire element at a time to avoid the need to model the stress singularity. The strain energy release rate at the crack tip is calculated based on the modified Virtual Crack Closure Technique (VCCT), which has been used to model delamination along a known and partially bonded surface (see "Crack propagation analysis," Section 11.4.3). However, unlike this method, the XFEM-based LEFM approach can be used to simulate crack propagation along an arbitrary, solution-dependent path in the bulk material without the requirement of a pre-existing crack in the model.

The modeling technique is very similar to the XFEM-based cohesive segment approach described above where phantom nodes are introduced to represent the discontinuity of the cracked element when

the fracture criterion is satisfied. The real node and the corresponding phantom node will separate when the equivalent strain energy release rate exceeds the critical strain energy release rate at the crack tip in an enriched element. The traction is initially carried as equal and opposite forces on the two surfaces of the cracked element. The traction is ramped down linearly over the separation between the two surfaces with the dissipated strain energy equal to either the critical strain energy required to initiate the separation or the critical strain energy required to propagate the crack depending on whether the VCCT or the enhanced VCCT criterion is specified.

Modeling low-cycle fatigue crack propagation based on the principles of LEFM

The XFEM-based LEFM approach can also be used to simulate a discrete crack growth subjected to sub-critical cyclic loading in a low-cycle fatigue analysis using the direct cyclic approach ("Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7). The fracture energy release rates at the crack tips in the enriched elements are calculated based on the above mentioned modified VCCT technique. The onset and crack growth are characterized by using the Paris law, which relates the relative fracture energy release rates to crack growth rates as illustrated in Figure 10.7.1–4. This approach has been used to model progressive delamination under a sub-critical cyclic loading along a known and partially bonded surface (see "Low-cycle fatigue criterion" in "Crack propagation analysis," Section 11.4.3). However, unlike this method, the XFEM-based LEFM approach can be used to simulate fatigue crack propagation along an arbitrary, solution-dependent path in the bulk material.

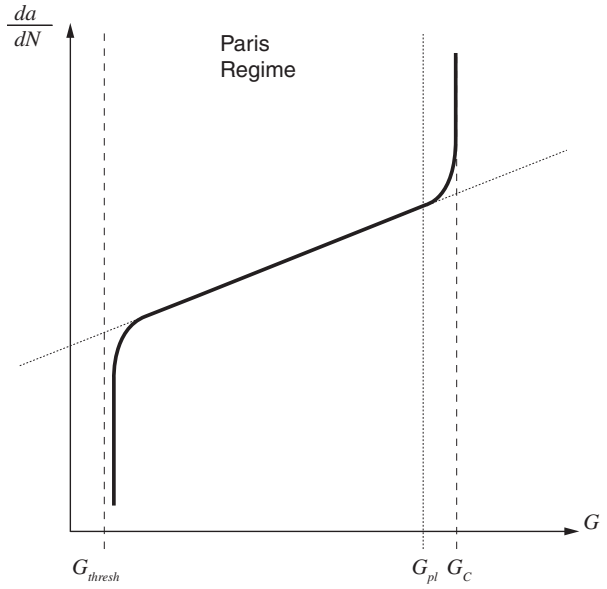


Figure 10.7.1–4 Fatigue crack growth governed by the Paris law.

Using the level set method to describe discontinuous geometry

A key development that facilitates treatment of cracks in an extended finite element analysis is the description of crack geometry, because the mesh is not required to conform to the crack geometry. The level set method, which is a powerful numerical technique for analyzing and computing interface motion, fits naturally with the extended finite element method and makes it possible to model arbitrary crack growth without remeshing. The crack geometry is defined by two almost-orthogonal signed distance functions, as illustrated in Figure 10.7.1–5. The first, ϕ, describes the crack surface, while the second, ψ, is used to construct an orthogonal surface so that the intersection of the two surfaces gives the crack front. $\mathbf{n}^+$ indicates the positive normal to the crack surface; $\mathbf{m}^+$ indicates the positive normal to the crack front. No explicit representation of the boundaries or interfaces is needed because they are entirely described by the nodal data. Two signed distance functions per node are generally required to describe a crack geometry.

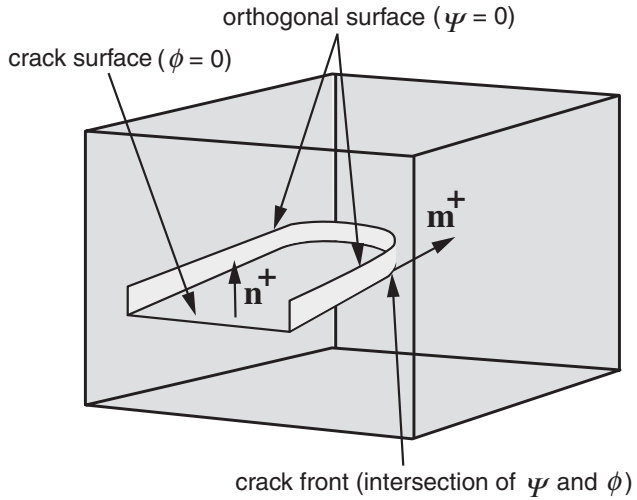


Figure 10.7.1–5 Representation of a nonplanar crack in three dimensions by two signed distance functions ϕ and ψ.

Defining an enriched feature and its properties

You must specify an enriched feature and its properties. One or multiple pre-existing cracks can be associated with an enriched feature. In addition, during an analysis one or multiple cracks can initiate in an enriched feature without any initial defects. However, multiple cracks can nucleate in a single enriched feature only when the damage initiation criterion is satisfied in multiple elements in the same

time increment. Otherwise, additional cracks will not nucleate until all the pre-existing cracks in an enriched feature have propagated through the boundary of the given enriched feature. If several crack nucleations are expected to occur at different locations sequentially during an analysis, multiple enriched features can be specified in the model. Enriched degrees of freedom are activated only when an element is intersected by a crack. Only stress/displacement or displacement/pore pressure solid continuum elements can be associated with an enriched feature.

Input File Usage: *ENRICHMENT

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create→XFEM**

Defining the type of enrichment

You can choose to model an arbitrary stationary crack or a discrete crack propagation along an arbitrary, solution-dependent path. The former requires that the elements around the crack tips are enriched with asymptotic functions to catch the singularity and that the elements intersected by the crack interior are enriched with the jump function across the crack surfaces. The latter infers that crack propagation is modeled with either the cohesive segments method or the linear elastic fracture mechanics approach in conjunction with phantom nodes. However, the options are mutually exclusive and cannot be specified simultaneously in a model.

Input File Usage: Use the following option to specify a crack propagation analysis (default):

*ENRICHMENT, TYPE=PROPAGATION CRACK

Use the following option to specify an analysis with stationary cracks:

*ENRICHMENT, TYPE=STATIONARY CRACK

Abaqus/CAE Usage: Use the following input to specify a crack propagation analysis:

Interaction module: crack editor: toggle on **Allow crack growth**

Use the following input to specify an analysis with stationary cracks:

Interaction module: crack editor: toggle off **Allow crack growth**

Assigning a name to the enriched feature

You must assign a name to an enriched feature, such as a crack. This name can be used in defining the initial location of the crack surfaces, in identifying a crack for contour integral output, in activating or deactivating the crack propagation analysis, and in generating cracked element surfaces.

Input File Usage: *ENRICHMENT, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: **XFEM**: **Name**: *name*

Identifying an enriched region

You must associate the enrichment definition with a region of your model. Only degrees of freedom in elements within these regions are potentially enriched with special functions. The region should consist of elements that are presently intersected by cracks and those that are likely to be intersected by cracks as the cracks propagate.

Input File Usage: *ENRICHMENT, ELSET=*element set name*

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create→**: **XFEM**: **Select the crack domain**: select region

Defining a crack surface

As a crack propagates through the model, a crack surface representing both facets of cracked elements is generated on those enriched elements that are intersected by a crack during the analysis. You must associate the name of an enriched feature with the surface (see "Assigning a name to the enriched feature," above).

The generated crack surface is supported only for the application of distributed pressure loads and the output of some surface variables.

Input File Usage: *SURFACE, TYPE=XFEM

Abaqus/CAE Usage: An XFEM-based crack surface is not supported in Abaqus/CAE.

Defining contact of cracked element surfaces using a small-sliding formulation

When an element is cut by a crack, the compressive behavior of the crack surfaces has to be considered. The formulae that govern behavior are very similar to those used for surface-based small-sliding penalty contact ("Mechanical contact properties: overview," Section 37.1.1).

For an element intersected by a stationary crack or a moving crack with the linear elastic fracture mechanics approach, it is assumed that the elastic cohesive strength of the cracked element is zero. Therefore, compressive behavior of the crack surfaces is fully defined with the above options when the crack surfaces come into contact. For a moving crack with the cohesive segments method, the situation is more complex; traction-separation cohesive behavior as well as compressive behavior of the crack surfaces are involved in a cracked element. In the contact normal direction, the pressure-overclosure relationship governing the compressive behavior between the surfaces does not interact with the cohesive behavior, since they each describe the interaction between the surfaces in a different contact regime. The pressure-overclosure relationship governs the behavior only when the crack is "closed"; the cohesive behavior contributes to the contact normal stress only when the crack is "open" (i.e., not in contact).

If the elastic cohesive stiffness of an element is undamaged in the shear direction, it is assumed that the cohesive behavior is active. Any tangential slip is assumed to be purely elastic in nature and is resisted by the elastic cohesive strength of the element, resulting in shear forces. If damage has been defined, the cohesive contribution to the shear stresses starts degrading with damage evolution. Once maximum degradation has been reached, the cohesive contribution to the shear stresses is zero. The friction model activates and begins contributing to the shear stresses.

Input File Usage: Use the following options to define contact of crack surfaces using a small-sliding formulation:

*ENRICHMENT, INTERACTION=*interaction_property_name*
*SURFACE INTERACTION, NAME=*interaction_property_name*
*SURFACE BEHAVIOR

Abaqus/CAE Usage: Interaction module: crack editor: toggle on **Specify contact property**

Applying cohesive material concepts to XFEM-based cohesive behavior

The formulae and laws that govern the behavior of XFEM-based cohesive segments for a crack propagation analysis are very similar to those used for cohesive elements with traction-separation constitutive behavior ("Defining the constitutive response of cohesive elements using a traction-separation description," Section 32.5.6) and those used for surface-based cohesive behavior ("Surface-based cohesive behavior," Section 37.1.10). The similarities extend to the linear elastic traction-separation model, damage initiation criteria, and damage evolution laws.

Linear elastic traction-separation behavior

The available traction-separation model in Abaqus assumes initially linear elastic behavior followed by the initiation and evolution of damage. The elastic behavior is written in terms of an elastic constitutive matrix that relates the normal and shear stresses to the normal and shear separations of a cracked element.

The nominal traction stress vector, $\mathbf{t}$, consists of the following components: t_n, t_s, and (in three-dimensional problems) t_t, which represent the normal and the two shear tractions, respectively. The corresponding separations are denoted by δ_n, δ_s, and δ_t. The elastic behavior can then be written as

$$\mathbf{t} = \begin{Bmatrix} t_n \\ t_s \\ t_t \end{Bmatrix} = \begin{bmatrix} K_{nn} & 0 & 0 \\ 0 & K_{ss} & 0 \\ 0 & 0 & K_{tt} \end{bmatrix} \begin{Bmatrix} \delta_n \\ \delta_s \\ \delta_t \end{Bmatrix} = \mathbf{K}\boldsymbol{\delta}.$$

The normal and tangential stiffness components will not be coupled: pure normal separation by itself does not give rise to cohesive forces in the shear directions, and pure shear slip with zero normal separation does not give rise to any cohesive forces in the normal direction.

The terms K_{nn}, K_{ss}, and K_{tt} are calculated based on the elastic properties for an enriched element. Specifying the elastic properties of the material in an enriched region is sufficient to define both the elastic stiffness and the traction-separation behavior.

Damage modeling

Damage modeling allows you to simulate the degradation and eventual failure of an enriched element. The failure mechanism consists of two ingredients: a damage initiation criterion and a damage evolution law. The initial response is assumed to be linear as discussed in the previous section. However, once a damage initiation criterion is met, damage can occur according to a user-defined damage evolution law. Figure 10.7.1–6 shows a typical linear and a typical nonlinear traction-separation response with a failure mechanism. The enriched elements do not undergo damage under pure compression.

Damage of the traction-separation response for cohesive behavior in an enriched element is defined within the same general framework used for conventional materials (see "Progressive damage and failure," Section 24.1.1). However, unlike cohesive elements with traction-separation behavior, you do not have to specify the undamaged traction-separation behavior in an enriched element.

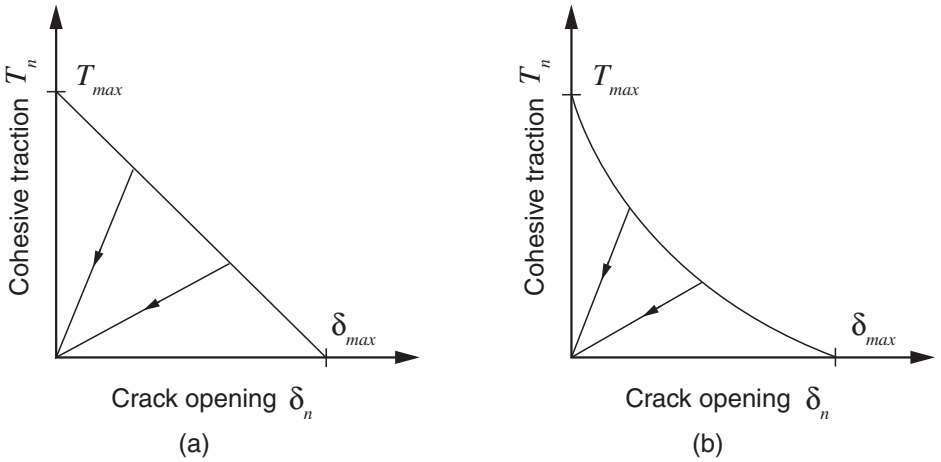


Figure 10.7.1–6 Typical linear (a) and nonlinear (b) traction-separation response.

Crack initiation and direction of crack extension

Crack initiation refers to the beginning of degradation of the cohesive response at an enriched element. The process of degradation begins when the stresses or the strains satisfy specified crack initiation criteria. Crack initiation criteria are available based on the following Abaqus/Standard built-in models:

- the maximum principal stress criterion,
- the maximum principal strain criterion,
- the maximum nominal stress criterion,
- the maximum nominal strain criterion,
- the quadratic traction-interaction criterion, and
- the quadratic separation-interaction criterion.

In addition, a user-defined damage initiation criterion can be specified in user subroutine **`UDMGINI`**.

An additional crack is introduced or the crack length of an existing crack is extended after an equilibrium increment when the fracture criterion, f, reaches the value 1.0 within a given tolerance:

$$1.0 \leq f \leq 1.0 + f_{tol}.$$

You can specify the tolerance f_{tol}. If $f > 1 + f_{tol}$, the time increment is cut back such that the crack initiation criterion is satisfied. The default value of f_{tol} is 0.05.

Input File Usage: *DAMAGE INITIATION, TOLERANCE=f_{tol}

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Tolerance**: f_{tol}

Specifying the crack direction

When the maximum principal stress or the maximum principal strain criterion is specified, the newly introduced crack is always orthogonal to the maximum principal stress/strain direction when the fracture criterion is satisfied. However, when one of the other Abaqus/Standard built-in crack initiation criteria is used, you have to specify if the newly introduced crack will be orthogonal to the element local 1-direction or orthogonal to the element local 2-direction (see "Conventions," Section 1.2.2) when the fracture criterion is satisfied. By default, the crack is orthogonal to the element local 1-direction. If a user-defined damage initiation criterion is specified, the normal direction to the crack plane or the crack line can be defined in user subroutine **`UDMGINI`**.

Input File Usage: Use one of the following options to specify the crack direction when the maximum nominal stress, the maximum nominal strain, the quadratic traction-interaction, or the quadratic separation-interaction criterion is specified:

∗DAMAGE INITIATION, NORMAL DIRECTION=1 (default)

∗DAMAGE INITIATION, NORMAL DIRECTION=2

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, or **Maxs Damage**: **Direction relative to local 1-direction (for XFEM)**: **Normal** or **Parallel**

Maximum principal stress criterion

The maximum principal stress criterion can be represented as

$$\mathrm{f} = \left\{ \frac{\langle \sigma_{max} \rangle}{\sigma^o_{max}} \right\}.$$

Here, σ^o_{max} represents the maximum allowable principal stress. The symbol $\langle\ \rangle$ represents the Macaulay bracket with the usual interpretation (i.e., $\langle \sigma_{max} \rangle = 0$ if $\sigma_{max} < 0$ and $\langle \sigma_{max} \rangle = \sigma_{max}$ if $\sigma_{max} \geq 0$). The Macaulay brackets are used to signify that a purely compressive stress state does not initiate damage. Damage is assumed to initiate when the maximum principal stress ratio (as defined in the expression above) reaches a value of one.

Input File Usage: ∗DAMAGE INITIATION, CRITERION=MAXPS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxps Damage**

Maximum principal strain criterion

The maximum principal strain criterion can be represented as

$$\mathrm{f} = \left\{ \frac{\langle \varepsilon_{max} \rangle}{\varepsilon^o_{max}} \right\}.$$

Here, ε^o_{max} represents the maximum allowable principal strain, and the Macaulay brackets signify that a purely compressive strain does not initiate damage. Damage is assumed to initiate when the maximum principal strain ratio (as defined in the expression above) reaches a value of one.

Input File Usage: *DAMAGE INITIATION, CRITERION=MAXPE

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxpe Damage**

Maximum nominal stress criterion

The maximum nominal stress criterion can be represented as

$$\mathrm{f} = \max\left\{\frac{\langle t_n \rangle}{t^o_n}, \frac{t_s}{t^o_s}, \frac{t_t}{t^o_t}\right\}.$$

The nominal traction stress vector, $\mathbf{t}$, consists of three components (two in two-dimensional problems). t_n is the component normal to the likely cracked surface, and t_s and t_t are the two shear components on the likely cracked surface. Depending on what you specify (see "Specifying the crack direction," above), the likely cracked surface will be orthogonal either to the element local 1-direction or to the element local 2-direction. Here, t^o_n, t^o_s, and t^o_t represent the peak values of the nominal stress. The symbol $\langle\ \rangle$ represents the Macaulay bracket with the usual interpretation. The Macaulay brackets are used to signify that a purely compressive stress state does not initiate damage. Damage is assumed to initiate when the maximum nominal stress ratio (as defined in the expression above) reaches a value of one.

Input File Usage: *DAMAGE INITIATION, CRITERION=MAXS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxs Damage**

Maximum nominal strain criterion

The maximum nominal strain criterion can be represented as

$$\mathrm{f} = \max\left\{\frac{\langle \varepsilon_n \rangle}{\varepsilon^o_n}, \frac{\varepsilon_s}{\varepsilon^o_s}, \frac{\varepsilon_t}{\varepsilon^o_t}\right\}.$$

Damage is assumed to initiate when the maximum nominal strain ratio (as defined in the expression above) reaches a value of one.

Input File Usage: *DAMAGE INITIATION, CRITERION=MAXE

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Maxe Damage**

Quadratic nominal stress criterion

The quadratic nominal stress criterion can be represented as

$$\mathrm{f} = \left\{ \frac{\langle t_n \rangle}{t_n^o} \right\}^2 + \left\{ \frac{t_s}{t_s^o} \right\}^2 + \left\{ \frac{t_t}{t_t^o} \right\}^2.$$

Damage is assumed to initiate when the quadratic interaction function involving the stress ratios (as defined in the expression above) reaches a value of one.

Input File Usage: *DAMAGE INITIATION, CRITERION=QUADS

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quads Damage**

Quadratic nominal strain criterion

The quadratic nominal strain criterion can be represented as

$$\mathrm{f} = \left\{ \frac{\langle \varepsilon_n \rangle}{\varepsilon_n^o} \right\}^2 + \left\{ \frac{\varepsilon_s}{\varepsilon_s^o} \right\}^2 + \left\{ \frac{\varepsilon_t}{\varepsilon_t^o} \right\}^2.$$

Damage is assumed to initiate when the quadratic interaction function involving the nominal strain ratios (as defined in the expression above) reaches a value of one.

Input File Usage: *DAMAGE INITIATION, CRITERION=QUADE

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quade Damage**

User-defined damage initiation criterion

User subroutine **UDMGINI** provides a general capability for implementing a user-defined damage initiation criterion.

You can define several damage initiation mechanisms in user subroutine **UDMGINI**. You represent each damage initiation mechanism by a fracture criterion, f_{indexi}, and its associated normal direction to the crack plane or the crack line. Although you can define several damage initiation mechanisms, the actual damage initiation for an enriched element is governed by the most severe damage initiation mechanism:

$$\mathrm{f} = \max\{f_{index1}, f_{index2}, ...f_{indexn}\}.$$

Damage is assumed to initiate when f, as defined in the expression above, reaches a value of one.

You must specify any material constants that are needed in user subroutine **UDMGINI** as part of a user-defined damage initiation criterion definition.

Input File Usage: Use the following option to define a user-defined damage initiation criterion:

*DAMAGE INITIATION, CRITERION=USER

Use the following option to specify the total number of failure mechanisms in the user-defined damage initiation criterion:

*DAMAGE INITIATION, CRITERION=USER, FAILURE MECHANISMS=n

Use the following option to define properties for a user-defined damage initiation criterion:

*DAMAGE INITIATION, CRITERION=USER, PROPERTIES=*number_of_constants*

Abaqus/CAE Usage: Defining a user-defined damage initiation criterion is not supported in Abaqus/CAE.

Local calculations of the stress and strain fields ahead of the crack tip

An accurate and efficient evaluation of the stress/strain fields ahead of the crack tip is important for both evaluating the crack initiation criterion and computing the crack propagation direction when needed. Abaqus/Standard offers several options for computing these fields.

Centroidal values of stress and strain

By default, the stress/strain computed at the element centroid ahead of the crack tip is used to determine if the damage initiation criterion is satisfied and to determine the crack propagation direction. See Figure 10.7.1–7.

Input File Usage: *DAMAGE INITIATION, POSITION=CENTROID (default)

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position→Centroid**

Computing the stress and strain fields at the crack tip

With a sufficiently refined mesh, the centroidal approximation is accurate and economical. However, if the finite element mesh in the vicinity of the crack tip is coarse relative to the gradients in the stress/strain fields, the default centroidal approximation may not be sufficient. In such cases you can use the stress/strain extrapolated to the crack tip to determine if the damage initiation criterion is satisfied and to determine the crack propagation direction. See Figure 10.7.1–7.

Input File Usage: *DAMAGE INITIATION, POSITION=CRACKTIP

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position→Crack tip**

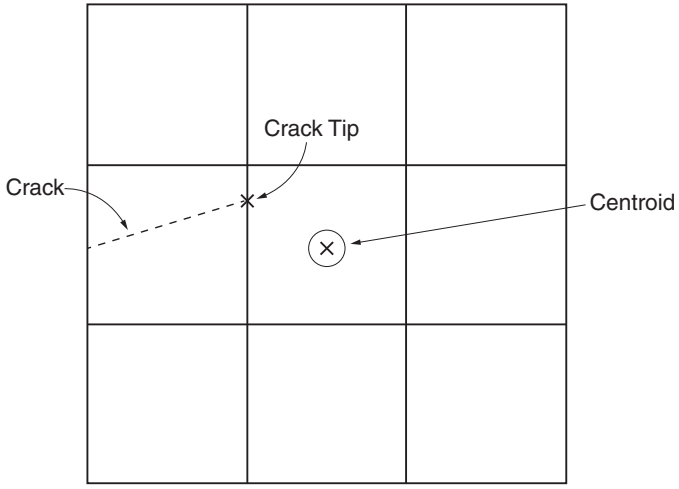


Figure 10.7.1–7 Centroidal and crack tip locations.

Combining crack tip and centroidal calculations

You can also choose to combine the two previous alternatives: you can use the stress/strain values extrapolated to the crack tip to determine if the damage initiation criterion is satisfied, and you can use the stress/strain values at the element centroid to determine the crack propagation direction.

Input File Usage: *DAMAGE INITIATION, POSITION=COMBINED

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Position→Combined**

Nonlocal averaging of the stress and strain fields to improve the accuracy of crack propagation directions

The three options for evaluating the stress and strain fields discussed above are local calculations in the sense that the evaluated fields are local to the single element ahead of the crack tip. In the case of coarse and/or unstructured meshes a nonlocal averaging of the stress and strain fields ahead of the crack tip can lead to a more accurate evaluation of those fields, which can improve the accuracy of the computed propagation directions. See Figure 10.7.1–8.

Input File Usage: *DAMAGE INITIATION, POSITION=NONLOCAL

Abaqus/CAE Usage: Nonlocal averaging of the stress/strain fields ahead of the crack tip is not supported in Abaqus/CAE.

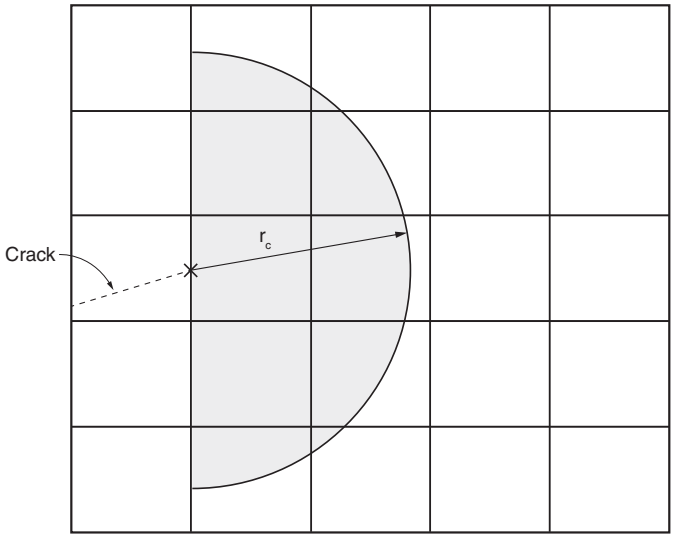


Figure 10.7.1–8 Nonlocal averaging region.

Specifying the region of the model used for nonlocal averaging

To control the range of elements used for nonlocal averaging in the crack direction calculations, you can specify a radius, r_c, within which the elements ahead of the crack tip are included. The default radius is three times the typical element characteristic length in the enriched region.

Input File Usage: *DAMAGE INITIATION, R CRACK DIRECTION=r_c

Abaqus/CAE Usage: Specifying the range of the model for nonlocal averaging is not supported in Abaqus/CAE.

Smoothing the stress/strain fields before averaging

To further improve the nonlocal averaging, you can request an initial smoothing of the stress/strain fields ahead of the crack. In this case Abaqus/Standard averages the field values to element nodes and then interpolates the smoothed fields to the integration points. Once smoothing is complete, the nonlocal averaging is applied. No smoothing is applied by default.

Input File Usage: Use one of the following options:

*DAMAGE INITIATION, SMOOTHING=NONE (default)
*DAMAGE INITIATION, SMOOTHING=NODAL

Abaqus/CAE Usage: Smoothing the stress/strain fields before averaging is not supported in Abaqus/CAE.

Weighting schemes for nonlocal averaging

Abaqus/Standard offers a number of weighting schemes for field smoothing that provide additional control over nonlocal averaging. For example, you may want to give a higher weighting to elements close to the crack tip. You can specify a weight function, ω, to compute the average stress/strain based on the distance from the element integration points to the crack tip, r. By default, a uniform weighting is applied to all elements used for averaging; alternatively, you can use a Gaussian function or a cubic spline function. You can also define a weight function with a user subroutine.

The Gaussian function is represented by:

$$\omega(r) = \frac{1}{(2\pi)^{3/2} r_c^3} exp(\frac{-r^2}{2r_c^2})$$

$$\omega(r) = \begin{cases} 4(\frac{r}{r_c} - 1)(\frac{r}{r_c})^2 + \frac{2}{3}, & 0 < r < \frac{r_c}{2} \\ \frac{4}{3}(1\frac{r}{r_c})^3, & \frac{r_c}{2} \le r \le r_c \\ 0, & \text{otherwise} \end{cases}$$

Input File Usage: Use one of the following options:

∗DAMAGE INITIATION, WEIGHTING METHOD=UNIFORM (default)
∗DAMAGE INITIATION, WEIGHTING METHOD=GAUSS
∗DAMAGE INITIATION, WEIGHTING METHOD=CUBIC SPLINE
∗DAMAGE INITIATION, WEIGHTING METHOD=USER

Abaqus/CAE Usage: Specifying a weighting scheme for nonlocal averaging is not supported in Abaqus/CAE.

Damage evolution

The damage evolution law describes the rate at which the cohesive stiffness is degraded once the corresponding initiation criterion is reached. The general framework for describing the evolution of damage is conceptually similar to that used for damage evolution in surface-based cohesive behavior ("Surface-based cohesive behavior," Section 37.1.10).

A scalar damage variable, D, represents the averaged overall damage at the intersection between the crack surfaces and the edges of cracked elements. It initially has a value of 0. If damage evolution is modeled, D monotonically evolves from 0 to 1 upon further loading after the initiation of damage. The normal and shear stress components are affected by the damage according to

$$t_n = \begin{cases} (1-D)T_n, & T_n \ge 0 \\ T_n, & \text{otherwise (no damage to compressive stiffness);} \end{cases}$$

$$t_s = (1-D)T_s,$$

$$t_t = (1-D)T_t,$$

where T_n, T_s, and T_t are the normal and shear stress components predicted by the elastic traction-separation behavior for the current separations without damage.

To describe the evolution of damage under a combination of normal and shear separations across the interface, an effective separation is defined as

$$\delta_m = \sqrt{\langle \delta_n \rangle^2 + \delta_s^2 + \delta_t^2}.$$

Input File Usage: Use the following option to specify a damage evolution law:

*DAMAGE EVOLUTION

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Maxpe Damage** or **Maxps Damage**: **Suboptions→Damage Evolution**

Use in conjunction with user-defined damage initiation criterion

A separate damage evolution law should be specified for each damage initiation criterion defined in user subroutine **UDMGINI**. Each combination of a damage initiation criterion and a corresponding damage evolution law is referred to as a failure mechanism. Damage will accumulate for only one failure mechanism per element, corresponding to the mechanism whose damage initiation criterion was achieved first.

Input File Usage: Use the following options to specify damage evolution laws for multiple user-defined damage initiation criteria:

*DAMAGE INITIATION, CRITERION=USER, FAILURE MECHANISMS=n
*DAMAGE EVOLUTION, FAILURE INDEX=1
*DAMAGE EVOLUTION, FAILURE INDEX=2
...
*DAMAGE EVOLUTION, FAILURE INDEX=n

Abaqus/CAE Usage: Defining a user-defined damage initiation criterion is not supported in Abaqus/CAE.

Viscous regularization in Abaqus/Standard

Models exhibiting various forms of softening behavior and stiffness degradation often lead to severe convergence difficulties in Abaqus/Standard. Viscous regularization of the constitutive equations defining cohesive behavior in an enriched element can be used to overcome some of these convergence difficulties. Viscous regularization damping causes the tangent stiffness matrix to be positive definite for sufficiently small time increments.

The approximate amount of energy associated with viscous regularization over the whole model is available using output variable ALLVD.

Input File Usage: Use the following option to specify viscous regularization:

*DAMAGE STABILIZATION

Abaqus/CAE Usage: Property module: material editor: **Mechanical→Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: **Suboptions→Damage Stabilization Cohesive**

Defining the constitutive response of fluid flow within the cracked element surfaces

The formulae and laws that govern the behavior of fluid flow within the XFEM-based cracked element surfaces are very similar to those used for fluid flow within the cohesive element gap ("Defining the constitutive response of fluid within the cohesive element gap," Section 32.5.7). The similarities extend to the traction-separation model, damage initiation criteria, damage evolution law, and the fluid flow behavior. The fluid constitutive response comprises the tangential flow within the cracked element surfaces and the normal flow across the cracked element surfaces due to caking or fouling effects in the enriched elements.

Tangential flow

The tangential flow within the cracked element surfaces can be modeled with either a Newtonian or power-law model. By default, there is no tangential flow of pore fluid within the cracked element surfaces. To allow tangential flow, define a gap flow property in conjunction with the pore fluid material definition.

In the case of a Newtonian fluid the volume flow rate density vector is given by the expression

$$\mathbf{q}d = -k_t \nabla p,$$

where k_t is the tangential permeability (the resistance to the fluid flow), ∇p is the pressure gradient along the cracked element surfaces, and d is the opening of the cracked element surfaces.

Abaqus defines the tangential permeability, or the resistance to flow, according to Reynold's equation:

$$k_t = \frac{d^3}{12\mu},$$

where μ is the fluid viscosity and d is the opening of the cracked element surfaces. You can also specify an upper limit on the value of k_t.

In the case of a power law fluid the constitutive relation is defined as

$$\tau = K\dot{\gamma}^{\alpha},$$

where τ is the shear stress, $\dot{\gamma}$ is the shear strain rate, K is the fluid consistency, and α is the power law coefficient. Abaqus defines the tangential volume flow rate density as

$$\mathbf{q}d = -\left(\frac{2\alpha}{1+2\alpha}\right)\left(\frac{1}{K}\right)^{\frac{1}{\alpha}}\left(\frac{d}{2}\right)^{\frac{1+2\alpha}{\alpha}} \|\nabla p\|^{\frac{1-\alpha}{\alpha}} \nabla p,$$

where d is the opening of the cracked element surfaces.

By default, the gap between the cracked element surfaces has an initial opening of 0.002 in both a Newtonian fluid and a power law fluid. However, you can specify this opening directly.

Input File Usage: Use the following option to define the tangential flow in a Newtonian fluid:

*GAP FLOW, TYPE=NEWTONIAN, KMAX

Use the following option to define the tangential flow in a power law fluid:

*GAP FLOW, TYPE=POWER LAW

Use the following option to define the initial gap opening directly:

*SECTION CONTROLS, INITIAL GAP OPENING

Abaqus/CAE Usage: Use the following option to define the tangential flow in a Newtonian fluid:

Property module: material editor: **Other→Pore Fluid→Gap Flow**: Type: **Newtonian**: Toggle on **Maximum Permeability** and enter the value of $k_{\max}$

Use the following option to define the tangential flow in a power law fluid:

Property module: material editor: **Other→Pore Fluid→Gap Flow**: Type: **Power law**

An initial gap opening is not supported in Abaqus/CAE.

Normal flow across the cracked element surfaces

You can permit normal flow by defining a fluid leakoff coefficient for the pore fluid material. This coefficient defines a pressure-flow relationship between the phantom nodes located at the cracked element edges and cracked element surfaces. The fluid leakoff coefficients can be interpreted as the permeability of a finite layer of material on the cracked element surfaces, as shown in Figure 10.7.1–9. The normal flow is defined as

$$q_t = c_t(p_i - p_t)$$

and

$$q_b = c_b(p_i - p_b),$$

where q_t and q_b are the flow rates into the top and bottom surfaces of a cracked element, respectively; p_i is the pressure at the phantom node located at the cracked element edge; and p_t and p_b are the pore pressures on the top and bottom surfaces of a cracked element, respectively. You can optionally define leakoff coefficients as functions of temperature and field variables.

Alternatively, you can use user subroutine **UFLUIDLEAKOFF** to define more complex leakoff behavior, including the ability to define a time accumulated resistance, or fouling, through the use of solution-dependent state variables.

Input File Usage: Use the following option to define the leakoff coefficients:

*FLUID LEAKOFF

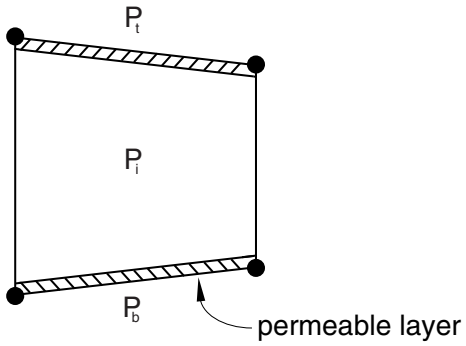


Figure 10.7.1–9 Leakoff coefficient interpretation as a permeable layer.

Use the following option to define leakoff coefficients as functions of temperature and field variables:

*FLUID LEAKOFF, DEPENDENCIES

Use the following option to define more complex leakoff behavior in user subroutine **UFLUIDLEAKOFF**:

*FLUID LEAKOFF, USER

Abaqus/CAE Usage: Use the following option to define the leakoff coefficients:

Property module: material editor: **Other→Pore Fluid→Fluid Leakoff**: Type: **Coefficients**

Use the following option to define leakoff coefficients as functions of temperature and field variables:

Property module: material editor: **Other→Pore Fluid→Fluid Leakoff**: Type: **Coefficients**: Toggle on **Use temperature-dependent data** and select the number of field variables.

Use the following option to define more complex leakoff behavior in user subroutine **UFLUIDLEAKOFF**:

Property module: material editor: **Other→Pore Fluid→Fluid Leakoff**: Type: **User**

Applying the VCCT technique to the XFEM-based LEFM approach

The formulae and laws that govern the behavior of the XFEM-based linear elastic fracture mechanics approach for crack propagation analysis are very similar to those used for modeling delamination along a known and partially bonded surface (see "Crack propagation analysis," Section 11.4.3), where the strain energy release rate at the crack tip is calculated based on the modified Virtual Crack Closure Technique (VCCT). However, unlike this method, the XFEM-based LEFM approach can be used to simulate crack propagation along an arbitrary, solution-dependent path in the bulk material with or without an initial

crack. You complete the definition of the crack propagation capability by defining a fracture-based surface behavior and specifying the fracture criterion in enriched elements.

Crack nucleation and direction of crack extension

By definition, the XFEM-based LEFM approach inherently requires the presence of a crack in the model since it is based on the principles of linear elastic fracture mechanics. The crack can be pre-existing, or it can nucleate during the analysis. If there is no pre-existing crack for a given enriched region, the XFEM-based LEFM approach is not activated until a crack nucleates. The crack nucleation is governed by one of the six built-in stress- or strain-based crack initiation criteria or a user-defined crack initiation criterion discussed in "Crack initiation and direction of crack extension," above. After a crack is nucleated in an enriched region, subsequent propagation of the crack is governed by the XFEM-based LEFM criterion.

Input File Usage: Use the following option to specify the crack nucleation criterion as part of the material definition when there is no pre-existing crack in an enriched region:

*DAMAGE INITIATION, TOLERANCE=f_{tol}

Abaqus/CAE Usage: Property module: material editor: **Mechanical**: **Damage for Traction Separation Laws**: **Quade Damage**, **Maxe Damage**, **Quads Damage**, **Maxs Damage**, **Maxpe Damage**, or **Maxps Damage**: f_{tol}

Specifying when a pre-existing crack will extend

If there is a pre-existing crack in an enriched region, the crack extends after an equilibrium increment when the fracture criterion, f, reaches the value 1.0 within a given tolerance:

$$1.0 \le f \le 1.0 + f_{tol}.$$

You can specify the tolerance f_{tol}. If $f > 1 + f_{tol}$, the time increment is cut back such that the crack extension criterion is satisfied. The default value of f_{tol} is 0.2.

Input File Usage: Use both of the following options:

*SURFACE BEHAVIOR
*FRACTURE CRITERION, TOLERANCE=f_{tol}, TYPE=VCCT

Abaqus/CAE Usage: Interaction module: **Interaction→Property→Create**, **Contact**, **Mechanical→Fracture Criterion**, **Tolerance**: f_{tol}

Specifying the crack propagation direction

You must specify the crack propagation direction when the fracture criterion is satisfied. The crack can extend at a direction normal to the direction of the maximum tangential stress, orthogonal to the element local 1-direction (see "Conventions," Section 1.2.2), or orthogonal to the element local 2-direction. By default, the crack propagates normal to the direction of the maximum tangential stress.

Input File Usage: Use one of the following options to specify the crack direction when the fracture criterion is satisfied:

*FRACTURE CRITERION, NORMAL DIRECTION=MTS (default)

∗FRACTURE CRITERION, NORMAL DIRECTION=1

∗FRACTURE CRITERION, NORMAL DIRECTION=2

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical→Fracture Criterion**: **Direction of crack growth relative to local 1-direction**: **Maximum tangential stress**, **Normal**, or **Parallel**

Mixed mode behavior

Abaqus provides three common mode-mix formulae for computing the equivalent fracture energy release rate G_{equivC}: the BK law, the power law, and the Reeder law models. The choice of model is not always clear in any given analysis; an appropriate model is best selected empirically.

BK law

The BK law model is described in Benzeggagh and Kenane (1996) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}} \right)^{\eta}.$$

To define this model, you must provide G_{IC}, G_{IIC}, and η. This model provides a power law relationship combining energy release rates in Mode I, Mode II, and Mode III into a single scalar fracture criterion.

Input File Usage: ∗FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=BK

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical→Fracture Criterion**: **Mixed mode behavior**: **BK**, and enter the critical energy release rates in the data table

Power law

The power law model is described in Wu and Reuter (1965) by the following formula:

$$\frac{G_{equiv}}{G_{equivC}} = \left(\frac{G_I}{G_{IC}} \right)^{a_m} + \left(\frac{G_{II}}{G_{IIC}} \right)^{a_n} + \left(\frac{G_{III}}{G_{IIIC}} \right)^{a_o}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}, a_m, a_n$, and a_o.

Input File Usage: ∗FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=POWER

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical→Fracture Criterion**: **Mixed mode behavior**: **Power**, and enter the critical energy release rates in the data table

Reeder law

The Reeder law model is described in Reeder et al. (2002) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC}) \left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}} \right)^{\eta} +$$

$$(G_{IIIC} - G_{IIC}) \left(\frac{G_{III}}{G_{II} + G_{III}} \right) \left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}} \right)^{\eta}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}$, and η. The Reeder law is best applied when $G_{IIC} \neq G_{IIIC}$; when $G_{IIC} = G_{IIIC}$, the Reeder law reduces to the BK law. The Reeder law applies only to three-dimensional problems.

Input File Usage: *FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=REEDER

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical→Fracture Criterion**: **Mixed mode behavior**: **Reeder**, and enter the critical energy release rates in the data table

Defining variable critical energy release rates

You can define a VCCT criterion with varying energy release rates by specifying the critical energy release rates at the nodes.

If you indicate that the nodal critical energy rates will be specified, any constant critical energy release rates you specify are ignored and the critical energy release rates are interpolated from the nodes. The critical energy release rates must be defined at all nodes in the enriched region.

Input File Usage: Use both of the following options:

*FRACTURE CRITERION, TYPE=VCCT, NODAL ENERGY RATE
*NODAL ENERGY RATE

Abaqus/CAE Usage: Defining a VCCT criterion with varying energy release rates is not supported in Abaqus/CAE.

Enhanced VCCT criterion

The formulae and laws governing the behavior of the enhanced VCCT criterion are very similar to those used for the VCCT criterion. However, unlike the VCCT criterion, the onset and growth of a crack can be controlled by two different critical fracture energy release rates: G_C and G_C^P. In a general case involving Mode I, II, and III fracture, when the fracture criterion is satisfied; i.e,

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0,$$

the traction on the two surfaces of the cracked element is ramped down over the separation with the dissipated strain energy equal to the critical equivalent strain energy required to propagate the crack, G^P_{equivC}, rather than the critical equivalent strain energy required to initiate the separation, G_{equivC}. The formulae for calculating G^P_{equivC} are identical to those used for G_{equivC} for different mixed-mode fracture criteria.

Input File Usage: Use both of the following options:

*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=ENHANCED VCCT

Abaqus/CAE Usage: Specifying the enhanced VCCT criterion is not supported in Abaqus/CAE.

Low-cycle fatigue criterion based on the principles of LEFM

If you specify the low-cycle fatigue criterion, progressive crack growth at the enriched elements subjected to sub-critical cyclic loading can be simulated. This criterion can be used only in a low-cycle fatigue analysis using the direct cyclic approach ("Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7). A low-cycle fatigue step can be the only step, can follow a general static step, or can be followed by a general static step. You can include multiple low-cycle fatigue analysis steps in a single analysis. If you perform a fatigue analysis in a model without a pre-existing crack, you must precede the fatigue step with a static step that nucleates a crack, as discussed in "Crack nucleation and direction of crack extension."

The onset and fatigue crack growth are characterized by using the Paris law, which relates the relative fracture energy release rate to crack growth rates as illustrated in Figure 10.7.1–4. The fracture energy release rates at the crack tips in the enriched elements are calculated based on the above mentioned VCCT technique.

The Paris regime is bounded by the energy release rate threshold, G_{thresh}, below which there is no consideration of fatigue crack initiation or growth, and the energy release rate upper limit, G_{pl}, above which the fatigue crack will grow at an accelerated rate. G_C is the critical equivalent strain energy release rate calculated based on the user-specified mode-mix criterion and the fracture strength of the bulk material. The formulae for calculating G_C have been provided above for different mixed mode fracture criteria. You can specify the ratio of G_{thresh} over G_C and the ratio of G_{pl} over G_C. The default values are $\frac{G_{thresh}}{G_C} = 0.01$ and $\frac{G_{pl}}{G_C} = 0.85$.

Input File Usage: Use both of the following options:

*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=FATIGUE

Abaqus/CAE Usage: Specifying a low-cycle fatigue criterion is not supported in Abaqus/CAE.

Onset of fatigue crack growth

The onset of fatigue crack growth refers to the beginning of fatigue crack growth at the crack tip in the enriched elements. In a low-cycle fatigue analysis the onset of the fatigue crack growth criterion is characterized by ΔG, which is the relative fracture energy release rate when the structure is loaded between its maximum and minimum values. The fatigue crack growth initiation criterion is defined as

$$f = \frac{N}{c_1 \Delta G^{c_2}} \geq 1.0,$$

where c_1 and c_2 are material constants and N is the cycle number. The enriched elements ahead of the crack tips will not be fractured unless the above equation is satisfied and the maximum fracture energy

release rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value, is greater than G_{thresh}.

Fatigue crack growth using the Paris law

Once the onset of the fatigue crack growth criterion is satisfied at the enriched element, the crack growth rate, da/dN, can be calculated based on the relative fracture energy release rate, ΔG. The rate of the crack growth per cycle is given by the Paris law if $G_{thresh} < G_{max} < G_{pl}$,

$$\frac{da}{dN} = c_3 \Delta G^{c_4},$$

where c_3 and c_4 are material constants.

At the end of cycle N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over an incremental number of cycles, ΔN to $a_{N+\Delta N}$ by fracturing at least one enriched element ahead of the crack tips. Given the material constants c_3 and c_4, combined with the known element length and the likely crack propagation direction $\Delta a_{N\,j} = a_{N+\Delta N} - a_N$ at the enriched elements ahead of the crack tips, the number of cycles necessary to fail each enriched element ahead of the crack tip can be calculated as ΔN_j, where j represents the enriched element ahead of the jth crack tip. The analysis is set up to advance the crack by at least one enriched element after the loading cycle is stabilized. The element with the fewest cycles is identified to be fractured, and its $\Delta N_{min} = min(\Delta N_j)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{N\,min} = min(\Delta a_{N\,j})$. The most critical element is completely fractured with a zero constraint and a zero stiffness at the end of the stabilized cycle. As the enriched element is fractured, the load is redistributed and a new relative fracture energy release rate must be calculated for the enriched elements ahead of the crack tips for the next cycle. This capability allows at least one enriched element ahead of the crack tips to be fractured completely after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

If $G_{max} > G_{pl}$, the enriched elements ahead of the crack tips will be fractured by increasing the cycle number count, dN, by one only.

For information on how to accelerate the low-cycle fatigue analysis and to provide a smooth solution for the crack front, see "Controlling the accuracy of damage extrapolation at the interface elements and at the enriched elements" in "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7.

Viscous regularization for the XFEM-based LEFM approach

The simulation of structures with unstable propagating cracks is challenging and difficult. Nonconvergent behavior may occur from time to time. Localized damping is included for the XFEM-based LEFM approach by using the viscous regularization technique. Viscous regularization damping causes the tangent stiffness matrix of the softening material to be positive for sufficiently small time increments.

Input File Usage: Use one of the following options

*FRACTURE CRITERION, TYPE=VCCT, VISCOSITY=μ

*FRACTURE CRITERION, TYPE=ENHANCED VCCT, VISCOSITY=μ

Abaqus/CAE Usage: Interaction module: contact property editor: **Mechanical→Fracture Criterion**: **Viscosity**: μ

Applying distributed pressure loads to cracked element surfaces

When an element is cut by a crack during the analysis, a XFEM-based crack surface is generated during the analysis (see "Defining a crack surface," above). A distributed pressure load can be applied to the cracked element surfaces.

Input File Usage: Use the following option to define a distributed pressure load to a crack surface:

∗DSLOAD
surface name, P or PNU, *magnitude*

Abaqus/CAE Usage: An XFEM-based crack surface is not supported in Abaqus/CAE.

Specifying the initial location of an enriched feature

Because the mesh is not required to conform to the geometric discontinuities, the initial location of a pre-existing crack must be specified in the model. The level set method is provided for this purpose. Two signed distance functions per node are generally required to describe a crack geometry. The first describes the crack surface, while the second is used to construct an orthogonal surface so that the intersection of the two surfaces gives the crack front (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

The first signed distance function must be either greater or less than zero and cannot be equal to zero. If an initial crack has to be defined at the boundaries of an element, a very small positive or negative value for the first signed distance function must be specified.

Input File Usage: Use the following option to specify the initial location of an enriched feature:

∗INITIAL CONDITIONS, TYPE=ENRICHMENT

Abaqus/CAE Usage: Interaction module: crack editor: **Crack location**: **Select**: select region

Activating and deactivating the enriched feature

The crack propagation capability can be activated or deactivated within the step definition.

Input File Usage: Use the following option to activate the crack propagation capability within the step definition:

∗ENRICHMENT ACTIVATION, NAME=*name*, ACTIVATE=ON (default)

Use the following option to deactivate the crack propagation capability within the step definition:

∗ENRICHMENT ACTIVATION, NAME=*name*, ACTIVATE=OFF

Use the following option to deactivate the crack propagation capability automatically once all the pre-existing cracks (or if there are no pre-existing cracks, all the allowable newly nucleated cracks) have propagated through the boundary of the given enriched feature within the step definition:

∗ENRICHMENT ACTIVATION, NAME=*name*, ACTIVATE=AUTO OFF

Abaqus/CAE Usage: To modify the status of the crack propagation capability in a step, you must first create an **XFEM crack growth** interaction:

Interaction module: **Create Interaction**: select initial step: **XFEM Crack Growth**: select crack: **Interaction manager**: select interaction in step: **Edit**: toggle on/off **Allow crack growth in this step**

Contour integral

When you evaluate the contour integrals using the conventional finite element method ("Contour integral evaluation," Section 11.4.2), you must define the crack front explicitly and specify the virtual crack extension direction in addition to matching the mesh to the cracked geometry. Detailed focused meshes are generally required and obtaining accurate contour integral results for a crack in a three-dimensional curved surface can be cumbersome. The extended finite element in conjunction with the level set method alleviates these shortcomings. The adequate singular asymptotic fields and the discontinuity are ensured by the special enrichment functions in conjunction with additional degrees of freedom. In addition, the crack front and the virtual crack extension direction are determined automatically by the level set signed distance functions.

Input File Usage: Use the following option to obtain contour integral for a named enriched feature with the extended finite element method:

∗CONTOUR INTEGRAL, XFEM, CRACK NAME=*name*

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*

Specifying the enrichment radius

Although XFEM has alleviated the shortcomings associated with refining the mesh in the neighborhood of the crack front due to the added asymptotic fields, you must generate a sufficient number of elements around the crack front to obtain path-independent contours. The group of elements within a small radius from the crack front are enriched and become involved in the contour integral calculations. The default enrichment radius is three times the typical element characteristic length in the enriched area. You must include the elements inside the enrichment radius in the element set used to define the enriched region.

Input File Usage: Use the following option to specify an enrichment radius:

∗ENRICHMENT, ENRICHMENT RADIUS

Abaqus/CAE Usage: Interaction module: crack editor: **Enrichment radius**: **Analysis default** or **Specify**

Procedures

Modeling discontinuities as an enriched feature can be performed using any of the following:

- static analysis (see "Static stress analysis," Section 6.2.2);
- implicit dynamic analysis (see "Implicit dynamic analysis using direct integration," Section 6.3.2); or

- low-cycle fatigue analysis using the direct cyclic approach ("Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7).
- geostatic stress field analysis (see "Geostatic stress state," Section 6.8.2); or
- coupled pore fluid diffusion/stress analysis (see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1).

Initial conditions

Initial conditions to identify initial boundaries or interfaces of an enriched feature can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Boundary conditions

Boundary conditions can be applied to any of the displacement or pore pressure degrees of freedom (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Loads

The following types of loading can be prescribed in a model with an enriched feature:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–3) or the pore pressure degree of freedom (8); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

Predefined fields

The following predefined fields can be specified in a model with an enriched feature, as described in "Predefined fields," Section 34.6.1:

- Nodal temperatures (although temperature is not a degree of freedom in stress/displacement elements). The specified temperature affects temperature-dependent critical stress and strain failure criteria.
- The values of user-defined field variables. The specified value affects field-variable-dependent material properties.

Material options

Any of the mechanical constitutive models in Abaqus/Standard, including user-defined materials (defined using user subroutine "UMAT," Section 1.1.44 of the Abaqus User Subroutines Reference Guide) can be used to model the mechanical behavior of the enriched element in a crack propagation analysis. See Part V, "Materials." The inelastic definition at a material point must be used in conjunction with the linear elastic material model ("Linear elastic behavior," Section 22.2.1) or the hypoelastic material model ("Hypoelastic behavior," Section 22.4.1). Only isotropic elastic materials are supported when evaluating the contour integral for a stationary crack.

Elements

Only first-order solid continuum stress/displacement elements, first-order displacement/pore pressure solid continuum elements, and second-order stress/displacement tetrahedron elements can be associated with an enriched feature. For propagating cracks these include bilinear plane strain and plane stress elements, bilinear axisymmetric elements, linear brick elements, linear tetrahedron elements, and second-order tetrahedron elements. For stationary cracks, these include linear brick elements, linear tetrahedron elements, and second-order tetrahedron elements.

For an incompatible mode element, Abaqus/Standard discards the contribution due to the incompatible deformation mode immediately after the element is fractured under a tensile loading. Therefore, the stress level at the cracked element may not return completely to its originally unloaded state even when this cracked element is unloaded completely and the contact of the cracked element surfaces is reestablished.

Output

In addition to the standard output identifiers available in Abaqus ("Abaqus/Standard output variable identifiers," Section 4.2.1), the following nodal, whole element, and surface variables have special meaning for a model with an enriched feature.

Nodal variables:

PHILSM	Signed distance function to describe the crack surface.
PSILSM	Signed distance function to describe the initial crack front.

Whole element variables:

STATUSXFEM	Status of the enriched element. (The status of an enriched element is 1.0 if the element is completely cracked and 0.0 if the element contains no crack. If the element is partially cracked, the value of STATUSXFEM lies between 1.0 and 0.0.)
ENRRTXFEM	All components of strain energy release rate when linear elastic fracture mechanics with the extended finite element method is used.
LOADSXFEM	Distributed pressure loads applied to the crack surface.

The following whole element output variables are available only when fluid flow is enabled within the cracked enriched element surfaces:

GFVRXFEM	Gap fluid volume rate of the enriched element.
CRDCUTXFEM	Crack midpoint coordinates at the element edges of the enriched element.

PFOPENXFEM	Fracture opening of the enriched element.
PFOPENXFEMCOMP	Fracture opening at the element edges of the enriched element.
PORPRES	Fluid pressure of the enriched element.
PORPRESCOMP	Fluid pressure at the element edges of the enriched element.
LEAKVRTXFEM	Leak-off flow rate at the top of the enriched element.
LEAKVRBXFEM	Leak-off flow rate at the bottom of the enriched element.
ALEAKVRTXFEM	Accumulated leak-off flow volume at the top of the enriched element.
ALEAKVRBXFEM	Accumulated leak-off flow volume at the bottom of the enriched element.

Surface variables (available only for propagating cracks modeled with first-order solid continuum elements):

CRKDISP	Crack opening and relative tangential motions on cracked surfaces in enriched elements.
CSDMG	Damage variable on cracked surfaces in enriched elements.
CRKSTRESS	Remaining residual pressure and tangential shear stresses on cracked surfaces in enriched elements.

The following surface output variables are available only when fluid flow is enabled within the cracked enriched element surfaces:

GFVR	Fluid volume rate within the cracked surfaces in the enriched element.
PORPRES	Pore pressure within the cracked surfaces in the enriched element.
PORPRESURF	Pore pressure on the cracked surfaces in the enriched element.
LEAKVR	Leak-off flow rate on the cracked surfaces in the enriched element.
ALEAKVR	Accumulated leak-off flow volume on the cracked surfaces in the enriched element.

Use of unsymmetric matrix storage and solution

When the pore pressure degrees of freedom are activated in the enriched elements, matrices are unsymmetric; therefore, unsymmetric matrix storage and solution may be needed to improve convergence (see “Matrix storage and solution scheme in Abaqus/Standard” in “Defining an analysis,” Section 6.1.2).

Visualization

A crack can be visualized through the iso-surface for the signed distance function PHILSM.

If a crack cuts through a very tiny corner of an enriched element, the displacements along the crack front in the enriched element may be distorted in rare cases in the Visualization module of Abaqus/CAE

(Abaqus/Viewer) when displaying the contours. The distortion, however, is not present when viewing only the deformed shape.

When an element is cut through by a crack, the cracked element splits into two parts, each part formed by a real domain and a phantom domain, as illustrated in Figure 10.7.1–2. Contour plot integration point values for cracked elements consider contributions from only the real domains in both parts of the cracked elements. However, when you probe cracked elements, only the contribution from the part of the elements containing the real domain Ω_0^- and the phantom domain Ω_p^+ is reported.

When evaluating the contour integrals in a stationary crack, additional integration stations are introduced internally in the elements enriched with singular asymptotic crack-tip fields. However, visualization of the element output variables in those additional integration points is not supported in the Visualization module of Abaqus/CAE (Abaqus/Viewer).

Limitations

The following limitations exist with an enriched feature:

- An enriched element cannot be intersected by more than one crack.
- A crack is not allowed to turn more than 90° in one increment during an analysis.
- Only asymptotic crack-tip fields in an isotropic elastic material are considered for a stationary crack.
- Adaptive remeshing is not supported.
- Composite solid elements are not supported.
- Import analysis is not supported.

Input file template

The following is an example of modeling crack propagation with the XFEM-based cohesive segments method:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=PROPAGATION CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, INTERACTION=INTERACTION
*SURFACE, TYPE=XFEM, NAME=SURF_NAME
Data lines to specify the names of enriched features
*MATERIAL, NAME=STEEL1
...
```

```
*MATERIAL, NAME=STEEL2
*DAMAGE INITIATION, CRITERION=MAXPS, TOLERANCE=0.05
*DAMAGE EVOLUTION, TYPE=ENERGY
```
Data lines to specify the failure mechanism
```
...
*SURFACE INTERACTION, NAME=INTERACTION
*SURFACE BEHAVIOR
```
Data lines to specify the contact of cracked element surfaces
```
...
*STEP
*STATIC
...
*END STEP
*STEP
*STATIC
...

*ENRICHMENT ACTIVATION, TYPE=PROPAGATION CRACK,
NAME=ENRICHMENT, ACTIVATE=OFF
...
*END STEP
```

The following is an example of modeling crack propagation with the XFEM-based LEFM approach:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=PROPAGATION CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, INTERACTION=INTERACTION
*MATERIAL, NAME=STEEL1
...
*MATERIAL, NAME=STEEL2
*DAMAGE INITIATION, CRITERION=MAXPS, TOLERANCE=0.05
```
Data lines to specify the crack nucleation mechanism
```
...
*SURFACE INTERACTION, NAME=INTERACTION
```

```
*SURFACE BEHAVIOR
*FRACTURE CRITERION, TYPE=VCCT, TOLERANCE=0.05,VISCOSITY=0.00001
```
Data lines to specify the crack propagation criterion
```
...
*END STEP
```

The following is an example of calculating contour integrals in stationary cracks with the extended finite element method:

```
*HEADING
...
*NODE, NSET=ALL
...
*ELEMENT, TYPE=C3D8, ELSET=REGULAR
*ELEMENT, TYPE=C3D8, ELSET=ENRICHED
...
*SOLID SECTION, MATERIAL=STEEL1, ELSET=REGULAR
*SOLID SECTION, MATERIAL=STEEL12, ELSET=ENRICHED

*ENRICHMENT, TYPE=STATIONARY CRACK, ELSET=ENRICHED,
NAME=ENRICHMENT, ENRICHMENT RADIUS
*MATERIAL, NAME=STEEL1
...
*MATERIAL, NAME=STEEL2
...
*STEP
*STATIC
...
*CONTOUR INTEGRAL, CRACK NAME=ENRICHMENT, XFEM
*END STEP
```

Additional references

- Belytschko, T., and T. Black, "Elastic Crack Growth in Finite Elements with Minimal Remeshing," International Journal for Numerical Methods in Engineering, vol. 45, pp. 601–620, 1999.
- Benzeggagh, M., and M. Kenane, "Measurement of Mixed-Mode Delamination Fracture Toughness of Unidirectional Glass/Epoxy Composites with Mixed-Mode Bending Apparatus," Composite Science and Technology, vol. 56, p. 439, 1996.
- Elguedj, T., A. Gravouil, and A. Combescure, "Appropriate Extended Functions for X-FEM Simulation of Plastic Fracture Mechanics," Computer Methods in Applied Mechanics and Engineering, vol. 195, pp. 501–515, 2006.

- Melenk, J., and I. Babuska, "The Partition of Unity Finite Element Method: Basic Theory and Applications," Computer Methods in Applied Mechanics and Engineering, vol. 39, pp. 289–314, 1996.
- Reeder, J., S. Kyongchan, P. B. Chunchu, and D. R.. Ambur, "Postbuckling and Growth of Delaminations in Composite Plates Subjected to Axial Compression" 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, vol. 1746, p. 10, 2002.
- Remmers, J. J. C., R. de Borst, and A. Needleman, "The Simulation of Dynamic Crack Propagation using the Cohesive Segments Method," Journal of the Mechanics and Physics of Solids, vol. 56, pp. 70–92, 2008.
- Song, J. H., P. M. A. Areias, and T. Belytschko, "A Method for Dynamic Crack and Shear Band Propagation with Phantom Nodes," International Journal for Numerical Methods in Engineering, vol. 67, pp. 868–893, 2006.
- Sukumar, N., Z. Y. Huang, J.-H. Prevost, and Z. Suo, "Partition of Unity Enrichment for Bimaterial Interface Cracks," International Journal for Numerical Methods in Engineering, vol. 59, pp. 1075–1102, 2004.
- Sukumar, N., and J.-H. Prevost, "Modeling Quasi-Static Crack Growth with the Extended Finite Element Method Part I: Computer Implementation," International Journal for Solids and Structures, vol. 40, pp. 7513–7537, 2003.
- Wu, E. M., and R. C. Reuter Jr., "Crack Extension in Fiberglass Reinforced Plastics," T and M Report, University of Illinois, vol. 275, 1965.

11. Special-Purpose Techniques

Inertia relief 11.1
Mesh modification or replacement 11.2
Geometric imperfections 11.3
Fracture mechanics 11.4
Surface-based fluid modeling 11.5
Mass scaling 11.6
Selective subcycling 11.7
Steady-state detection 11.8

11.1 Inertia relief

- “Inertia relief,” Section 11.1.1

11.1.1 INERTIA RELIEF

Products: Abaqus/Standard Abaqus/CAE

References

- "Distributed loads," Section 34.4.3
- "Defining an analysis," Section 6.1.2
- *INERTIA RELIEF
- "Defining an inertia relief load," Section 16.9.16 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Inertia relief:

- involves balancing externally applied forces on a free or partially constrained body with loads derived from constant rigid body accelerations;
- requires material density or mass and/or rotary inertia values to be specified for computing inertia relief loads;
- can be performed for static, dynamic, and buckling analyses in Abaqus/Standard;
- varies the inertia relief loading with the applied loading in static analysis;
- applies inertia relief load corresponding to the static preload in dynamic analysis;
- can be used to balance applied perturbation loads when used with buckling analysis;
- uses rigid body accelerations consistent with the specified boundary conditions to compute the inertia relief loads;
- can be geometrically linear or nonlinear;
- may require the use of the unsymmetric solver if there are large inertia relief moments in a geometrically nonlinear analysis;
- is an inexpensive alternative to doing a full dynamic free body analysis when applied loads vary slowly compared to the eigenfrequencies of the body; and
- can be used with multiple load cases.

Typical applications

Inertia relief loading can be applied in static ("Static stress analysis," Section 6.2.2), dynamic ("Implicit dynamic analysis using direct integration," Section 6.3.2), and eigenvalue buckling prediction steps ("Eigenvalue buckling prediction," Section 6.2.3).

In a static step the inertia relief loading varies with the applied external loading. An example of using an inertia relief load is modeling a rocket undergoing constant or slowly varying acceleration during lift-off (i.e., a free body subjected to a constant or slowly varying external force) with a static analysis

procedure. The inertia forces experienced by the body are included in the static solution through inertia relief loading that balances the external loading.

In a dynamic step the inertia relief loading is calculated based on the static preload and is held constant during the step. The following is an example of using an inertia relief load in a dynamic analysis procedure: Consider a free body submerged in water and subjected to shock wave loading due to an explosion. A dynamic analysis is needed to compute the transient solution. If it is known that initially the body is stationary under gravity and hydrostatic pressure from the fluid, the gravity load should exactly balance the buoyancy force. However, if the finite element model does not include all the mass existing in the body (for example, ballast), without additional loading, the body would accelerate due to out-of-balance external forces. Applying inertia relief loading exactly balances these unbalanced external loads, placing the body in static equilibrium. The dynamic analysis then provides the transient response of the body to the shock wave loading as deformation of the body relative to its static equilibrium position.

In a buckling analysis the inertia relief load can be applied in the static preload step, in the eigenvalue buckling prediction step, or in both steps. In the eigenvalue buckling prediction step the inertia relief load is calculated based on the perturbation loads. Consider the static analysis rocket example. If we use inertia relief in a buckling analysis of the rocket with the rocket thrust as the perturbation load, we can predict the critical thrust that causes the rocket to buckle.

Basic formulation

In inertia relief the total response, $\{u_t\}$, of the body is written as a combination of a rigid body response due to rigid body motion of a reference point, $\{u_b\}$, and a relative response, $\{u\}$:

$$\{u_t\} = \{u\} + \{u_b\},$$

with corresponding expressions for velocities and accelerations. The reference point is the center of mass except when you must specify the reference point. Then, the finite element approximation to the dynamic equilibrium equation becomes

$$[M]\{\ddot{u}\} + [M]\{\ddot{u}_b\} + \{I\} = \{P\},$$

where $[M]$ is the mass matrix, $\{I\}$ is the internal force vector, and $\{P\}$ is the external force vector. The response of interest in a static analysis involving inertia relief is the rigid body response corresponding to the dynamic motion of the reference point and the static response relative to the rigid body motion. Hence, the relative acceleration term $[M]\{\ddot{u}\}$ drops from the equilibrium equation.

The rigid body response can be expressed in terms of the acceleration of the reference point, $\ddot{z}_j$, and rigid body mode vectors, $\{T\}_j$, $j = 1, 2, ..., 6$ (in three dimensions):

$$\{\ddot{u}_b\} = \sum_{j=1}^{6} \{T\}_j \ddot{z}_j.$$

By definition, $\{T\}_j$ represents the acceleration vector corresponding to a unit imposed acceleration (displacement or rotation) in the ***j***-direction at the reference point. For example, at a node with the usual three displacements and three rotations $\{T\}_j$ is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & (z-z_0) & -(y-y_0) \\ 0 & 1 & 0 & -(z-z_0) & 0 & (x-x_0) \\ 0 & 0 & 1 & (y-y_0) & -(x-x_0) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \\ \hat{e}_4 \\ \hat{e}_5 \\ \hat{e}_6 \end{Bmatrix},$$

where $\hat{e}_j$ is unity; all other $\hat{e}_i$ are zero; ***x***, ***y***, and ***z*** are the coordinates of the node; and x_0, y_0, and z_0 represent the coordinates of the reference point that is the center of rotation. If the system undergoes finite changes in geometry, $\ddot{z}_j$ and $\{T\}_j$ will both be functions of time.

Projecting the dynamic equilibrium equation onto the rigid body modes, we have

$$\sum_{j=1}^{6} m_{ij}\ddot{z}_j = \{T\}_i^T\{P\},$$

where $m_{ij} = \{T\}_i^T[M]\{T\}_j$ is the "rigid body inertia" and $\ddot{z}_j$ is the rigid body acceleration associated with the rigid body mode ***j***. The actual number of rigid body modes will be less than 6 in the presence of symmetry planes as well as for two-dimensional and axisymmetric analyses. Thus, the rigid body response can be evaluated directly from the external loads.

The relative response of the body can be obtained by solving the equilibrium equation with the known inertial term $[M]\{\ddot{u}_b\}$ moved to the right hand side; that is, applied as a body force. The static equilibrium equation then becomes

$$\{I\} = \{P\} + \{P^{ir}\},$$

where $\{P^{ir}\} = -[M]\sum_{j=1}^{6}\{T\}_j\ddot{z}_j$.

In a dynamic analysis involving inertia relief the rigid body mode vectors $\{T\}_j^0$ are calculated in the configuration at the start of the dynamic analysis, and the reference point accelerations $\ddot{z}_j^0$ are calculated to balance the static preloads in this configuration. The relative acceleration term is not dropped, so the dynamic equilibrium equation becomes

$$[M]\{\ddot{u}\} + \{I\} = \{P\} + \{P^{ir}\},$$

where $\{P^{ir}\} = -[M]\sum_{j=1}^{6}\{T\}_j^0\ddot{z}_j^0$. In a geometrically nonlinear analysis the rigid body mode vectors are recomputed during the analysis using the current configuration but the reference point accelerations are kept constant. This keeps the total magnitude of inertia relief loads constant during the analysis but allows the loads to be proportional to the spatial mass distribution, which changes with geometry.

Input File Usage: *INERTIA RELIEF

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**

Inertia relief loading directions

By default, all rigid body motion directions in a model can be loaded by inertia relief loading (in this discussion we use the word "direction" to mean any rigid body translation or rotation). In models with symmetry planes or models that are allowed to move freely in only specific directions, the free directions for which inertia relief loading is applied can be specified. For example, in a three-dimensional analysis with one symmetry plane only three free directions exist—two translations and one rotation. Add an additional symmetry plane and only one free translation remains. A cylinder-piston arrangement is an example where the only free direction considered is motion along the cylinder's axis. In these situations you specify the free directions that are loaded by inertia relief loading by indicating the degrees of freedom.

The case of two free rotation directions is not permitted. For cyclic symmetric models with inertia relief only translation in the *Z*-direction and rotation about the *Z*-direction are considered for computing inertia relief loading.

Input File Usage: ∗INERTIA RELIEF
integer list of global degrees of freedom identifying the free directions

For example, the list 1, 3, 5 implies that translations in the *X*- and *Z*-directions and rotation about the *Y*-axis are free directions.

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: toggle on the degrees of freedom to define the **Free Directions** (the degrees of freedom displayed are dependent on the modeling space)

Defining the free directions in a local coordinate system

If the free directions are not global directions, an orientation can be used to define the local coordinate system to which the integer list of degree of freedom identifiers refers.

Input File Usage: ∗INERTIA RELIEF, ORIENTATION=*orientation_name*
integer list of local degrees of freedom identifying the free directions

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: click **Edit**, and choose a local **CSYS**

Defining free direction combinations that require a user-specified reference point

Not all user-chosen combinations of free directions admit unconstrained rigid body motion; that is, there are certain combinations of free directions for which an additional point is required to define the rigid body motion vectors. For example, in three dimensions the choice 4, 5, 6 corresponds to free rotations about a fixed point. The fixed point must be given to define the rigid body motion vectors. In other examples the free directions include rotation about a fixed axis. Consider a turbine blade rotating about its axis, as shown in Figure 11.1.1–1.

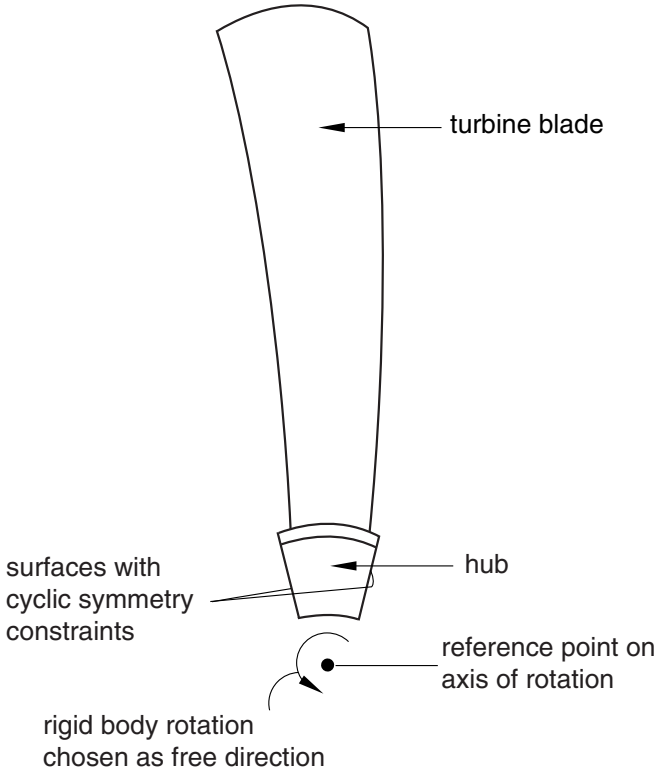


Figure 11.1.1–1 Inertia relief for a turbine blade with rotation about the axis as the only free direction.

To find the angular acceleration of the blade as it rotates under an applied force couple or moment, you should specify the coordinates of the point on the shaft about which the blade is rotating. The free direction combinations for which you must specify a reference point are given in Table 11.1.1–1.

Input File Usage: ∗INERTIA RELIEF, ORIENTATION=*orientation_name*
integer list of local degrees of freedom identifying the free directions
X, Y, Z coordinates of the reference point for defining the rigid body vectors

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: toggle on **Global position of reference point**, and enter the **X**, **Y**, and (if available) **Z** coordinates

Table 11.1.1–1 Free direction combinations requiring a reference point.

Degree of freedom identifiers defining free directions	Reference point definition		
	Fixed rotation point	Point on rotation axis	Point on symmetry line
4, 5, 6	√		
1, 4, 5, 6	√		
2, 4, 5, 6	√		
3, 4, 5, 6	√		
1, 2, 4, 5, 6	√		
1, 3, 4, 5, 6	√		
2, 3, 4, 5, 6	√		
4		√	
5		√	
6		√	
2, 4		√	
3, 4		√	
1, 5		√	
3, 5		√	
1, 6		√	
2, 6		√	
1, 2, 4		√	
1, 2, 5		√	
1, 3, 4		√	
1, 3, 6		√	
2, 3, 5		√	
2, 3, 6		√	
1, 4			√
2, 5			√
3, 6			√

Initial conditions

Initial conditions can be specified in the same way as in static and dynamic analyses without inertia relief loads. If inertia relief is used in the first step in the analysis, these initial conditions form the base state of the body. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Boundary conditions

Boundary conditions are specified in the same way as in analyses without inertia relief loads (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). In theory, a statically determinate set of restraints is needed when inertia relief is used in a static step. By "statically determinate" we mean a set of restraints that restrain all rigid body modes but no deformation modes. Such a set provides a unique displacement solution and ensures that the inertia relief loading exactly balances the user-specified external loading: zero reaction forces with no rigid body motion of the center of mass. Table 11.1.1–2 summarizes the restraint requirements for various cases.

Table 11.1.1–2 Necessary and sufficient statically determinate restraints.

Problem dimensionality	Free directions	Number of required restraints
2D	2 Translations and 1 Rotation	3
Axisymmetric	1 Translation	1
Axisymmetric with twist	1 Translation and 1 Rotation	2
3D	3 Translations and 3 Rotations	6

However, it is not necessary for the user to explicitly specify boundary conditions ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) with inertia relief except in the case of buckling analysis. If no boundary conditions or insufficient boundary conditions are specified, a warning message will be issued and boundary conditions necessary to restrain the rigid body modes will be imposed internally at the point in the model that corresponds to the original location of the reference point. On the other hand, if too many boundary conditions are specified in certain directions, a warning message will be issued to indicate that the reaction forces may be nonzero at the nodes with overspecified boundary conditions. If there are insufficient boundary conditions in certain directions and too many boundary conditions in other directions, the problem will be treated as a combination of these cases.

If a model has no boundary conditions or insufficient boundary conditions, a particular number of numerical singularity warnings can be issued during each equilibrium iteration in the analysis. The displacement solution is postprocessed to remove unconstrained rigid body motion. However, the number of numerical singularities should not exceed the number of unconstrained rigid body modes; any extra numerical singularity messages may indicate other problems.

Similarly, a model with no boundary conditions or insufficient boundary conditions may produce negative eigenvalue messages. If the number of negative eigenvalues at each equilibrium iteration in the analysis does not exceed the maximum reasonable number of numerical singularities associated with the boundary conditions for inertia relief, the results can be trusted, but extra negative eigenvalues may indicate other problems.

If a model contains symmetry planes or is constrained to move freely in specific directions, inertia relief loading should be applied only in those free directions. No boundary conditions should be specified in the free directions; however, sufficient boundary conditions must be specified in the other directions. Any boundary conditions that violate the above requirements will be flagged as an error. An error will also be issued if the combination of free directions includes only two free rotations or if a reference point is required but not specified.

In a buckling analysis, proper boundary conditions are important for getting the correct mode shape. Sufficient boundary conditions must be specified when inertia relief loading is applied in such an analysis. See "Eigenvalue buckling prediction," Section 6.2.3, for details on how to apply boundary conditions in a buckling analysis.

Loads

An analysis that uses inertia relief can include concentrated nodal forces at displacement degrees of freedom (1–6), distributed pressure forces or body forces, and user-defined loading.

Inertia relief loads are used to balance the external loads. They are computed and applied when inertia relief is included in the step definition. The rules for propagating load definitions between steps hold for inertia relief loads. See "Applying loads: overview," Section 34.4.1. The inertia relief loads will not be propagated to steps where inertia relief is not valid for the specified procedure.

If there are large inertia relief moments in a geometrically nonlinear analysis, their contribution to the stiffness matrix may be unsymmetric. In such cases unsymmetric equation solution may improve the computational efficiency (see "Defining an analysis," Section 6.1.2).

Computing inertia relief loads

The nodal force vector corresponding to the inertia relief loads is calculated as follows. The applied loads are projected onto the rigid body modes, $\{T\}_j$. These force and moment components (six components in three dimensions) are used with the "rigid body inertia" to solve for the rigid body accelerations, $\ddot{z}_j$. Only the rigid body acceleration components corresponding to the inertia relief loading directions are nonzero. The nodal force vector is calculated using the assembled mass matrix $[M]$ as

$$\{P^{ir}\} = -[M]\sum_{j=1}^{6}\{T\}_j\ddot{z}_j.$$

Fixed inertia relief loads

You can specify that the inertia relief loads should be held fixed in magnitude and direction at the values calculated at the end of the previous step.

Input File Usage: *INERTIA RELIEF, FIXED

Abaqus/CAE Usage: Load module: **Create Load**: choose **Mechanical** for the **Category** and **Inertia relief** for the **Types for Selected Step**: **Method: Fix at current loading**

Removing inertia relief loads

You can specify that the inertia relief loads that were applied in the previous general analysis step should be removed in the current step.

Input File Usage: *INERTIA RELIEF, REMOVE

Abaqus/CAE Usage: Load module: **Load Manager**: **Deactivate**

Predefined fields

User-defined field variables can be specified in the same way as in static and dynamic analyses without inertia relief loads. See "Predefined fields," Section 34.6.1.

Material options

Any of the mechanical constitutive models that are available in Abaqus/Standard for use in static, dynamic, or buckling analyses can be used with inertia relief (see Part V, "Materials," for details on the material models available in Abaqus/Standard). Since inertia relief loading is calculated using the inertia properties of the model, the density must be specified (see "Density," Section 21.2.1) to define the model's inertia properties.

Elements

Most of the stress/displacement elements that are available in Abaqus/Standard for use in static, dynamic, and buckling analyses (including mass and rotary inertia elements and user elements) can be used. A warning will be issued when the model contains elements that do not have associated mass or inertia (for example, hydrostatic fluid elements and pore pressure elements). An error will be issued if the model contains elements that do not allow finite boundaries (for example, infinite elements and elastic element foundations). Although five degree of freedom shell elements can be used in a step with inertia relief loads, they may cause convergence difficulties if the model has no boundary conditions or insufficient boundary conditions. To improve convergence, these elements should be replaced with other conventional shell elements.

In the case of a substructure you must generate a reduced mass matrix for the substructure (see "Generating a reduced structural damping matrix for a substructure" in "Defining substructures," Section 10.1.2). The reduced mass matrix is included in the global mass matrix of the entire model to compute rigid body accelerations and inertia relief loads. Inertia relief can be used only with substructures in a geometrically linear analysis. An error message is issued if inertia relief is used with substructures in a geometrically nonlinear analysis.

Output

In addition to the usual output variables available in Abaqus/Standard (see "Abaqus/Standard output variable identifiers," Section 4.2.1), the following variables are provided specifically for inertia relief:

Variables for the entire model:

IRX	Current coordinates of the reference point.
IRX*n*	Coordinate *n* of the reference point ($n = 1, 2, 3$).
IRA	Equivalent rigid body acceleration components.
IRA*n*	Component *n* of the equivalent rigid body acceleration ($n = 1, 2, 3$).
IRAR*n*	Component *n* of the equivalent rigid body angular acceleration with respect to the reference point ($n = 1, 2, 3$).
IRF	Inertia relief load corresponding to the equivalent rigid body acceleration.
IRF*n*	Component *n* of the inertia relief load corresponding to the equivalent rigid body acceleration ($n = 1, 2, 3$).
IRM*n*	Component *n* of the inertia relief moment corresponding to the equivalent rigid body angular acceleration with respect to the reference point ($n = 1, 2, 3$).
IRRI	Rotary inertia about the reference point.
IRRI*ij*	ij-component of the rotary inertia about the reference point ($i \le j \le 3$).
IRMASS	Whole model mass.

For most cases inertia relief loads correspond to the product of "rigid body inertia" and the equivalent rigid body acceleration vector. However, when only a few rigid body directions are chosen as free directions for inertia relief, inertia relief loads are computed in all rigid body directions for output purposes, but equivalent rigid body accelerations are computed in only the free directions with the equivalent rigid body angular accelerations computed from the diagonal entries of the "rigid body inertia."

Limitations

You need to be aware of limitations that may be encountered in analyses with inertia relief loads.

Internal boundary conditions and convergence in geometrically linear and nonlinear analysis

In a model containing internal boundary conditions that generate unbalanced internal forces or moments, such as is possible from certain elements (for example, SPRING1, DASHPOT1, SPRING2, DASHPOT2, or GAPUNI elements) or kinematic constraints (for example, coupling constraints, linear constraint equations, multi-point constraints, or surface-based tie constraints), inertia relief loads will not balance these internal forces or moments. If the model contains sufficient boundary conditions, these internal forces or moments will appear as nonzero reaction forces or moments. If the model does not contain sufficient boundary conditions, these internal forces or moments will appear as unconverged residual fluxes in the message file for geometrically linear as well as nonlinear analyses. The model

should be treated as having internal boundary conditions, with the unconverged residuals representing the reaction forces or moments needed to impose the internal boundary conditions. Ideally, the internal boundary conditions should be removed or sufficient boundary conditions should be added to the model.

Unconnected regions and analyses with contact

Inertia relief is not supported for models consisting of multiple unconnected regions, even if contact is defined between them. An exception is when tied contact is defined between the regions. In this case it is the user's responsibility to ensure that different parts are tied in such a way that no rigid body motion is possible between them.

In addition, models involving contact with inertia relief loads may show poor convergence or fail to converge in cases when the surfaces are not in contact or when contact stabilization is used.

Mass and stiffness defined using matrices

Mass and stiffness cannot be defined using matrices in analyses with inertia relief loads.

Assembly loads

An analysis with inertia relief and assembly loads (see "Prescribed assembly loads," Section 34.5.1) may experience poor convergence or no convergence if the model is not properly constrained by boundary conditions.

Input file template

```
*HEADING
…

*DENSITY
Data line to specify material density
*BOUNDARY
Data lines to specify zero-valued boundary conditions
**
*STEP (, NLGEOM) (, PERTURBATION)
Use the NLGEOM parameter to include nonlinear geometric effects;
it will remain active in all subsequent steps.
*STATIC (or *DYNAMIC)
…
*CLOAD and/or *DLOAD
Data lines to specify loads
*INERTIA RELIEF, ORIENTATION=orientation_name
Data lines to specify global (or local, if the ORIENTATION parameter is used) degrees
of freedom that define free directions and to provide coordinates of a reference point
*END STEP
**
```

***STEP**
STATIC(or* ***DYNAMIC)**
…
***INERTIA RELIEF, FIXED or REMOVE**
Include the FIXED parameter to keep inertia relief loads fixed at their current values from the beginning of the step; include the REMOVE parameter to remove inertia relief loads from the beginning of the step.
***END STEP**

11.2 Mesh modification or replacement

- “Element and contact pair removal and reactivation,” Section 11.2.1

11.2.1 ELEMENT AND CONTACT PAIR REMOVAL AND REACTIVATION

Products: Abaqus/Standard Abaqus/CAE

References

- "Removing and reactivating contact pairs" in "Defining contact pairs in Abaqus/Standard," Section 36.3.1
- *MODEL CHANGE
- "Defining a model change interaction," Section 15.13.13 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Element and contact pair removal/reactivation:

- can be used to simulate removal of part of the model, either temporarily or for the remainder of the analysis;
- allows reactivation of elements strain-free or with strain;
- can be used to save computational time when a contact pair is not needed;
- can be used only in general analysis steps; and
- can be used in a restart analysis only if it was used or activated in the original analysis.

Removing elements

You can remove specified elements from the model in a general analysis step. Just prior to the removal step, Abaqus/Standard stores the forces/fluxes that the region to be removed is exerting on the remaining part of the model at the nodes on the boundary between them. These forces are ramped down to zero during the removal step; therefore, the effect of the removed region on the rest of the model is completely absent only at the end of the removal step. The forces are ramped down gradually to ensure that element removal has a smooth effect on the model.

No further element calculations are performed for elements being removed, starting from the beginning of the step in which they are removed. The removed elements remain inactive in subsequent steps unless you reactivate them as described below.

Input File Usage: Use the following option to remove elements from the model:

*MODEL CHANGE, TYPE=ELEMENT, REMOVE

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Deactivated in this step**

Removing elements in transient procedures

Care must be taken in removing elements in transient procedures. The nodal flux that the removed elements apply at the boundary with the rest of the model is ramped down over the step. In transient heat transfer, fully coupled temperature-displacement, or fully coupled thermal-electrical-structural analysis if the fluxes are high and the step is long, this ramping down may have the effect of cooling down or heating up the rest of the body. In dynamic analysis if the forces are high and the step is long, kinetic energy can be imparted to the remaining portion of the model. This problem can be avoided by removing the elements in a very short transient step prior to the rest of the analysis. This step can be done in a single increment.

Reactivating stress/displacement elements

Two distinct types of reactivation are provided for stress/displacement elements (including substructures): strain-free reactivation and reactivation with strain. Strain-free reactivation resets the initial configuration; reactivation with strain does not.

Although elements cannot be created within an analysis, a similar effect can be achieved by creating elements in the model definition, removing them in the first step, and subsequently reactivating them.

Strain-free reactivation

When stress/displacement elements are reactivated in a strain-free state, they become fully active immediately at the moment of reactivation (the start of the step in which they are reactivated). They are reset to an "annealed" state (zero stress, strain, plastic strain, etc.) in the configuration in which they lie at the start of the reactivation step. This configuration depends on whether a small- or large-displacement analysis is being conducted. Alternatively, reactivation in a nonvirgin state can be specified, as described below.

Since these elements are reactivated in a virgin state (i.e., with zero stress), they exert zero nodal forces on the rest of the model. This result allows reactivation to be done immediately, without an adverse effect on the smoothness of the solution.

After reactivation the strains and the deformation gradients are based on the displacements subsequent to the moment of reactivation, rather than on their total displacements. Thus, the current configuration at the start of the reactivation step is the new initial configuration for the element.

This kind of reactivation usually is used to model the creation of an undeformed and unstrained region of the model that is sharing a boundary with another, possibly stressed, deformed region. For example, in tunnel excavation an unstressed tunnel liner is added to line the walls of an already deformed tunnel (see "Stress-free element reactivation," Section 1.1.11 of the Abaqus Example Problems Guide).

Input File Usage: Use the following option to reactivate elements in a strain-free state:

*MODEL CHANGE, ADD=STRAIN FREE (default)

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Small-displacement analysis

In small-displacement analysis the displacements at reactivation are considered to be small; therefore, volume, mass, initial length, and orientation directions do not change.

Large-displacement analysis

In large-displacement analysis the new configuration can be significantly different from the original configuration specified in the model definition. The change in configuration may result from large deformation or rigid body motion. For the nodes of the reactivated elements to be in the correct position upon reactivation, these nodes must be shared by elements that are not removed. Otherwise, the nodes of the removed elements remain at the location occupied at the time of removal. For cases where an enclosed region of material is reactivated, the shared-node restriction may require that a duplicate set of elements whose material properties do not influence the stress solution be defined on top of the removed elements. These duplicate elements provide a means of tracking the position of the nodes of the removed elements.

Upon reactivation an element can have a significantly different volume or mass, so the mass matrix is reformed for the element. Any local orientations applicable to the element are redefined on the new configuration. For shell and membrane elements, however, the thickness of the reactivated elements is the thickness as specified at the start of the analysis by the element's section definition, a nodal thickness definition ("Nodal thicknesses," Section 2.1.3), or an import definition ("Transferring results between Abaqus analyses: overview," Section 9.2.1).

The current normals on structural elements at the moment of reactivation become new initial normals for that element. The current normal is the element's original normal (as specified in the model definition) rotated by the nodal rotation at the moment of reactivation. This scheme preserves the angle between the normals of reactivated elements and those of the elements with which they share nodes. (Usually, this angle should be zero and the normals should be identical, such as when a strain-free layer is added to an already deformed shell or beam. This can be achieved by ensuring that the normals are identical in the model definition.) If the reactivated structural elements share nodes with only non-structural elements (elements that do not provide stiffness to rotational degrees of freedom), duplicate structural elements are required so that the rotational degrees of freedom at the shared nodes will follow the deformation and rigid body motion before reactivation.

In a large-displacement analysis an element that is being reactivated strain free fits into whatever configuration is given by its nodes at the moment of reactivation. You must ensure that this configuration is meaningful and is not severely distorted. Abaqus/Standard will apply geometry checks on the reactivated elements; these checks are the same as the checks that are done in the analysis input file processor. Warnings are printed in the message file if the elements seem inappropriately distorted; and error messages are given if the distortion is severe, in which case the analysis will be stopped. If a geometry check on an element produces a warning or an error message, its current coordinates—and normals if applicable—are printed to the message file for your inspection. The current coordinates can be printed for all elements being reactivated by requesting detailed printout for element removal/reactivation, as explained in "The Abaqus/Standard message file" in "Output," Section 4.1.1.

Reactivating axisymmetric elements

Abaqus/Standard will not stop the analysis if an axisymmetric element has a very small negative radial coordinate at reactivation (if the magnitude of the radial coordinate is less than 10^{-4} times the average element length). In this case a warning is printed, and a radial coordinate of zero is assumed. If the radial coordinate is negative and larger than 10^{-4} times the average element length in magnitude, the analysis will stop.

For axisymmetric-asymmetric elements (SAXA and CAXA) the displacements at reactivation are considered small even in large-displacement analysis because these elements require an axisymmetric original configuration, but the configuration given by the nodes of these elements at reactivation would not, in general, be axisymmetric. Therefore, the original configuration is assumed not to change for these elements.

Reactivating coupled temperature-displacement and coupled thermal-electrical-structural elements

In a fully coupled temperature-displacement analysis and a fully coupled thermal-electrical-structural analysis, continuum elements attain their full mechanical stiffness immediately upon strain-free reactivation; however, to ensure smoothness of the solution, thermal conductivity is ramped up from zero over the step.

Reactivating spring elements and substructures

If spring elements or substructures are reactivated "without strain," the configuration at the moment of reactivation represents the zero-displacement state of the element; the forces in the spring or substructures are based on relative displacements subsequent to the moment of reactivation.

Reactivation with strain

Elements reactivated with strain start in an annealed state unless reactivation in a nonvirgin state is specified, as described below.

The following scheme is implemented for the elements during the reactivation step: Let u^g represent the displacements of the nodes of this element, which are the displacements as shared by the rest of the model or as specified by boundary conditions. In general, these displacements can vary with time over the reactivation step. At any time in the reactivation step Abaqus/Standard enforces displacements, u^e, for the element:

$$u^e = \alpha(t)u^g,$$

where $\alpha(t)$ is a parameter that ramps linearly from 0 to 1 during the step. Thus, during the step the displacements felt by the reactivated elements ramp up to their actual values. To produce a consistent stiffness matrix, the element stiffness is also multiplied by $\alpha(t)$; therefore, the rest of the model experiences the reactivated elements as though their stiffnesses were ramped up during the step.

This ramping up of displacements instead of direct ramping up of element forces ensures that the strain in the element ramps up from zero to the strain given by the displacement of its nodes. This gradual

ramping up of strains is desirable so that the response of history-dependent materials can be integrated gradually.

Subsequent to the end of the reactivation step, the strains in reactivated elements correspond to the displacements of their nodes from their initial configuration, rather than to their displacements since the moment of reactivation. This is appropriate, for example, in the refueling of a nuclear reactor, where the new fuel assembly must conform to the distortion of its old neighbors.

This reactivation scheme does not work for the rotations of shell elements that have five degrees of freedom per node because a total rotation is not stored at those nodes. Consequently, reactivation with strain is not allowed for these elements.

If an element is reactivated with strain after having been previously reactivated strain free, the strains are based on the displacements from the configuration in which the element was reactivated strain free (because this defined the new initial configuration for the element). In this case the u^g in the formula above should be interpreted as the displacement of the node relative to the position in which the element was reactivated strain free.

Input File Usage: Use the following option to reactivate elements with strain:

*MODEL CHANGE, ADD=WITH STRAIN

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**; toggle on **Reactivated elements with strain (when applicable)**

Reactivating elements with rebar

Rebars are reactivated strain free or with strain exactly like the element in which they are defined. The annealing that takes place upon reactivation is also applied to rebars in the model. Reactivation of rebars can also be done in a nonvirgin state.

Reactivating other element types

During reactivation of all element types other than stress/displacement elements, substructures, and contact elements, the nodal forces caused by stress in the element and by distributed loads are scaled by a value that ramps from zero to one during the reactivation step. (The nodal fluxes are scaled similarly for heat transfer elements.) In effect this scaling ramps the element stiffness up from zero during the step; for elements with mass or damping this scaling also ramps up the mass or damping during the step.

During the reactivation step the thermal conductivity of heat transfer elements and the permeability of pore pressure elements are ramped up from zero over the step.

User-defined elements can be removed and reactivated. User subroutine **`UEL`** is not called in steps in which the element is being removed or has already been removed.

Input File Usage: *MODEL CHANGE, ADD

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Removing and reactivating contact pairs

You can remove specified slave and master surfaces from the model in a general analysis step. Contact pair removal and reactivation is explained in "Removing and reactivating contact pairs" in "Defining contact pairs in Abaqus/Standard," Section 36.3.1.

Input File Usage: *MODEL CHANGE, TYPE=CONTACT PAIR, REMOVE or ADD

Abaqus/CAE Usage: Use the following option to remove contact pairs:

Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)** or **Self-contact (Standard)**: toggle off **Active in this step**

Use the following option to reactivate contact pairs:

Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)** or **Self-contact (Standard)**: toggle on **Active in this step**

Removing and reactivating contact elements

Contact elements are removed and reactivated by Abaqus/Standard in the same way as contact pairs, as described in "Removing and reactivating contact pairs" in "Defining contact pairs in Abaqus/Standard," Section 36.3.1.

Input File Usage: *MODEL CHANGE, TYPE=ELEMENT, REMOVE or ADD

Abaqus/CAE Usage: Use the following option to remove contact elements:

Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Deactivated in this step**

Use the following option to reactivate contact elements:

Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Region**, **Activation state of region elements**: **Reactivated in this step**

Modeling issues

In some cases element removal/reactivation may cause numerical problems. The following guidelines can be used to reduce the chance of difficulty:

- If elements are removed in a static stress analysis and this removal leaves a region of the model with an unconstrained rigid body mode, solver problems will occur and the analysis most likely will fail to converge. Therefore, ensure that the remainder of the model is constrained sufficiently.
- If elements that are connected to a contact pair are removed, the contact pair should also be removed to avoid solver problems.
- If all elements attached to a node constrained with a multi-point constraint or a linear constraint equation are being removed, this node should be the dependent node of the multi-point constraint or linear constraint equation.

In some cases element removal may cause Abaqus/Standard to report extra unconnected regions in the message file. These messages can be safely ignored.

Removing or reactivating elements and contact pairs in a restart analysis

Elements or contact pairs can be removed or reactivated in a restart analysis ("Restarting an analysis," Section 9.1.1) only if elements or contact pairs were removed or reactivated in the original analysis. In situations where it is expected that the addition or removal of elements or contact pairs will be required in a restart analysis, but there is no such need in the original analysis, you must activate element or contact pair removal/reactivation in the original analysis. Activating this capability does not add or remove any elements or contact pairs; it only prepares Abaqus/Standard to allow for these changes in a subsequent restart analysis.

Input File Usage: Use the following option to activate element or contact pair removal/reactivation:

*MODEL CHANGE, ACTIVATE

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Model Change**: **Definition**: **Restart**

Procedures

Elements or contact pairs cannot be removed or reactivated in a linear perturbation step (see "General and linear perturbation procedures," Section 6.1.3) or in a static Riks step (see "Unstable collapse and postbuckling analysis," Section 6.2.4). For elements to be absent in such steps, they must have been inactive at the end of the previous general analysis (nonperturbation) step.

Initial conditions

When elements are added back into the model, they are usually assumed to be "annealed"; that is, they have zero plastic strain, creep strain, etc. and zero stress at the start of the step in which they are reactivated. It is possible to reactivate an element so that it starts with a nonzero stress, equivalent plastic strain, and, if relevant, backstress (in a nonvirgin state).

Reactivation in a nonvirgin state

To reactivate elements with nonzero stress, define initial stress conditions (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) to specify the required stress in the model definition. Then the elements must be removed in the first step of the analysis. When reactivated, they will have the initial stress specified. The reactivation is done immediately, so the initial stress (which is applied in full during the first increment) must be self-equilibrating to avoid convergence issues.

If the elements were not removed in the first step, if they were removed again after the first step, or if initial conditions were not specified for them, they will have zero stresses when reactivated.

In a similar manner a material can be reactivated with a nonzero initial equivalent plastic strain and, if relevant, backstress.

When elements are reactivated, any applied initial stress is not displayed in the zero increment frame.

Input File Usage: Use the following option to specify initial stress conditions:

*INITIAL CONDITIONS, TYPE=STRESS

Use the following option to specify initial equivalent plastic strain and backstress:

*INITIAL CONDITIONS, TYPE=HARDENING

Abaqus/CAE Usage: Use the following options to specify the initial stress conditions:

Load module: **Create Predefined Field**: **Step**: **Initial**, choose **Mechanical** for the **Category** and **Stress** for the **Types for Selected Step**

Use the following options to specify the initial equivalent plastic strain and backstress:

Load module: **Create Predefined Field**: **Step**: **Initial**, choose **Mechanical** for the **Category** and **Hardening** for the **Types for Selected Step**

Boundary conditions

The nodal variables of removed elements are not changed when the elements are removed. You can reset these variables by defining a boundary condition while the elements are inactive (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Loads

Distributed and concentrated loads that are applied in an area where elements are removed or reactivated may need to be modified.

Distributed loads

Any distributed loads, fluxes, flows, and foundations specified for inactive elements are also inactive. However, unless you explicitly remove them, records of these loads are still kept and are listed in the data (`.dat`) file as though the elements were still present. Continuation of loads across steps is not affected by removal; on element reactivation unremoved distributed loads are also reactivated.

By default, if a distributed load is applied to an element that is being reactivated in a step, the distributed load magnitude is scaled up linearly from zero to its end-of-step value during the step. If such a load is applied with an amplitude reference, the magnitude value given by the amplitude reference is scaled again by a value that ramps from zero to one throughout the step. This scheme ensures that reactivation has a smooth effect on the solution, even in cases where a distributed load with an amplitude reference on a reactivated element is carried over from a previous step.

Concentrated loads

Concentrated loads or fluxes are not removed when the surrounding elements are removed; therefore, you must ensure that any concentrated loads or fluxes that are carried solely by removed elements are

also removed. Otherwise, a solver problem will occur during the removal step (a force is applied to a degree of freedom with zero stiffness). Concentrated loads or fluxes should be ramped up when they are reintroduced along with reactivated elements.

Predefined fields

The nodal variables of removed elements are not changed directly when the elements are removed. You can reset these variables by defining temperature or other predefined field variables while the elements are inactive (see "Predefined fields," Section 34.6.1). For example, elements that are removed in a stress/displacement analysis can be reintroduced at a different temperature by setting the temperatures at the nodes on these elements to the desired value while the elements are inactive due to removal.

Temperatures

The temperatures at the start of the reactivation step become the initial temperatures for reactivated elements; thermal strains (and, thus, also the thermal stresses) are based on the temperature change subsequent to the instant of reactivation (see "Thermal expansion," Section 26.1.2).

Material options

On annealing, compaction-related quantities—such as the yield stress in hydrostatic compression, p_c, in crushable foam plasticity ("Crushable foam plasticity models," Section 23.3.5); the yield stress in hydrostatic compression, p_b, in cap plasticity ("Modified Drucker-Prager/Cap model," Section 23.3.2); and the void volume fraction, f, in porous metal plasticity ("Porous metal plasticity," Section 23.2.9)—are reset to the values they had at the start of the analysis.

For porous materials the porosity, n, is reset to its initial value and the saturation, s, retains its value from the instant of removal (see "Pore fluid flow properties," Section 26.6.1).

Elements with a user-defined material type can be removed and reactivated; user subroutines **UMAT** and **UMATHT** are not called while the elements are inactive. On reactivation the stresses and strains in user subroutine **UMAT** are set to zero, and conductivity and heat fluxes defined in user subroutine **UMATHT** are ramped up from zero during the reactivation step. Solution-dependent state variables must be reset in user subroutine **UMAT**, **UMATHT**, or **SDVINI**, which will be called on reactivation.

Elements

Removal is not currently supported for rigid, cohesive, gasket, and piezoelectric elements. All other element types in Abaqus/Standard can be removed and reactivated. See "Choosing the appropriate element for an analysis type," Section 27.1.3.

Output

Output is not available for elements or contact surfaces that have been removed. Inactive elements and contact surfaces are visible in Abaqus/CAE.

Input file template

```
*HEADING
…
*STEP
*STATIC
…
** Remove all elements in element set SIDE
*MODEL CHANGE, REMOVE
 SIDE,
** Remove contact pair (SLAVE1, MASTER1)
*MODEL CHANGE, TYPE=CONTACT PAIR, REMOVE
 SLAVE1, MASTER1
…
*END STEP
**
*STEP
*STATIC
…
** Reactivate elements in element set SIDE
*MODEL CHANGE, ADD=STRAIN FREE
 SIDE,
** Reactivate contact pair (SLAVE1, MASTER1)
*MODEL CHANGE, TYPE=CONTACT PAIR, ADD
 SLAVE1, MASTER1
…
*END STEP
```

11.3 Geometric imperfections

- “Introducing a geometric imperfection into a model,” Section 11.3.1

11.3.1 INTRODUCING A GEOMETRIC IMPERFECTION INTO A MODEL

Products: Abaqus/Standard Abaqus/Explicit

References

- "Unstable collapse and postbuckling analysis," Section 6.2.4
- *IMPERFECTION

Overview

A geometric imperfection pattern:

- is generally introduced in a model for a postbuckling load-displacement analysis;
- can be defined as a linear superposition of buckling eigenmodes obtained from a previous eigenvalue buckling prediction or eigenfrequency extraction analysis performed with Abaqus/Standard;
- can be based on the solution obtained from a previous static analysis performed with Abaqus/Standard; or
- can be specified directly.

General postbuckling analysis

In Abaqus/Standard the Riks method ("Unstable collapse and postbuckling analysis," Section 6.2.4) can be used to solve postbuckling problems, both with stable and unstable postbuckling behavior. However, the exact postbuckling problem often cannot be analyzed directly due to the discontinuous response (bifurcation) at the point of buckling. To analyze a postbuckling problem, you must turn it into a problem with continuous response instead of bifurcation, which can be accomplished by introducing a geometric imperfection pattern in the "perfect" geometry so that there is some response in the buckling mode before the critical load is reached.

Introducing geometric imperfections

Imperfections are usually introduced by perturbations in the geometry. Abaqus offers three ways to define an imperfection: as a linear superposition of buckling eigenmodes, from the displacements of a static analysis, or by specifying the node number and imperfection values directly. Only the translational degrees of freedom are modified. Abaqus will then calculate the normals using the usual algorithm based on the perturbed coordinates. Unless the precise shape of an imperfection is known, an imperfection consisting of multiple superimposed buckling modes can be introduced ("Eigenvalue buckling prediction," Section 6.2.3).

The usual approach involves two analysis runs with the same model definition, using Abaqus/Standard to establish the probable collapse modes and either Abaqus/Standard or Abaqus/Explicit to perform the postbuckling analysis:

1. In the first analysis run perform an eigenvalue buckling analysis with Abaqus/Standard on the "perfect" structure to establish probable collapse modes and to verify that the mesh discretizes those modes accurately. Write the eigenmodes in the default global system to the results file as nodal data ("Output to the data and results files," Section 4.1.2).
2. In the second analysis run use Abaqus/Standard or Abaqus/Explicit to introduce an imperfection in the geometry by adding these buckling modes to the "perfect" geometry. The lowest buckling modes are frequently assumed to provide the most critical imperfections, so usually these are scaled and added to the perfect geometry to create the perturbed mesh. The imperfection thus has the form

$$\Delta \boldsymbol{x}_i = \sum_{i=1}^{M} w_i \boldsymbol{\phi}_i,$$

where $\boldsymbol{\phi}_i$ is the i^{th} mode shape and w_i is the associated scale factor.

You must choose the scale factors of the various modes; usually (if the structure is not imperfection sensitive) the lowest buckling mode should have the largest factor. The magnitudes of the perturbations used are typically a few percent of a relative structural dimension such as a beam cross-section or shell thickness.

3. Use either Abaqus/Standard or Abaqus/Explicit to perform the postbuckling analysis.
   - In Abaqus/Standard perform a geometrically nonlinear load-displacement analysis of the structure containing the imperfection using the Riks method. In this way the Riks method can be used to perform postbuckling analyses of "stiff" structures that show linear behavior prior to buckling, if perfect. By performing a load-displacement analysis, other important nonlinear effects, such as material inelasticity or contact, can be included.
   - In Abaqus/Explicit perform a postbuckling analysis on the perturbed structure.

Abaqus imports imperfection data through the user node labels. Abaqus does not check model compatibility between both analysis runs. Node set definitions in the original model and the model with the imperfection may be different. Care must be taken for models in which Abaqus generates additional nodes (for example, the nodes generated for contact surfaces on 20-node brick elements). In such cases you have to ensure that the models for both analysis runs are identical and that the nodal information for the generated nodes is written to the results file.

If the model is defined in terms of an assembly of part instances, the part (**`.prt`**) file from the original analysis is required to read the eigenmodes from the results file. Both the original model and the subsequent model must be defined consistently in terms of an assembly of part instances.

Defining an imperfection based on eigenmode data

To define an imperfection based on the superposition of weighted mode shapes, specify the results file and step from a previous eigenfrequency extraction or eigenvalue buckling prediction analysis. Optionally, you can import eigenmode data for a specified node set.

Input File Usage: ∗IMPERFECTION, FILE=*results_file*, STEP=*step*, NSET=*name*

Defining an imperfection based on static analysis data

To define an imperfection based on the deformed geometry of a previous static analysis ("Unstable collapse and postbuckling analysis," Section 6.2.4), specify the results file and step (and, optionally, the increment number) from a previous static analysis. (If the increment number is not specified, Abaqus will read data from the last increment available for the specified step in the results file.) Optionally, you can import modal data for a specified node set.

Input File Usage: *IMPERFECTION, FILE=*results_file*, STEP=*step*, INC=*inc*, NSET=*name*

Defining an imperfection directly

You can specify the imperfection directly as a table of node numbers and coordinate perturbations in the global coordinate system or, optionally, in a cylindrical or spherical coordinate system. Alternatively, you can read the imperfection data from a separate input file.

Input File Usage: *IMPERFECTION, SYSTEM=*name*, INPUT=*input file*

If no input file is specified, Abaqus assumes that the data follow the option.

Imperfection sensitivity

The response of some structures depends strongly on the imperfections in the original geometry, particularly if the buckling modes interact after buckling occurs. Hence, imperfections based on a single buckling mode tend to yield nonconservative results. By adjusting the magnitude of the scaling factors of the various buckling modes, the imperfection sensitivity of the structure can be assessed. Normally, a number of analyses should be conducted to investigate the sensitivity of a structure to imperfections. Structures with many closely spaced eigenmodes tend to be imperfection sensitive, and imperfections with shapes corresponding to the eigenmode for the lowest eigenvalue may not give the worst case.

The imperfect structure will be easier to analyze if the imperfection is large. If the imperfection is small, the deformation will be quite small (relative to the imperfection) below the critical load. The response will grow quickly near the critical load, introducing a rapid change in behavior.

On the other hand, if the imperfection is large, the postbuckling response will grow steadily before the critical load is reached. In this case the transition into postbuckled behavior will be smooth and relatively easy to analyze.

Input file template

The following example illustrates a postbuckling analysis of a structure with an imperfection defined by a linear superposition of the buckling eigenmodes and involves two analysis runs with the same model definition.

The initial analysis run performs an eigenvalue buckling analysis with Abaqus/Standard to establish the probable collapse modes and writes them to the results file.

```
*HEADING
Initial analysis run to write the buckling modes to the results file
```

```
*NODE
Data lines to define initial "perfect" geometry
…
**
*STEP
*BUCKLE
Data lines to define the number of buckling eigenmodes
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
Data lines to specify the reference load, P_ref
*NODE FILE, GLOBAL=YES, LAST MODE=n
U
*END STEP
```

The second analysis run introduces the imperfection and performs a postbuckling analysis employing the modified Riks method in Abaqus/Standard.

```
*HEADING
Second analysis run to define the imperfection and perform the postbuckling analysis
*NODE
Data lines to define initial "perfect" geometry
…
*IMPERFECTION, FILE=results_file, STEP=step
Data lines specifying the mode number and its associated scale factor
…
**
*STEP, NLGEOM
*STATIC, RIKS
Data line to define incrementation and stopping criteria
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
Data lines to specify reference loading, P_ref
*END STEP
```

An alternative second analysis run introduces the imperfection and performs a postbuckling analysis with Abaqus/Explicit.

```
*HEADING
Second analysis run to define the imperfection and perform the postbuckling analysis
*NODE
Data lines to define initial "perfect" geometry
…
*IMPERFECTION, FILE=results_file, STEP=step
Data lines specifying the mode number and its associated scale factor
…
**
```

```
*STEP
*DYNAMIC, EXPLICIT
```
Data line to define the time period of the step.
```
*CLOAD and/or *DLOAD and/or *DSLOAD and/or *TEMPERATURE
*END STEP
```

11.4 Fracture mechanics

- "Fracture mechanics: overview," Section 11.4.1
- "Contour integral evaluation," Section 11.4.2
- "Crack propagation analysis," Section 11.4.3

11.4.1 FRACTURE MECHANICS: OVERVIEW

Abaqus/Standard provides the following methods for performing fracture mechanics studies:

- **Onset of cracking:** The onset of cracking can be studied in quasi-static problems by using contour integrals ("Contour integral evaluation," Section 11.4.2). The J-integral, the C_t-integral (for creep), the stress intensity factors for both homogeneous materials and interfacial cracks, the crack propagation direction, and the T-stress are calculated by Abaqus/Standard. Contour integrals can be used in two- or three-dimensional problems. In these types of problems focused meshes are generally required and the propagation of a crack is not studied.
- **Crack propagation:** The crack propagation capability allows quasi-static, including low-cycle fatigue, crack growth along predefined paths to be studied ("Crack propagation analysis," Section 11.4.3). Cracks debond along user-defined surfaces. Several crack propagation criteria are available, and multiple cracks can be included in the analysis. Contour integrals can be requested in crack propagation problems.
- **Line spring elements:** Part-through cracks in shells can be modeled inexpensively by using line spring elements in a static procedure, as explained in "Line spring elements for modeling part-through cracks in shells," Section 32.9.1.
- **Extended finite element method (XFEM):** XFEM models a crack as an enriched feature by adding degrees of freedom in elements with special displacement functions ("Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1). XFEM does not require the mesh to match the geometry of the discontinuities. It can be used to simulate initiation and propagation of a discrete crack along an arbitrary, solution-dependent path without the requirement of remeshing. XFEM can also be used to perform contour integral evaluation without the need to refine the mesh around the crack tip.

11.4.2 CONTOUR INTEGRAL EVALUATION

Products: Abaqus/Standard Abaqus/CAE

References

- "Fracture mechanics: overview," Section 11.4.1
- *CONTOUR INTEGRAL
- "Using contour integrals to model fracture mechanics," Section 31.2 of the Abaqus/CAE User's Guide

Overview

Abaqus/Standard offers the evaluation of several parameters for fracture mechanics studies based on either the conventional finite element method or the extended finite element method (XFEM, see "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1):

- the J-integral, which is widely accepted as a quasi-static fracture mechanics parameter for linear material response and, with limitations, for nonlinear material response;
- the C_t-integral, which has an equivalent role to the J-integral in the context of time-dependent creep behavior ("Rate-dependent plasticity: creep and swelling," Section 23.2.4) in a quasi-static step ("Quasi-static analysis," Section 6.2.5);
- the stress intensity factors, which are used in linear elastic fracture mechanics to measure the strength of the local crack-tip fields;
- the crack propagation direction—i.e., the angle at which a preexisting crack will propagate; and
- the T-stress, which represents a stress parallel to the crack faces and is used as an indicator of the extent to which parameters like the J-integral are useful characterizations of the deformation field around the crack.

Contour integrals:

- are output quantities—they do not affect the results;
- can be requested only in general analysis steps;
- can be used only with two-dimensional quadrilateral elements or three-dimensional brick elements when used with the conventional finite element method;
- can be evaluated without requiring a detailed refined mesh around the crack tips when used with XFEM; and
- are currently available only for first-order or second-order tetrahedron and first-order brick elements with isotropic elastic material when used with XFEM.

Contour integral evaluation

Abaqus/Standard offers two different ways to evaluate the contour integral. The first approach is based on the conventional finite element method, which typically requires you to conform the mesh to the cracked geometry, to explicitly define the crack front, and to specify the virtual crack extension direction. Detailed focused meshes are generally required, and obtaining accurate contour integral results for a crack in a three-dimensional curved surface can be quite cumbersome. The extended finite element method (XFEM) alleviates these shortcomings. XFEM does not require the mesh to match the cracked geometry. The presence of a crack is ensured by the special enriched functions in conjunction with additional degrees of freedom. This approach also removes the requirement for explicitly defining the crack front or specifying the virtual crack extension direction when evaluating the contour integral. The data required for the contour integral are determined automatically based on the level set signed distance functions at the nodes in an element (see "Modeling discontinuities as an enriched feature using the extended finite element method," Section 10.7.1).

Several contour integral evaluations are possible at each location along a crack. In a finite element model each evaluation can be thought of as the virtual motion of a block of material surrounding the crack tip (in two dimensions) or surrounding each node along the crack line (in three dimensions). Each block is defined by contours, where each contour is a ring of elements completely surrounding the crack tip or the nodes along the crack line from one crack face to the opposite crack face. These rings of elements are defined recursively to surround all previous contours.

Abaqus/Standard automatically finds the elements that form each ring from the regions defined as the crack tip or crack line. Each contour provides an evaluation of the contour integral. The possible number of evaluations is the number of such rings of elements. You must specify the number of contours to be used in calculating contour integrals. In addition, you must specify the type of contour integral to be calculated, as described below. By default, Abaqus/Standard calculates the J-integral.

You can assign a name to a crack that is used to identify the contour integral values in the data file and in the output database file. The name is also used by Abaqus/CAE to request contour integral output. If you are using the conventional finite element method and do not specify a crack name, by default Abaqus/Standard generates crack numbers that follow the order in which the cracks are defined. If you are using XFEM, you must set the crack name equal to the name assigned to the enriched feature.

Input File Usage: Use the follow option to evaluate the contour integral with the conventional finite element method:

∗CONTOUR INTEGRAL, CRACK NAME=*crack name*, CONTOURS=*n*, TYPE=*integral_type*

Use the following option to evaluate the contour integral with XFEM:

∗CONTOUR INTEGRAL, CRACK NAME=*crack name*, XFEM, CONTOURS=*n*, TYPE=*integral_type*

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: **Name:** *crack name*, **Type: Contour integral** or **XFEM**
Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type:** *integral_type*

The domain integral method

Using the divergence theorem, the contour integral can be expanded into an area integral in two dimensions or a volume integral in three dimensions, over a finite domain surrounding the crack. This domain integral method is used to evaluate contour integrals in Abaqus/Standard. The method is quite robust in the sense that accurate contour integral estimates are usually obtained even with quite coarse meshes. The method is robust because the integral is taken over a domain of elements surrounding the crack and because errors in local solution parameters have less effect on the evaluated quantities such as J, C_t, the stress intensity factors, and the T-stress.

Requesting multiple contour integrals

Contour integrals at several different crack tips in two dimensions or along several different crack lines in three dimensions can be evaluated at any time by repeating the contour integral request as often as needed in the step definition. When you are using the conventional finite element method, you must specify the crack front and the direction of virtual crack extension (or the normal to the crack plane if this normal is constant) for each crack tip or crack line, as described below. When you are using XFEM, you do not need to specify the crack front or the virtual crack extension direction because they will be determined by Abaqus/Standard. However, you must set each crack name equal to the corresponding enriched feature, with each enriched feature consisting of only one crack. In addition, regardless of whether you are using either the conventional finite element method or XFEM, you must specify the number of contours to be calculated for each integral.

The J-integral

The J-integral is usually used in rate-independent quasi-static fracture analysis to characterize the energy release associated with crack growth. It can be related to the stress intensity factor if the material response is linear.

The J-integral is defined in terms of the energy release rate associated with crack advance. For a virtual crack advance $\lambda(s)$ in the plane of a three-dimensional fracture, the energy release rate is given by

$$\bar{J} = \int_A \lambda(s)\mathbf{n} \cdot \mathbf{H} \cdot \mathbf{q}\, dA\,,$$

where dA is a surface element along a vanishing small tubular surface enclosing the crack tip or crack line, $\mathbf{n}$ is the outward normal to dA, and $\mathbf{q}$ is the local direction of virtual crack extension. $\mathbf{H}$ is given by

$$\mathbf{H} = \left(W\mathbf{I} - \boldsymbol{\sigma} \cdot \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \right).$$

For elastic material behavior W is the elastic strain energy; for elastic-plastic or elasto-viscoplastic material behavior W is defined as the elastic strain energy density plus the plastic dissipation, thus representing the strain energy in an "equivalent elastic material." Therefore, the J-integral calculated is suitable only for monotonic loading of elastic-plastic materials.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=J

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: J-integral**

Domain dependence

The *J*-integral should be independent of the domain used provided that the crack faces are parallel to each other, but *J*-integral estimates from different rings may vary because of the approximate nature of the finite element solution. Strong variation in these estimates, commonly called domain dependence or contour dependence, typically indicates an error in the contour integral definition. Gradual variation in these estimates may indicate that a finer mesh is needed or, if plasticity is included, that the contour integral domain does not completely include the plastic zone. If the "equivalent elastic material" is not a good representation of the elastic-plastic material, the contour integrals will be domain independent only if they completely include the plastic zone. Since it is not always possible to include the plastic zone in three dimensions, a finer mesh may be the only solution.

If the first contour integral is defined by specifying the nodes at the crack tip, the first few contours may be inaccurate. To check the accuracy of these contours, you can request more contours and determine the value of the contour integral that appears approximately constant from one contour to the next. The contour integral values that are not approximately equal to this constant should be discarded. In linear elastic problems the first and second contours typically should be ignored as inaccurate.

For some three-dimensional models with an open crack front, the *J*-integral estimates may be inaccurate from the node sets (or elements in the case with XFEM) at the crack front ends. The resolution difficulty is compounded by the skewness of the outmost layer of elements. This accuracy loss is confined only to the contour integrals at the front ends and has no effect on the accuracy of the contour integral values at the neighboring node sets (or elements in the case with XFEM) along the crack front.

Including the effect of a residual stress field on *J*-integral evaluation

A residual stress field often occurs in a structure; for example, as a result of service loads that produce plasticity, a metal forming process in the absence of an anneal treatment, thermal effects, or swelling effects. When the residual stresses are significant, the standard definition of the *J*-integral as described above may lead to a path-dependent value. To ensure its path independence, the *J*-integral evaluation must include an additional term that accounts for the residual stress field. In Abaqus/Standard the problem with a residual stress field is treated as an initial strain problem. If the total strain is written as the sum of mechanical strain, $\boldsymbol{\varepsilon}^m$, and initial strain, $\boldsymbol{\varepsilon}^o$; i.e.,

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^m + \boldsymbol{\varepsilon}^o,$$

a path-independent energy release rate in the presence of a residual stress field is given by

$$\bar{J} = \int_A \lambda(s)\mathbf{n} \cdot (W\mathbf{I} - \boldsymbol{\sigma} \cdot \frac{\partial \mathbf{u}}{\partial \mathbf{x}}) \cdot \mathbf{q}\, dA + \int_V \boldsymbol{\sigma} : \frac{\partial \boldsymbol{\varepsilon}^o}{\partial \mathbf{x}} \cdot \mathbf{q}\, dV\,,$$

where V is the domain volume enclosing the crack tip or crack line, W is defined as the mechanical strain energy density only,

$$W = \int_0^{\boldsymbol{\varepsilon}^m} \boldsymbol{\sigma} : d\boldsymbol{\varepsilon}^m,$$

and ε^{o} remains constant during the entire deformation.

The residual stress field can be specified by reading the stress data from a previous analysis step or by defining an initial condition (see "Defining initial stresses" in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). You specify the step number from which the stress data in the last available increment of the specified step will be considered as residual stresses. If the step number is set equal to zero (default), the residual stress field is defined by the initial condition definition. When XFEM is used, the residual stress field can be defined only with an initial condition definition.

Input File Usage: ∗CONTOUR INTEGRAL, RESIDUAL STRESS STEP=*n*, TYPE=J

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: J-integral**

The C_t-integral

The C_t-integral is supported with the conventional finite element method; however, it is not supported with XFEM.

The C_t-integral can be used for time-dependent creep behavior, where it characterizes creep crack deformation under certain creep conditions, including transient crack growth. C_t is, for example, proportional to the rate of growth of the crack-tip/crack-line creep zone for a stationary crack under small-scale creep conditions. Under steady-state creep conditions, when creep dominates throughout the specimen, C_t becomes path independent and is known as C^*. C_t-integrals should be requested only in a quasi-static step.

The C_t-integral is obtained by replacing the displacements with velocities and the strain energy density with the strain energy rate density in the J-integral expansion. The strain energy rate density is defined as

$$\dot{W} \stackrel{\text{def}}{=} \int_0^{\dot{\boldsymbol{\varepsilon}}} \boldsymbol{\sigma} : d\dot{\boldsymbol{\varepsilon}}\,.$$

$\dot{W}$ is not uniquely defined if multiple deformation mechanisms contribute to the strain rate. However, the creep mechanism will dominate within a zone surrounding a crack tip or crack line, so elastic and plastic contributions to $\dot{W}$ are negligible. The size of that zone depends on the extent of creep relaxation: the zone is initially small but eventually encompasses the entire specimen when steady-state creep is reached. Abaqus/Standard considers only creep in the calculation of $\dot{W}$. Neglecting elastic and plastic strain rates, the strain energy density for the power law creep model with time hardening form in Abaqus/Standard is

$$\dot{W} = \frac{n}{n+1}\, q\, \dot{\bar{\varepsilon}},$$

where n is the power law exponent, q is the equivalent Mises stress, and $\dot{\bar{\varepsilon}}$ is the equivalent uniaxial strain rate.

For the hyperbolic-sine law an analytical expression of $\dot{W}$ is not available. For this law $\dot{W}$ is obtained by numerical integration; a five-point Gauss quadrature scheme gives reasonable accuracy in the range of realistic creep strain rates.

The domain integral method is used for C_t-integrals as described above for J-integrals.

For user-defined creep laws the strain energy rate density must be defined in user subroutine **CREEP**.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=C

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Ct-integral**

Domain dependence

Prior to steady state C_t-integral estimates will exhibit domain dependence, even if the finite element mesh is sufficiently refined, because of the assumption of creep dominance within the domain specified. These C_t estimates should be extrapolated to zero radius to obtain an improved C_t estimate corresponding to a contour shrunk onto the crack tip or crack line (see "C_t-integral evaluation," Section 1.16.6 of the Abaqus Benchmarks Guide).

Including the effect of a residual stress field on C_t-integral evaluation

An additional term is included to account for the residual stress field when calculating the C_t-integral, as described in "Including the effect of a residual stress field on J-integral evaluation."

Input File Usage: *CONTOUR INTEGRAL, RESIDUAL STRESS STEP=*n*, TYPE=C

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: Ct-integral**

The stress intensity factors

The stress intensity factors K_I, K_{II}, and K_{III} are usually used in linear elastic fracture mechanics to characterize the local crack-tip/crack-line stress and displacement fields. They are related to the energy release rate (the J-integral) through

$$J = \frac{1}{8\pi}\mathbf{K}^T \cdot \mathbf{B}^{-1} \cdot \mathbf{K},$$

where $\mathbf{K} = \lfloor K_I, K_{II}, K_{III} \rfloor^T$ are the stress intensity factors and $\mathbf{B}$ is called the pre-logarithmic energy factor matrix. For homogeneous, isotropic materials $\mathbf{B}$ is diagonal, and the above equation simplifies to

$$J = \frac{1}{\bar{E}}(K_I^2 + K_{II}^2) + \frac{1}{2G}K_{III}^2,$$

where $\bar{E} = E$ for plane stress and $\bar{E} = E/(1-\nu^2)$ for plane strain, axisymmetry, and three dimensions. For an interfacial crack between two dissimilar isotropic materials,

$$J = \frac{1-\beta^2}{E^*}(K_I^2 + K_{II}^2) + \frac{1}{2G^*}K_{III}^2,$$

where

$$\frac{1}{E^*} = \frac{1}{2}\Big(\frac{1}{\bar{E}_1} + \frac{1}{\bar{E}_2}\Big), \qquad \frac{1}{G^*} = \frac{1}{2}\Big(\frac{1}{G_1} + \frac{1}{G_2}\Big);$$

$$\beta = \frac{G_1(\kappa_2 - 1) - G_2(\kappa_1 - 1)}{G_1(\kappa_2 + 1) + G_2(\kappa_1 + 1)};$$

$\kappa = 3 - 4\nu$ for plane strain, axisymmetry, and three dimensions; and $\kappa = (3-\nu)/(1+\nu)$ for plane stress. Unlike their analogues in a homogeneous material, K_I and K_{II} are no longer the pure Mode I and Mode II stress intensity factors for an interfacial crack. They are simply the real and imaginary parts of a complex stress intensity factor.

Although the energy release rate is calculated directly in Abaqus/Standard, it is usually not straightforward to compute stress intensity factors from a known ***J***-integral for mixed-mode problems. Abaqus/Standard provides an interaction integral method to compute the stress intensity factors directly for a crack under mixed-mode loading. This capability is available for linear isotropic and anisotropic materials. The theory is described in detail in "Stress intensity factor extraction," Section 2.16.2 of the Abaqus Theory Guide.

In this case the ***J***-integrals calculated from the stress intensity factors will also be output. These ***J***-integral values may be slightly different from those estimated by requesting the ***J***-integral directly, due to the different algorithms used for the calculations.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=K FACTORS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**

Domain dependence

The stress intensity factors have the same domain dependence features as the ***J***-integral.

Including the effect of a residual stress field on stress intensity factor evaluation

An additional term is included to account for the residual stress field when calculating the stress intensity factors, as described in "Including the effect of a residual stress field on ***J***-integral evaluation."

Input File Usage: *CONTOUR INTEGRAL, RESIDUAL STRESS STEP=*n*, TYPE=K FACTORS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: Stress intensity factors**

The crack propagation direction

For homogeneous, isotropic elastic materials the direction of cracking initiation can be calculated using one of the following three criteria: the maximum tangential stress criterion, the maximum energy release rate criterion, or the $K_{II} = 0$ criterion. K_{III} is not taken into account in any of these criteria.

The maximum tangential stress criterion

Using either the condition $\partial\sigma_{\theta\theta}/\partial\theta = 0$ or $\tau_{r\theta} = 0$ (where r and θ are polar coordinates centered at the crack tip in a plane orthogonal to the crack line), we can obtain

$$\hat{\theta} = \cos^{-1}\left(\frac{3K_{II}^2 + \sqrt{K_I^4 + 8K_I^2K_{II}^2}}{K_I^2 + 9K_{II}^2}\right),$$

where the crack propagation angle $\hat{\theta}$ is measured with respect to the crack plane and $\hat{\theta} = 0$ represents the crack propagation in the "straight-ahead" direction. $\hat{\theta} < 0$ if $K_{II} > 0$, while $\hat{\theta} > 0$ if $K_{II} < 0$. The crack propagation angle is measured from $\mathbf{q}$ to $\mathbf{n}$; i.e., it is measured about the direction $-\mathbf{t}$, or counterclockwise measured from $\mathbf{q}$ in Figure 11.4.2–1.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=MTS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: Maximum tangential stress**

The maximum energy release rate criterion

This criterion postulates that a crack initially propagates in the direction that maximizes the energy release rate.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=MERR

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: Maximum energy release rate**

The $K_{II} = 0$ criterion

This criterion assumes that a crack initially propagates in the direction that makes $K_{II} = 0$.

The crack propagation angle $\hat{\theta}$ will be output.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=K FACTORS, DIRECTION=KII0

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: Stress intensity factors**, **Crack initiation criterion: K11=0**

The T-stress

The T-stress component represents a stress parallel to the crack faces at the crack tip. Its magnitude can alter not only the size and shape of the plastic zone but also the stress triaxiality ahead of the crack tip. It is, therefore, a useful indicator of whether measures of the strength of the crack-tip singularity (such as the J-integral or the stress intensity factors) are useful in characterizing a crack under a particular loading. In a linear elastic analysis the T-stress should be calculated using loads equal to the loads in the elastic-plastic analysis. See "T-stress extraction," Section 2.16.3 of the Abaqus Theory Guide, for more information.

Input File Usage: *CONTOUR INTEGRAL, CONTOURS=*n*, TYPE=T-STRESS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Type: T-stress**

Domain dependence

In general, the T-stress has larger domain dependence or contour dependence than the J-integral and the stress intensity factors. Numerical tests suggest that the estimates from the first two rings of elements abutting the crack tip or crack line generally do not provide accurate results. Sufficient contours extending from the crack tip or crack line should be chosen so that the T-stress can be determined to be independent of the number of contours, within engineering accuracy. Particularly for axisymmetric models, the closer the crack tip is to the symmetry axis, the more refined the mesh in the domain should be to achieve path independence of the contour integral.

Including the effect of a residual stress field on T-stress evaluation

An additional term is included to account for the residual stress field when calculating the T-stress, as described in "Including the effect of a residual stress field on J-integral evaluation."

Input File Usage: *CONTOUR INTEGRAL, RESIDUAL STRESS STEP=*n*, TYPE=T-STRESS

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Step for residual stress initialization values:** *step*, **Type: T-stress**

Defining the data required for a contour integral with the conventional finite element method

To request contour integral output with the conventional finite element method, you must define the crack front and specify the virtual crack extension direction.

Defining the crack front

You must specify the crack front; i.e., the region that defines the first contour. Abaqus/Standard uses this region and one layer of elements surrounding it to compute the first contour integral. An additional layer of elements is used to compute each subsequent contour.

The crack front can be equivalent to the crack tip in two dimensions or the crack line in three dimensions; or it can be a larger region surrounding the crack tip or crack line, in which case it must include the crack tip or crack line.

If blunted crack tips are modeled, the crack front should include all the nodes going from one crack face to the other that would collapse onto the crack tip if the radius of the blunted tip were reduced to zero. Otherwise, the contour integral value will depend on the path until the contour region reaches the parallel crack faces.

Input File Usage: ∗CONTOUR INTEGRAL, CONTOURS=*n*
Specify the crack front node set name on the data line; the format depends on the method you use to specify the virtual crack extension direction.

For two-dimensional cases only one crack front node set (the crack front at the crack tip) must be specified. For three-dimensional cases you must repeat the data line to specify the crack front for each node (or cluster of focused nodes) along the crack line in order from one end of the crack to the other, including the midside nodes of second-order elements; it is not permissible to skip nodes along the crack line.

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front

Defining the crack tip or crack line

By default, Abaqus/Standard defines the crack tip as the first node specified for the crack front and the crack line as the sequence of first nodes specified for the crack front. The first node is the node with the smallest node number, unless the node set is generated as unsorted. Alternatively, you can specify the crack-tip node or crack-line nodes directly. This specification plays a critical role for a three-dimensional crack with a blunt crack tip.

Abaqus/CAE cannot determine the crack tip or crack line automatically based on the specified crack front. However, if you select a point to define the crack front in two dimensions, the same point defines the crack tip; likewise, if you select edges to define the crack front in three dimensions, the same edges define the crack line. For all other cases you must define the crack tip or crack line directly.

Input File Usage: Use the following option to specify the crack-tip nodes directly:

∗CONTOUR INTEGRAL, CONTOURS=*n*, CRACK TIP NODES
Specify the crack front node set name and the crack tip node number or node set name on the data line; the format depends on the method you use to specify the virtual crack extension direction.

Repeat the data line for three-dimensional cases.

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front, then select the crack tip (in two dimensions) or crack line (in three dimensions)

Defining a closed-loop crack line

Sometimes a crack line may form a closed loop (for example, when modeling a full penny-shaped crack without invoking symmetry conditions). In such cases the finite element mesh in the crack-tip region can be created with or without seams; i.e., linear constraint equations ("Linear constraint equations," Section 35.2.1) or multi-point constraints ("General multi-point constraints," Section 35.2.2) may or may not be used to tie two layers of nodes together.

If a crack line forms a closed loop, the starting node set of the crack front can be chosen arbitrarily and the other node sets defining the crack front must go around the crack front sequentially. The last node set defining the crack front must be the same as the first node set. If a closed loop is formed by creating coincident nodes that are then tied together by linear constraint equations and multi-point constraints, the node sets must be specified in order starting from one of the node sets involved in the constraint equation or multi-point constraint and terminating with the other node set.

Specifying the virtual crack extension direction

You must specify the direction of virtual crack extension at each crack tip in two dimensions or at each node along the crack line in three dimensions by specifying either the normal to the crack plane, $\mathbf{n}$, or the virtual crack extension direction, $\mathbf{q}$.

If the virtual crack extension direction is specified to point into the material (parallel to the crack faces), the *J*-integral values calculated will be positive. Negative *J*-integral values are obtained when the virtual crack extension direction is specified in the opposite direction.

Specifying the normal to the crack plane

The virtual crack extension direction can be defined by specifying the normal, $\mathbf{n}$, to the crack plane. In this case Abaqus/Standard will calculate a virtual crack extension direction, $\mathbf{q}$, that is orthogonal to the crack front tangent, $\mathbf{t}$, and the normal, $\mathbf{n}$. As shown in Figure 11.4.2–1, $\mathbf{q} = \mathbf{t} \times \mathbf{n}$ for a three-dimensional crack; for a two-dimensional crack, we simply have $q_x = -n_y$ and $q_y = n_x$. Specifying the normal implies that the crack plane is flat since only one value of $\mathbf{n}$ can be given per contour integral.

Input File Usage: ∗CONTOUR INTEGRAL, CONTOURS=*n*, NORMAL
n_x-direction cosine (or n_r), n_y-direction cosine (or n_z), n_z-direction cosine (or blank)
crack front node set name (2D) or names (3D)

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front:
Specify crack extension direction using: Normal to crack plane

Specifying the virtual crack extension direction

Alternatively, the virtual crack extension direction, $\mathbf{q}$, can be specified directly. In three dimensions the virtual crack extension direction, $\mathbf{q}$, will be corrected to be orthogonal to any normal defined at a node or in other cases to the tangent to the crack line itself. The tangent, $\mathbf{t}$, to the crack line at a particular point is obtained by parabolic interpolation through the crack front for which the virtual crack extension

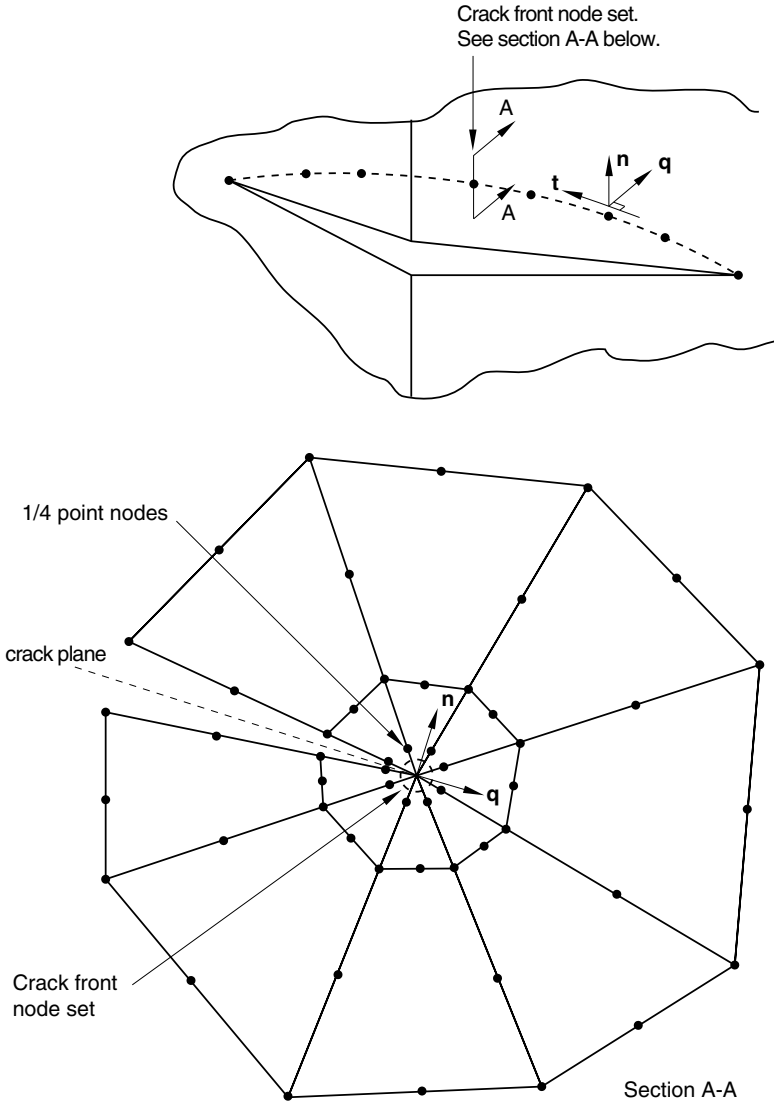


Figure 11.4.2–1 Typical focused mesh for fracture mechanics evaluation.

vector is defined and the nearest node sets on either side of this region. Abaqus/Standard will normalize the virtual crack extension direction, $\mathbf{q}$.

Input File Usage: ∗CONTOUR INTEGRAL, CONTOURS=*n*
crack front node set name, q_x-direction cosine (or q_r), q_y-direction cosine (or q_z), q_z-direction cosine (or blank)

Repeat the data line for three-dimensional cases to specify the crack front and virtual crack extension vector for each node (or cluster of focused nodes) along the crack line.

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front: **Specify crack extension direction using: q vectors**

Defining surface normals

In a case where the crack front intersects the external surface of a three-dimensional solid, where there is a surface of material discontinuity in the model, or where the crack is in a curved shell, the virtual crack extension direction, $\mathbf{q}$, must lie in the plane of the surface for accurate contour integral evaluation. Surface normals should be specified at all nodes that lie on such surfaces within the contours requested for this purpose (these nodes are printed out under the "Contour Integral" information in the data file). For shell element models the normals can be specified with the nodal coordinates if the normals calculated by Abaqus/Standard are not adequate. For solid element models the normals can be specified either directly (see "Normal definitions at nodes," Section 2.1.4, and "A plate with a part-through crack: elastic line spring modeling," Section 1.4.1 of the Abaqus Example Problems Guide) or using the nodal coordinates (the fourth–sixth coordinates). If surface normals are not specified for the nodes on the crack surfaces and the external surfaces at the ends of a crack line, Abaqus/Standard will calculate the normals automatically for these nodes to correct any inadequate virtual crack extension directions, $\mathbf{q}$.

Defining the data required for a contour integral with XFEM

If you are using XFEM to evaluate the contour integral, both the crack front and the virtual crack extension direction are determined by Abaqus/Standard.

Symmetry with the conventional finite element method

If the crack is defined on a symmetry plane, only half the structure needs to be modeled. The change in potential energy calculated from the virtual crack front advance is doubled to compute the correct contour integral values.

Input File Usage: Use the following option to indicate that the crack is defined on a symmetry plane:

∗CONTOUR INTEGRAL, CONTOURS=*n*, SYMM

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack tip or crack line, and specify the crack extension direction: **General**: toggle on **On symmetry plane (half-crack model)**

Constructing a fracture mechanics mesh for small-strain analysis with the conventional finite element method

Sharp cracks (where the crack faces lie on top of one another in the undeformed configuration) are usually modeled using small-strain assumptions. Focused meshes, as shown in Figure 11.4.2–1, should

normally be used for small-strain fracture mechanics evaluations. However, for a sharp crack the strain field becomes singular at the crack tip. This result is obviously an approximation to the physics; however, the large-strain zone is very localized, and most fracture mechanics problems can be solved satisfactorily using only small-strain analysis.

The crack-tip strain singularity depends on the material model used. Linear elasticity, perfect plasticity, and power-law hardening are commonly used in fracture mechanics analysis. Power-law hardening has the form

$$\frac{\bar{\varepsilon}}{\varepsilon_0} = \alpha \left(\frac{\bar{\sigma}}{\sigma_0} \right)^n ,$$

where $\bar{\varepsilon}$ is the equivalent total strain, ε_0 is a reference strain, $\bar{\sigma}$ is the Mises stress, σ_0 is the initial yield stress, n is the power-law hardening exponent (typically in the range of 3 to 8; $n > 10$ is very close to perfect plasticity for large ε), and α is a material constant (typically in the range 0.5 to 1.0).

Results for pure power-law nonlinear elastic materials in a body under traction loading are proportional to the load to some power. Therefore, the fracture parameters for one geometry under a particular load can be scaled to any other load of the same distribution but different magnitude.

If the loading is proportional (the direction of the stress increase in stress space is approximately constant) and monotonically increasing, power-law hardening deformation plasticity and incremental plasticity are essentially equivalent. However, deformation plasticity is a nonlinear elastic material for which more analytical results are available. Abaqus uses the Ramberg-Osgood form of deformation plasticity (see "Deformation plasticity," Section 23.2.13); this model is not a pure power law model, which must be considered.

Creating the singularity

In most cases the singularity at the crack tip should be considered in small-strain analysis (when geometric nonlinearities are ignored). Including the singularity often improves the accuracy of the J-integral, the stress intensity factors, and the stress and strain calculations because the stresses and strains in the region close to the crack tip are more accurate. If r is the distance from the crack tip, the strain singularity in small-strain analysis is

$\varepsilon \propto r^{-1/2}$	for linear elasticity,
$\varepsilon \propto r^{-1}$	for perfect plasticity, and
$\varepsilon \propto r^{-\frac{n}{n+1}}$	for power-law hardening.

Modeling the crack-tip singularity in two dimensions

The square root and $1/r$ singularity can be built into a finite element mesh using standard elements. The crack tip is modeled with a ring of collapsed quadrilateral elements, as shown in Figure 11.4.2–2.

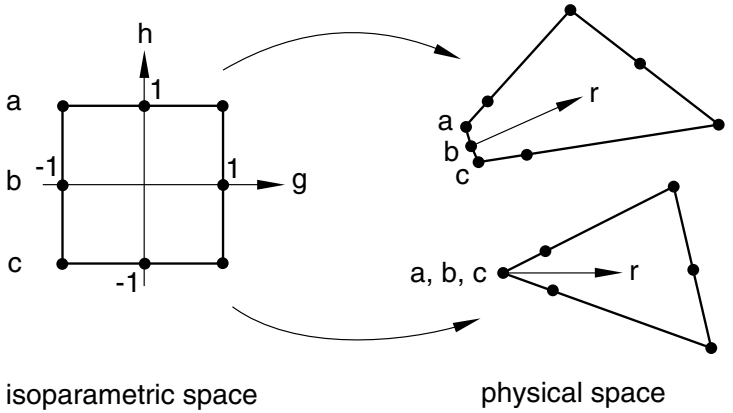


Figure 11.4.2–2 Collapsed two-dimensional element.

To obtain a mesh singularity, generally second-order elements are used and the elements are collapsed as follows:

1. Collapse one side of an 8-node isoparametric element (CPE8R, for example) so that all three nodes—*a*, *b*, and *c*—have the same geometric location (on the crack tip).
2. Move the midside nodes on the sides connected to the crack tip to the 1/4 point nearest the crack tip. You can create "quarter point" spacing with second-order isoparametric elements when you generate nodes for a region of a mesh; see "Creating quarter-point spacing" in "Node definition," Section 2.1.1.

This procedure will create the strain singularity

$$\varepsilon \to \frac{A}{r} + \frac{B}{r^{1/2}} \quad \text{as} \quad r \to 0.$$

The $r^{-\frac{n}{n+1}}$ singularity cannot be created using Abaqus elements, but the combination of the r^{-1} and $r^{-1/2}$ terms can provide a reasonable approximation for $r^{-\frac{n}{n+1}}$.

If 4-node isoparametric elements (for example, CPE4R) are used, one side of the element is collapsed, and the two coincident nodes are free to displace independently, a $1/r$ singularity is created.

If the crack region is meshed with linear elements, the position specified for the midside nodes is ignored.

Creating a square root singularity

If nodes *a*, *b*, and *c* are constrained to move together, $A = 0$ and the strains and stresses are square root singular (suitable for linear elasticity).

Input File Usage: *NFILL, SINGULAR

Constrain the collapsed nodes to move together by specifying the same node number in the list of nodes forming the element or by using a linear constraint equation or multi-point constraint to tie them together.

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, single node**

Creating a *1/r* singularity

If the midside nodes remain at the midside points rather than being moved to the 1/4 points and nodes a, b, and c are allowed to move independently, only the $1/r$ singularity in strain is created (suitable for perfect plasticity).

Input File Usage: *NFILL

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.5, **Collapsed element side, duplicate nodes**

Creating a combined square root and *1/r* singularity

If the midside nodes are moved to the 1/4 points but nodes a, b, and c are allowed to move independently, the singularity created is a combination of the square root and $1/r$ singularities. This combination is usually best for a power-law hardening material. However, since the $1/r$ singularity dominates, moving the midside nodes to the 1/4 points gives only slightly better results than if the nodes are left at the midside points. Since creating a mesh with the midside nodes moved to the quarter points can be difficult, it is often best to simply use the $1/r$ singularity.

Input File Usage: *NFILL, SINGULAR

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack tip, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, duplicate nodes**

Modeling the crack-tip singularity in three dimensions

To create singular fields, 20-node bricks and 27-node bricks can be used with a collapsed face (see Figure 11.4.2–3).The planes of the three-dimensional elements perpendicular to the crack line should be planar for the best accuracy. If they are not planar, the element Jacobian may become negative at some integration points when the midside nodes are moved to the 1/4 points. To correct this problem, move the midside nodes slightly away from the 1/4 points toward the midpoint position (the distance moved is not critical).

See "Meshing the crack region and assigning elements," Section 31.2.7 of the Abaqus/CAE User's Guide, for information on creating a three-dimensional fracture mechanics mesh in Abaqus/CAE.

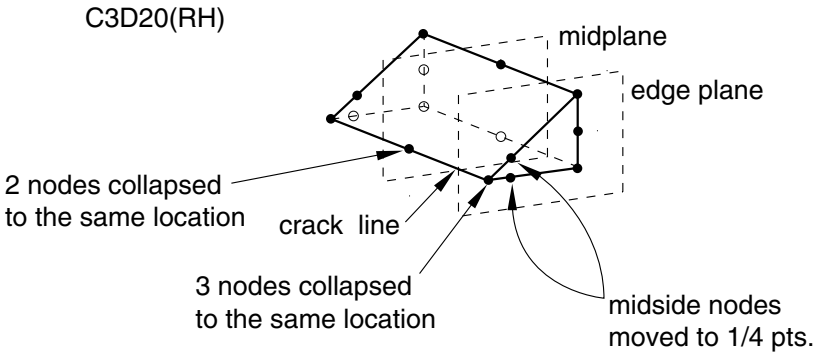


Figure 11.4.2–3 Collapsed three-dimensional element.

Creating a square root singularity

To obtain a square root singularity, constrain the nodes on the collapsed face of the edge planes to move together and move the nodes to the 1/4 points.

If the nodes at the midplane of a collapsed 20-node brick are constrained to move together, $\varepsilon \propto r$; therefore, the singularity is not the same on the midplane as on an edge plane. This difference causes local oscillations in the solution about the crack tip along the crack line, although normally the oscillations are not significant.

If all midface nodes and the centroid node are included in a 27-node brick and the midside and midface nodes are moved to the 1/4 points closest to the crack line, the oscillation in the local stress and strain fields can be reduced.

Input File Usage: *NFILL, SINGULAR

Constrain the collapsed nodes to move together by specifying the same node number in the list of nodes forming the element or by using a linear constraint equation or multi-point constraint to tie them together.

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, single node**

Creating a *1/r* singularity

To obtain a $1/r$ singularity, allow the three nodes on the collapsed face to displace independently and keep the midside nodes at the midpoints.

Input File Usage: *NFILL

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.5, **Collapsed element side, duplicate nodes**

Creating a combined square root and $1/r$ singularity

To obtain a combined square root and $1/r$ singularity, allow the nodes on the collapsed face to displace independently and move the midside nodes to the 1/4 points. As in the two-dimensional case, if it is difficult to create the mesh with the nodes moved to the 1/4 points, simply use the $1/r$ singularity.

Input File Usage: *NFILL, SINGULAR

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: select the crack front and crack line, and specify the crack extension direction: **Singularity**: **Midside node parameter:** 0.25, **Collapsed element side, duplicate nodes**

Mesh refinement

The size of the crack-tip elements influences the accuracy of the solutions: the smaller the radial dimension of the elements from the crack tip, the better the stress, strain, etc. results will be and, therefore, the better the contour integral calculations will be.

The angular strain dependence is not modeled with the singular elements. Reasonable results are obtained if typical elements around the crack tip subtend angles in the range of 10° (accurate) to 22.5° (moderately accurate).

Since the crack tip causes a stress concentration, the stress and strain gradients are large as the crack tip is approached. Path dependence in the evaluation of the J-integral may be an indication that the mesh is not sufficiently refined, but path independence does not prove mesh convergence. The finite element mesh must be refined in the vicinity of the crack to get accurate stresses and strains; however, accurate J-integral results can frequently be obtained even with a relatively coarse mesh.

In many cases if sufficiently fine meshes are used, accurate contour integral values can be obtained without using singular elements.

Modeling the crack-tip region in shells

Focused meshes can be used, but not all of the three-dimensional shell elements in Abaqus/Standard can be collapsed. Elements S8R and S8RT cannot be degenerated into triangles; element types S4, S4R, S4R5, S8R5, and S9R5 can.

The quarter-point technique (moving the midside nodes to the quarter points to give a $1/\sqrt{r}$ singularity for elastic fracture mechanics applications) can be used with S8R5 and S9R5 elements but not with S8R(T) elements. When the quarter-point technique is used with S9R5 elements, the midface node should be moved to the quarter-point position along with the two midside nodes.

If S8R(T) elements are used, a keyhole should be introduced at the crack tip.

Flaws lying in the plane through the thickness of a shell can be modeled using line spring elements; see "Line spring elements for modeling part-through cracks in shells," Section 32.9.1. In many cases line spring elements provide accurate J-integral and stress intensity values, but these elements are limited to modeling small strain and rotations. Limited modeling of plasticity is also allowed with line springs.

Constructing a fracture mechanics mesh for finite-strain analysis with the conventional finite element method

In large-strain analysis (when geometric nonlinearities are included) singular elements should not normally be used. The mesh must be sufficiently refined to model the very high strain gradients around the crack tip if details in this region are required. Even if only the ***J***-integral is required, the deformation around the crack tip may dominate the solution and the crack-tip region will have to be modeled with sufficient detail to avoid numerical problems.

Physically, the crack tip is not perfectly sharp. Therefore, it is normally modeled as a blunted notch with a radius of $\sim 10^{-3} r_p$, where r_p is a characteristic dimension of the plastic zone ahead of the crack tip. The notch must be small enough that, at the loads of interest, the deformed shape of the notch no longer depends on the original geometry. Typically, the notch must blunt out to more than four times its original radius for the deformed shape to be independent of the original geometry. The size of the elements around the notch should be about 1/10 the notch-tip radius to obtain accurate results.

If a crack is modeled as sharp, the finite elements near the crack tip may not be able to approximate the high gradients, resulting in convergence problems. The stress and strain results around the crack tip will probably be inaccurate even if convergence is achieved. However, if the solution converges, the contour integral results should be reasonably accurate. The convergence difficulties will probably be greater in three dimensions than in two dimensions.

In situations involving finite rotations but small strains, such as bending of slender structures, a small "keyhole" around the crack tip should be modeled. If the hole is small, the results will not be affected significantly and problems in dealing with the singular strains at the crack tip will be avoided.

Using constraints with the conventional finite element method

General multi-point constraints and linear constraint equations ("Kinematic constraints: overview," Section 35.1.1) should not be used on nodes in the mesh regions where contour integrals are calculated unless the nodes involved in the constraint are located at the same point. The nodes at the crack tip of a focused mesh can be tied together using multi-point constraints without adversely affecting the contour integral calculations. Tying these nodes will change the singularity at the crack tip, but path independence of the contour integral will be maintained. In addition, path independence of the contour integrals will not be affected if two faces of a model are joined using MPC type TIE or a linear constraint equation, provided that all nodes of the two faces are coincident. Using multi-point constraints for mesh refinement or for applying symmetry/antisymmetry boundary conditions within the contour integral region will result in path dependence of the contour integrals. No warning or error messages are provided if this rule is violated.

Procedures

You can request contour integrals in fracture mechanics problems that were modeled using the following procedures:

- static ("Static stress analysis," Section 6.2.2) with both XFEM and the conventional finite element methods;

- quasi-static ("Quasi-static analysis," Section 6.2.5) with the conventional finite element method only;
- steady-state transport ("Steady-state transport analysis," Section 6.4.1) with the conventional finite element method only;
- coupled thermal-stress procedures ("Fully coupled thermal-stress analysis," Section 6.5.3) with the conventional finite element method only; and
- crack propagation ("Crack propagation analysis," Section 11.4.3) with the conventional finite element method only.

Contour integrals can be requested only in general analysis steps: they are not calculated in linear perturbation analyses ("General and linear perturbation procedures," Section 6.1.3).

A crack analysis with pressure applied on the crack surfaces may give inaccurate contour integral values if geometric nonlinearity is included in a step.

Loads

Contour integral calculations include the following distributed load types:

- thermal loads;
- distributed loads, including crack face pressure and traction loads on continuum elements as well as those applied using user subroutine **DLOAD** and **UTRACLOAD**;
- distributed loads, including surface traction loads and crack face edge loads on shell elements as well as those applied using user subroutine **UTRACLOAD**;
- uniform and nonuniform body forces; and
- centrifugal loads on continuum and shell elements.

Contributions to the contour integral due to concentrated loads in the domain are not included; instead, the mesh must be modified to include a small element and a distributed load must be applied to this element.

Contributions due to contact forces are not included.

Material options

J-integral calculations are valid for linear elastic, nonlinear elastic, and elastic-plastic materials. Plastic behavior can be modeled as nonlinear elastic ("Deformation plasticity," Section 23.2.13), but the results are generally best if the material is modeled by incremental plasticity and is subject to proportional, monotonic traction loading.

If unloading has taken place in the plastic zone around the crack tip, the *J*-integral will not be valid except in very limited cases.

The C_t-integral is valid for problems involving creep ("Rate-dependent plasticity: creep and swelling," Section 23.2.4).

The stress intensity factor calculation is valid for cracks in homogeneous, linear elastic materials. It is also valid for an interfacial crack between two different isotropic linear elastic materials. It is not valid for any other types of materials, including user-defined materials.

The crack propagation direction is valid only for homogeneous, isotropic linear elastic materials.

The T-stress is valid only for homogeneous, isotropic linear elastic materials. Although the T-stress is calculated using the linear elastic material properties of the body with a crack, it is usually used with the J-integral calculated using the elastic-plastic material properties of the body (see "T-stress extraction," Section 2.16.3 of the Abaqus Theory Guide).

If there is material discontinuity, the normal to the material discontinuity line must be specified for all nodes on the material discontinuity that will lie in a contour integral domain. The normal can be specified by defining user-specified normals (see "Normal definitions at nodes," Section 2.1.4) for the elements on both sides of the discontinuity or by using nodal normal coordinates for the nodes on the discontinuity. Contour integral calculations cannot be performed for a crack with a material discontinuity line passing through its tip (except for an interfacial crack between two different materials). Therefore, you should be careful when specifying a normal that is not perpendicular to the virtual crack extension direction, $\mathbf{q}$, for the nodes at the crack tip.

Elements

When used with XFEM, the contour integral can be evaluated only in first-order or second-order tetrahedron and first-order brick elements. The following paragraphs apply only to the conventional finite element method.

The contour integral evaluation capability in Abaqus/Standard assumes that the elements that lie within the domain used for the calculations are quadrilaterals in two-dimensional or shell models or bricks in continuum three-dimensional models. Triangles, tetrahedra, or wedges should not be used in the mesh that is included in the contour integral regions. When the elements around the crack tip are generated in Abaqus/CAE, triangular elements (in two dimensions) or wedge elements (in three dimensions) are converted to collapsed quadrilateral or hexahedral elements. The elements within the contour domain should be of the same type.

In shell structures the contour integrals calculated by Abaqus/Standard will be contour independent only if the deformation mode around the crack tip is primarily membrane. If there are significant bending or transverse shear effects in the domain, the contour integrals may not be contour independent and contour integral values should be obtained directly from the displacements and/or the stresses.

Generalized plane strain elements, generalized axisymmetric elements with twist, asymmetric-axisymmetric elements, membrane elements, and cylindrical elements should not be used in the contour integral regions.

The contribution of rebar is included only in the calculations of the J-integral and the C_t-integral for shell elements defined with a shell section integrated during the analysis (see "Using a shell section integrated during the analysis to define the section behavior," Section 29.6.5).

Output

The domain associated with each contour is calculated automatically. The nodes belonging to each domain can be printed in the data file; see "Controlling the amount of analysis input file processor information written to the data file" in "Output," Section 4.1.1. If you are using the conventional contour integral method, for each domain Abaqus/Standard creates a new node set in the output database to include these nodes; you can view these node sets in Abaqus/CAE. In addition, new node sets are

created in the output database for nodes on crack surfaces and on free surfaces whose nodal normals are calculated by Abaqus/Standard.

Contour integrals cannot be recovered from the restart file as described in "Output," Section 4.1.1.

You should not request element output extrapolated to the nodes ("Element output" in "Output to the data and results files," Section 4.1.2) for second-order elements with one collapsed side in two dimensions or one collapsed face in three dimensions.

Default contour integral output

By default, the contour integral values are written to the data file and to the output database file. The following naming convention is used for contour integrals written to the output database:

integral-type`:` *abbrev-integral-type* `at` *history-output-request-name_crack-name_internal-crack-tip-node-set-name*`__Contour_`*contour-number*

where *integral-type* can be

- `Crack propagation direction (Cpd)`
- `J-integral (J)`
- `J-integral estimated from Ks (JKs)`
- `Stress intensity factor K1 (K1)`
- `Stress intensity factor K2 (K2)`
- `T-stress (T)`

For example,

```
J-integral: J at JINT_CRACK_CRACKTIP-1__Contour_1
```

Writing the contour integrals to the results file

You can choose to write the contour integral values to the results file in addition to the data file.

Input File Usage: Use the following option to write the contour integrals to the results file instead of the data file:

∗CONTOUR INTEGRAL, CONTOURS=*n*, OUTPUT=FILE

Use the following option to write the contour integrals to the results file in addition to the data file:

∗CONTOUR INTEGRAL, CONTOURS=*n*, OUTPUT=BOTH

Abaqus/CAE Usage: You cannot write contour integrals to the results file from Abaqus/CAE.

Controlling the output frequency

You can control the output frequency, in increments, of contour integrals. By default, the crack-tip location and associated quantities will be printed every increment. Specify an output frequency of 0 to suppress contour integral output.

The output frequency for contour integral output to the output database is controlled by the larger of the frequency values specified for history output to the output database (see "Output to the output

database," Section 4.1.3) or for contour integral output. If you specify an output frequency of 0 for the history output to the output database, contour integral values will not be written to the output database.

Input File Usage: *CONTOUR INTEGRAL, CRACK NAME=*crack name*, CONTOURS=*n*, FREQUENCY=*f*

Abaqus/CAE Usage: Step module: history output request editor: **Domain: Crack**: *crack name*, **Number of contours:** *n*, **Save output at**

11.4.3 CRACK PROPAGATION ANALYSIS

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Fracture mechanics: overview," Section 11.4.1
- "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7
- "Surface-based cohesive behavior," Section 37.1.10
- *COHESIVE BEHAVIOR
- *CONTACT CLEARANCE
- *DEBOND
- *DIRECT CYCLIC
- *FRACTURE CRITERION
- *NODAL ENERGY RATE
- "Defining surface-to-surface contact in an Abaqus/Standard analysis" in "Defining surface-to-surface contact," Section 15.13.7 of the Abaqus/CAE User's Guide

Overview

Crack propagation analysis:

- allows for six types of fracture criteria in Abaqus/Standard—critical stress at a certain distance ahead of the crack tip, critical crack opening displacement, crack length versus time, VCCT (the Virtual Crack Closure Technique), enhanced VCCT, and the low-cycle fatigue criterion based on the Paris law;
- allows for the VCCT fracture criterion in Abaqus/Explicit;
- in Abaqus/Standard models quasi-static crack growth in two dimensions (planar and axisymmetric) for all types of fracture criteria and in three dimensions (solid, shells, and continuum shells) for VCCT, enhanced VCCT, and the low-cycle fatigue criteria; and
- in Abaqus/Explicit models crack growth in three dimensions (solid, shells, and continuum shells) for VCCT criterion; and
- requires that you define two distinct initially bonded contact surfaces between which the crack will propagate.

Defining initially bonded crack surfaces in Abaqus/Standard

Potential crack surfaces are modeled as slave and master contact surfaces (see "Defining contact pairs in Abaqus/Standard," Section 36.3.1). Any contact formulation except the finite-sliding, surface-to-surface formulation can be used. The predetermined crack surfaces are assumed to be initially partially bonded

so that the crack tips can be identified explicitly by Abaqus/Standard. Initially bonded crack surfaces cannot be used with self-contact.

Define an initial condition to identify which part of the crack is initially bonded. You specify the slave surface, the master surface, and a node set that identifies the initially bonded part of the slave surface. The unbonded portion of the slave surface will behave as a regular contact surface. Either the slave surface or the master surface must be specified; if only the master surface is given, all of the slave surfaces associated with this master surface that have nodes in the node set will be bonded at these nodes.

If a node set is not specified, the initial contact conditions will apply to the entire contact pair; in this case, no crack tips can be identified, and the bonded surfaces cannot separate.

If a node set is specified, the initial conditions apply only to the slave nodes in the node set. Abaqus/Standard checks to ensure that the node set defined includes only slave nodes belonging to the contact pair specified.

By default, the nodes in the node set are considered to be initially bonded in all directions.

Input File Usage: *INITIAL CONDITIONS, TYPE=CONTACT

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)**

Bonding only in the normal direction

For fracture criteria based on the critical stress, critical crack opening displacement, or crack length versus time, it is possible to bond the nodes in the node set (or the contact pair if a node set is not defined) only in the normal direction. In this case the nodes are allowed to move freely tangential to the contact surfaces. Friction ("Frictional behavior," Section 37.1.5) cannot be specified if the nodes are bonded only in the normal direction.

Bonding only in the normal direction is typically used to model bonded contact conditions in Mode I crack problems where the shear stress ahead of the crack along the crack plane is zero.

Input File Usage: *INITIAL CONDITIONS, TYPE=CONTACT, NORMAL

Abaqus/CAE Usage: Bonding only in the normal direction is not supported in Abaqus/CAE.

Activating the crack propagation capability in Abaqus/Standard

The crack propagation capability must be activated within the step definition to specify that crack propagation may occur between the two surfaces that are initially partially bonded. You specify the surfaces along which the crack propagates.

If the crack propagation capability is not activated for partially bonded surfaces, the surfaces will not separate; in this case the specified initial contact conditions would have the same effect as that provided by the tied contact capability, which generates a permanent bond between two surfaces during the entire analysis (see "Defining tied contact in Abaqus/Standard," Section 36.3.7).

Input File Usage: *DEBOND, SLAVE=*slave_surface_name*, MASTER=*master_surface_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Surface-to-surface contact (Standard)**, select master and slave surfaces

Propagation of multiple cracks

Cracks can propagate from either a single crack tip or multiple crack tips. The crack propagation capability in Abaqus/Standard requires that the surfaces be initially partially bonded so that the crack tips can be identified. A contact pair can have crack propagation from multiple crack tips. However, only one crack propagation criterion is allowed for a given contact pair. Crack propagation along several contact pairs can be modeled by specifying multiple crack propagation definitions.

Defining and activating crack propagation in Abaqus/Explicit

In Abaqus/Explicit potential crack surfaces are modeled as bonded general contact surfaces (see "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1) in the context of surface-based cohesive behavior (see "Surface-based cohesive behavior," Section 37.1.10). Hence, the capability is available in three-dimensional analyses only and is implemented using a pure master-slave formulation. As is the case in Abaqus/Standard, the predetermined crack surfaces are assumed to be initially partially bonded so that the crack tips can be identified explicitly.

To identify which pair of surfaces determine the crack and which part of the crack is initially bonded, you must define and assign a contact clearance (see "Controlling initial contact status for general contact in Abaqus/Explicit," Section 36.4.4). You first define a contact clearance to specify the node set that is initially bonded, and then you assign this contact clearance to a pair of two single-sided surfaces that define the crack. The unbonded portion behaves as a regular contact surface. The nodes in the node set are considered to be initially bonded in all directions.

The crack tip is identified only from the specified two surfaces and the node set. No attempt is made to determine a crack tip from all surfaces included in the general contact domain. Consequently, to be able to identify the crack tip, the surface including the specified node set must extend past the node set. Otherwise, the surfaces will not debond, and the crack cannot propagate.

You complete the definition of the crack propagation capability by defining a fracture-based cohesive behavior surface interaction. You activate the crack propagation by assigning it to the pair of surfaces that are initially partially bonded. If the fracture criterion is met, crack propagation occurs between these two surfaces. Cohesive behavior is also used to specify the elastic behavior of the bonds (see "Surface-based cohesive behavior," Section 37.1.10).

If a fracture-based surface interaction is not assigned to a pair of surfaces, the crack definition is incomplete. Unlike Abaqus/Standard where the identified nodes will stay bonded if the crack is not activated, in Abaqus/Explicit the nodes identified by the contact clearance definition will separate without generating any interface stress.

Similar to Abaqus/Standard, cracks can propagate from single or multiple crack tips for the same pair of surfaces.

Input File Usage: Use the following options:

```
*CONTACT CLEARANCE, NAME=clearance_name,
SEARCH NSET=bonded_nset_name
**
*SURFACE INTERACTION, NAME=interaction_name
```

∗COHESIVE BEHAVIOR
∗FRACTURE CRITERION
..**
∗CONTACT
∗CONTACT CLEARANCE ASSIGNMENT
slave_surface, *master_surface*, *clearance_name*
∗CONTACT PROPERTY ASSIGNMENT
slave_surface, *master_surface*, *interaction_name*

Abaqus/CAE Usage: Defining and activating crack propagation in Abaqus/Explicitis not supported in Abaqus/CAE.

Specifying a fracture criterion

You can specify the crack propagation criteria, as discussed below. Table 11.4.3–1 shows which criteria are supported by Abaqus/Standard and Abaqus/Explicit. Only one crack propagation criterion is allowed per contact pair even if multiple cracks are present.

Table 11.4.3–1

Crack propagation criterion	Abaqus/Standard	Abaqus/Explicit
Critical stress	Yes	No
Critical crack opening displacement	Yes	No
Crack length versus time	Yes	No
VCCT	Yes	Yes
Enhanced VCCT	Yes	No
Low-cycle fatigue	Yes	No

Crack propagation analysis is carried out on a nodal basis. The crack-tip node debonds when the fracture criterion, f, reaches the value 1.0 within a given tolerance:

$$f_{LL} \leq f \leq f_{UL},$$

where $f_{UL} = 1 + f_{tol}$ and $f_{LL} = 1$ for VCCT, enhanced VCCT, and low-cycle fatigue criteria or $f_{LL} = 1 - f_{tol}$ for other fracture criteria. You can specify the tolerance f_{tol}. In Abaqus/Standard, if $f > 1 + f_{tol}$, the time increment is cut back such that the crack propagation criterion is satisfied except in the case of an unstable crack growth problem where multiple nodes at and ahead of a crack tip are allowed to debond without the cut back of increment size in one increment. The default value of f_{tol} is 0.1 for the critical stress, critical crack opening displacement, and crack length versus time criteria and is 0.2 for the VCCT, enhanced VCCT, and low-cycle fatigue criteria.

Input File Usage: *FRACTURE CRITERION, TOLERANCE=f_{tol}, TYPE=*type*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT** or **Enhanced VCCT**, **Tolerance**

Critical stress criterion

This criterion is available only in Abaqus/Standard.

If you specify a critical stress criterion at a critical distance ahead of the crack tip, the crack-tip node debonds when the local stress across the interface at a specified distance ahead of the crack tip reaches a critical value.

This criterion is typically used for crack propagation in brittle materials. The critical stress criterion is defined as

$$f = \sqrt{\left(\frac{\hat{\sigma}_n}{\sigma^f}\right)^2 + \left(\frac{\tau_1}{\tau_1^f}\right)^2 + \left(\frac{\tau_2}{\tau_2^f}\right)^2}, \quad \hat{\sigma}_n = \max(\sigma_n, 0),$$

where σ_n is the normal component of stress carried across the interface at the distance specified; τ_1 and τ_2 are the shear stress components in the interface; and σ^f and τ_1^f are the normal and shear failure stresses, which you must specify. The second component of the shear failure stress, τ_2, is not relevant in a two-dimensional analysis; therefore, the value of τ_2^f need not be specified. The crack-tip node debonds when the fracture criterion, f, reaches the value 1.0.

If the value of τ_1^f is not given or is specified as zero, it will be taken to be a very large number so that the shear stress has no effect on the fracture criterion.

The distance ahead of the crack tip is measured along the slave surface, as shown in Figure 11.4.3–1. The stresses at the specified distance ahead of the crack tip are obtained by interpolating the values at the adjacent nodes. The interpolation depends on whether first-order or second-order elements are used to define the slave surface.

Input File Usage: *FRACTURE CRITERION, TYPE=CRITICAL STRESS, DISTANCE=*n*

Abaqus/CAE Usage: The critical stress criterion is not supported in Abaqus/CAE.

Critical crack opening displacement criterion

This criterion is available only in Abaqus/Standard.

If you base the crack propagation analysis on the crack opening displacement criterion, the crack-tip node debonds when the crack opening displacement at a specified distance behind the crack tip reaches a critical value. This criterion is typically used for crack propagation in ductile materials.

The crack opening displacement criterion is defined as

$$f = \frac{\delta}{\delta_c},$$

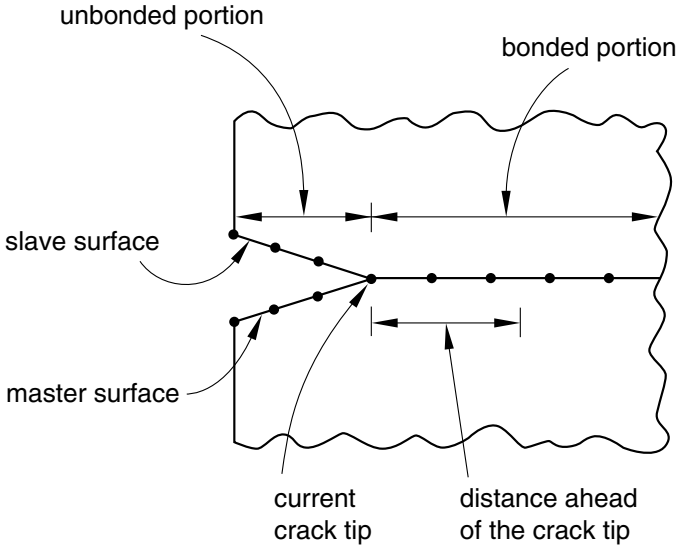


Figure 11.4.3–1 Distance specification for the critical stress criterion.

where δ is the measured value of crack opening displacement and δ_c is the critical value of the crack opening displacement (user-specified). The crack-tip node debonds when the fracture criterion reaches the value 1.0.

You must supply the crack opening displacement versus cumulative crack length data. In Abaqus/Standard the cumulative crack length is defined as the distance between the initial crack tip and the current crack tip measured along the slave surface in the current configuration. The crack opening displacement is defined as the normal distance separating the two faces of the crack at the given distance.

You specify the position, n, behind the crack tip where the critical crack opening displacement is calculated. The value of this position must be specified as the length of the straight line joining the current crack tip and points on the slave and master surfaces (Figure 11.4.3–2).

Abaqus/Standard computes the crack opening displacement at that point by interpolating the values at the adjacent nodes. The interpolation depends on whether first-order or second-order elements are used to define the slave surface. An error message will be issued if the value of n is not within the end points of the contact pair.

Input File Usage: *FRACTURE CRITERION, TYPE=COD, DISTANCE=*n*

Abaqus/CAE Usage: The critical crack opening displacement criterion is not supported in Abaqus/CAE.

Modeling symmetry

In problems where the debonding surfaces lie on a symmetry plane, you can specify that Abaqus/Standard should consider only half of the user-specified crack opening displacement values. In this case the initial bonding must be in the normal direction only (see “Bonding only in the normal direction” above).

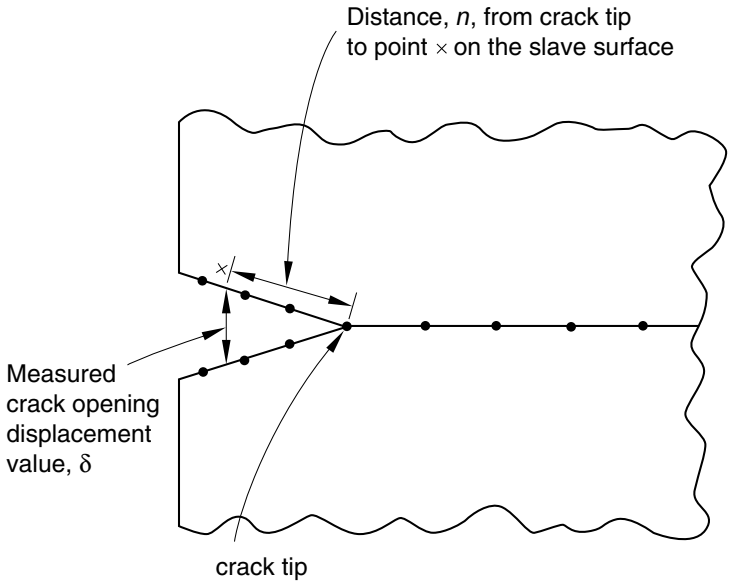


Figure 11.4.3–2 Distance specification for the critical crack opening displacement criterion.

Input File Usage: *FRACTURE CRITERION, TYPE=COD, DISTANCE=*n*, SYMMETRY

Abaqus/CAE Usage: Modeling symmetry is not supported in Abaqus/CAE.

Crack length versus time criterion

This criterion is available only in Abaqus/Standard.

To specify the crack propagation explicitly as a function of total time, you must provide a crack length versus time relationship and a reference point from which the crack length is measured. This reference point is defined by specifying a node set. Abaqus/Standard finds the average of the current positions of the nodes in the set to define the reference point. During crack propagation the crack length is measured from this user-specified reference point along the slave surface in the deformed configuration. The time specified must be total time, not step time.

The fracture criterion, f, is stated in terms of the user-specified crack length and the length of the current crack tip. The length of the current crack tip from the reference point is measured as the sum of the straight line distance of the initial crack tip from the reference point and the distance between the initial crack tip and the current crack tip measured along the slave surface.

Referring to Figure 11.4.3–3, let node 1 be the initial location of the crack tip and node 3 be the current location of the crack tip. The distance of the current crack tip located at node 3 is given by

$$l_3 = l_1 + \Delta l_{12} + \Delta l_{23},$$

where l_1 is the length of the straight line joining node 1 and the reference point, Δl_{12} is the distance between nodes 1 and 2, and Δl_{23} is the distance between nodes 2 and 3 measured along the slave surface.

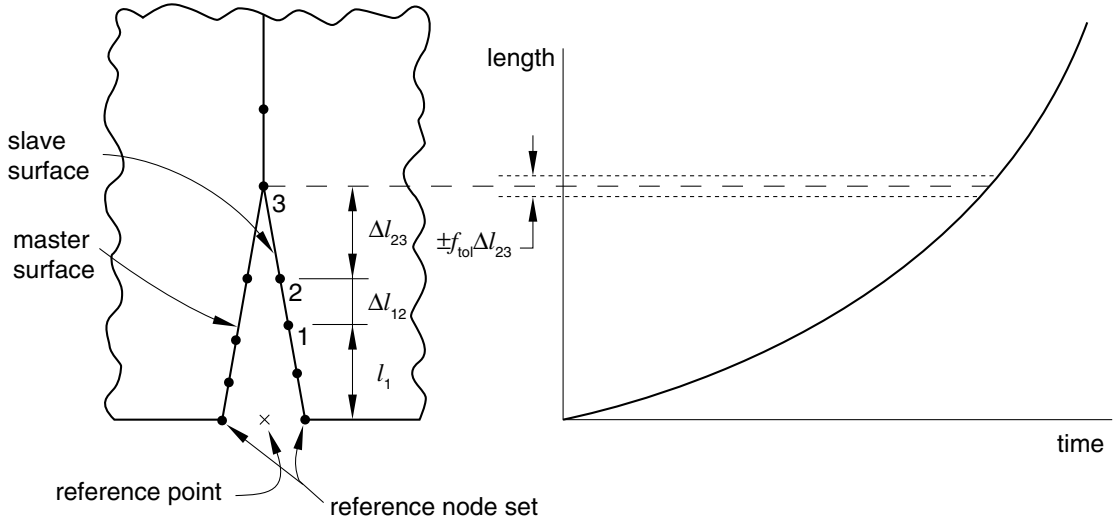


Figure 11.4.3–3 Crack propagation as a function of time.

The fracture criterion, f, is given by

$$f = \frac{l - (l_3 - \Delta l_{23})}{\Delta l_{23}},$$

where l is the length at the current time obtained from the user-specified crack length versus time curve. Crack-tip node 3 will debond when the failure function f reaches the value of 1.0 (within the user-defined tolerance).

If geometric nonlinearity is considered in the step ("Defining an analysis," Section 6.1.2), the reference point may move as the body deforms; you must ensure that this movement does not invalidate the crack length versus time criterion.

Abaqus/Standard does not extrapolate beyond the end points of your crack data. Therefore, if the first crack length specified is greater than the distance from the crack reference point to the first bonded node, the first bonded node will never debond and the crack will not propagate. In this case Abaqus/Standard will print warning messages in the message (**`.msg`**) file.

Input File Usage: *FRACTURE CRITERION, TYPE=CRACK LENGTH, NSET=*name*

Abaqus/CAE Usage: The crack length versus time criterion is not supported in Abaqus/CAE.

VCCT criterion

This criterion is available in both Abaqus/Standard and Abaqus/Explicit.

The Virtual Crack Closure Technique (VCCT) criterion uses the principles of linear elastic fracture mechanics (LEFM) and, therefore, is appropriate for problems in which brittle crack propagation occurs along predefined surfaces.

VCCT is based on the assumption that the strain energy released when a crack is extended by a certain amount is the same as the energy required to close the crack by the same amount. For example, Figure 11.4.3–4 illustrates the similarity between crack extension from *i* to *j* and crack closure at *j*.

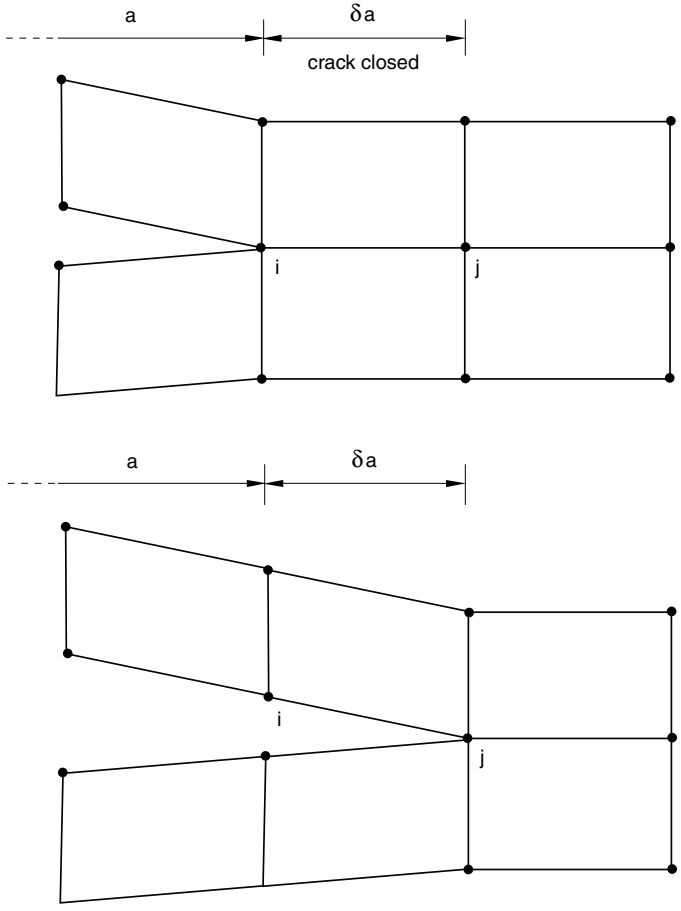


Figure 11.4.3–4 Mode I: The energy released when a crack is extended by a certain amount is the same as the energy required to close the crack.

In Figure 11.4.3–5 nodes 2 and 5 will start to release when

$$f = \frac{G_I}{G_{IC}} = \frac{1}{2}\left(\frac{v_{1,6}F_{v,2,5}}{bd}\right)\frac{1}{G_{IC}} \geq 1.0,$$

where G_I is the Mode I energy release rate, G_{IC} is the critical Mode I energy release rate, b is the width, d is the length of the elements at the crack front, $F_{v,2,5}$ is the vertical force between nodes 2 and 5, and $v_{1,6}$ is the vertical displacement between nodes 1 and 6. Assuming that the crack closure is governed by linear elastic behavior, the energy to close the crack (and, thus, the energy to open the crack) is calculated from the previous equation. Similar arguments and equations can be written in two dimensions for Mode II and for three-dimensional crack surfaces including Mode III.

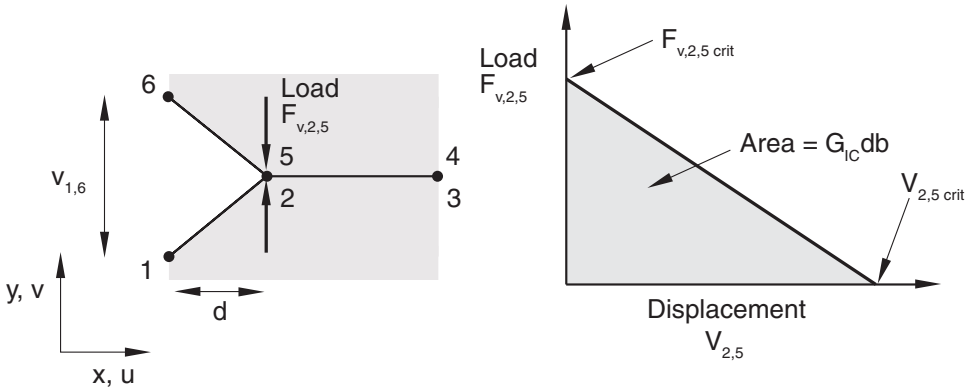


Figure 11.4.3–5 Pure Mode I modified.

In the general case involving Mode I, II, and III the fracture criterion is defined as

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0,$$

where G_{equiv} is the equivalent strain energy release rate calculated at a node, and G_{equivC} is the critical equivalent strain energy release rate calculated based on the user-specified mode-mix criterion and the bond strength of the interface. The crack-tip node will debond when the fracture criterion reaches the value of 1.0.

Abaqus provides three common mode-mix formulae for computing G_{equivC}: the BK law, the power law, and the Reeder law models. The choice of model is not always clear in any given analysis; an appropriate model is best selected empirically.

BK law

The BK law model is described in Benzeggagh (1996) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC})\left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}}\right)^{\eta}.$$

To define this model, you must provide G_{IC}, G_{IIC}, and η. This model provides a power law relationship combining energy release rates in Mode I, Mode II, and Mode III into a single scalar fracture criterion.

Input File Usage: *FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=BK

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **BK**

Power law

The power law model is described in Wu (1965) by the following formula:

$$\frac{G_{equiv}}{G_{equivC}} = \left(\frac{G_I}{G_{IC}}\right)^{a_m} + \left(\frac{G_{II}}{G_{IIC}}\right)^{a_n} + \left(\frac{G_{III}}{G_{IIIC}}\right)^{a_o}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}, a_m, a_n$, and a_o.

Input File Usage: *FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=POWER

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **Power**

Reeder law

The Reeder law model is described in Reeder (2002) by the following formula:

$$G_{equivC} = G_{IC} + (G_{IIC} - G_{IC})\left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}}\right)^{\eta} +$$

$$(G_{IIIC} - G_{IIC})\left(\frac{G_{III}}{G_{II} + G_{III}}\right)\left(\frac{G_{II} + G_{III}}{G_I + G_{II} + G_{III}}\right)^{\eta}.$$

To define this model, you must provide $G_{IC}, G_{IIC}, G_{IIIC}$, and η. The Reeder law is best applied when $G_{IIC} \neq G_{IIIC}$. When $G_{IIC} = G_{IIIC}$, the Reeder law reduces to the BK law. The Reeder law applies only to three-dimensional problems.

Input File Usage: *FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=REEDER

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT**, **Mixed mode behavior**: **Reeder**

Releasing multiple nodes in one increment in Abaqus/Standard

For an unstable crack growth problem, sometimes it is more efficient to allow multiple nodes at and ahead of a crack tip to debond in one increment without cutting back the increment size when the VCCT fracture criterion is satisfied. This capability is activated automatically if you specify an unstable growth tolerance, f_{tol}^u. In this case if the fracture criterion, f, is within the given unstable growth tolerance:

$$1 + f_{tol} \leq f \leq 1 + f_{tol}^u,$$

where f_{tol} is the tolerance described earlier in this section, rather than cut back the increment size, more nodes at and ahead of the crack tip are allowed to debond in one increment until $f < 1$ for all the nodes ahead of the crack tip. The forces at those debonded nodes are completely released immediately during the following increment. If you do not specify a value for the unstable growth tolerance, the default value is infinity. In this case the fracture criterion, f, for unstable crack growth is not limited by any upper-bound value in the above equation.

Input File Usage: *FRACTURE CRITERION, TYPE=VCCT,
UNSTABLE GROWTH TOLERANCE=f_{tol}^u

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT**, toggle on **Specify tolerance for unstable crack propagation**: *specify value*

Defining variable critical energy release rates

You can define a VCCT criterion with varying energy release rates by specifying the critical energy release rates at the nodes.

If you indicate that the nodal critical energy rates will be specified, any constant critical energy release rates you specify are ignored, and the critical energy release rates are interpolated from the nodes. The critical energy release rates must be defined at all nodes on the slave surface.

Input File Usage: Use both of the following options:

*FRACTURE CRITERION, TYPE=VCCT, NODAL ENERGY RATE
*NODAL ENERGY RATE

Abaqus/CAE Usage: Defining variable critical energy release rates is not supported in Abaqus/CAE.

Enhanced VCCT criterion

This criterion is available only in Abaqus/Standard.

The enhanced VCCT criterion is very similar to the original VCCT criterion described above. As in the original VCCT criterion, the fracture criterion in the general case involving Mode I, II, and III is defined as

$$f = \frac{G_{equiv}}{G_{equivC}} \geq 1.0.$$

The crack-tip node debonds when the fracture criterion reaches the value of 1.0. However, unlike the original VCCT criterion, you can specify two different critical fracture energy release rates: G_C for the onset of a crack and G_C^P for the growth of a crack. When the enhanced VCCT criterion is used in the general case involving Mode I, II, and III fracture, the amount of energy dissipated associated with the release of the debonding force is controlled by the critical equivalent strain energy release rate required to propagate the crack, G_{equivC}^P, rather than by the critical equivalent strain energy release rate required to initiate the crack, G_{equivC}. The formulae for calculating G_{equivC}^P are identical to those used for G_{equivC} for different mixed-mode fracture criteria.

Input File Usage: *FRACTURE CRITERION, TYPE=ENHANCED VCCT

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **Enhanced VCCT**

Low-cycle fatigue criterion

This criterion is available only in Abaqus/Standard.

If you specify the low-cycle fatigue criterion, progressive delamination growth at the interfaces in laminated composites subjected to sub-critical cyclic loadings can be simulated. This criterion can be used only in a low-cycle fatigue analysis using the direct cyclic approach ("Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7). The onset and delamination growth are characterized by using the Paris law, which relates the relative fracture energy release rate to crack growth rates as illustrated in Figure 11.4.3–6. The fracture energy release rates at the crack tips in the interface elements are calculated based on the above mentioned VCCT technique.

The Paris regime is bounded by the energy release rate threshold, G_{thresh}, below which there is no consideration of fatigue crack initiation or growth, and the energy release rate upper limit, G_{pl}, above which the fatigue crack will grow at an accelerated rate. G_C is the critical equivalent strain energy release rate calculated based on the user-specified mode-mix criterion and the bond strength of the interface. The formulae for calculating G_C have been provided above for different mixed mode fracture criteria. You can specify the ratio of G_{thresh} over G_C and the ratio of G_{pl} over G_C. The default values are $\frac{G_{thresh}}{G_C} = 0.01$ and $\frac{G_{pl}}{G_C} = 0.85$.

Input File Usage: *FRACTURE CRITERION, TYPE=FATIGUE

Abaqus/CAE Usage: The low-cycle fatigue criterion is not supported in Abaqus/CAE.

Onset of delamination growth

The onset of delamination growth refers to the beginning of fatigue crack growth at the crack tip along the interface. In a low-cycle fatigue analysis the onset of the fatigue crack growth criterion is characterized by ΔG, which is the relative fracture energy release rate when the structure is loaded between its maximum and minimum values. The fatigue crack growth initiation criterion is defined as

$$f = \frac{N}{c_1 \Delta G^{c_2}} \geq 1.0,$$

where c_1 and c_2 are material constants and N is the cycle number. The interface elements at the crack tips will not be released unless the above equation is satisfied and the maximum fracture energy release

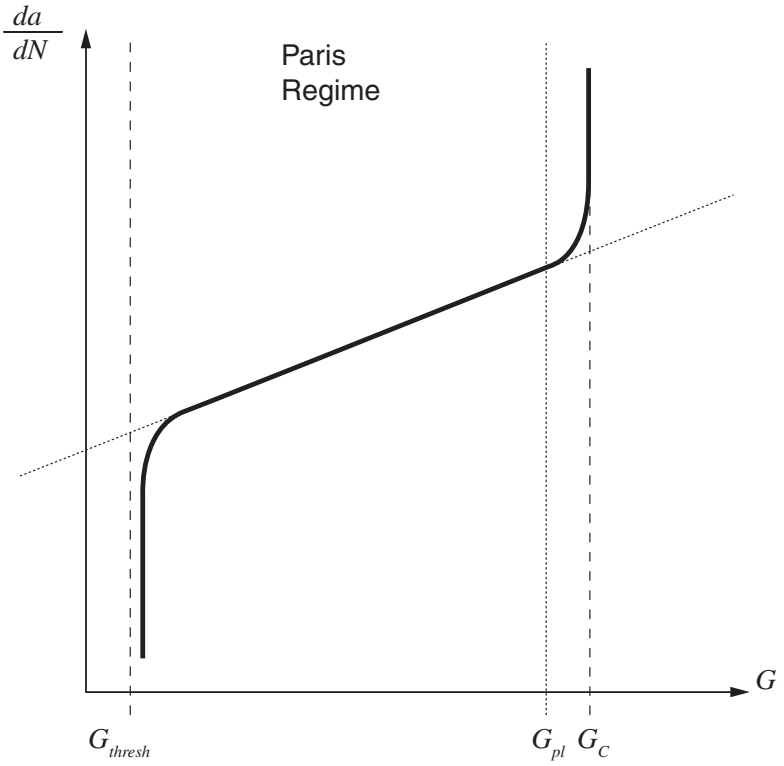


Figure 11.4.3–6 Fatigue crack growth govern by Paris law.

rate, G_{max}, which corresponds to the cyclic energy release rate when the structure is loaded up to its maximum value, is greater than G_{thresh}.

Fatigue delamination growth using the Paris law

Once the onset of delamination growth criterion is satisfied at the interface, the delamination growth rate, da/dN, can be calculated based on the relative fracture energy release rate, ΔG. The rate of the delamination growth per cycle is given by the Paris law if $G_{thresh} < G_{max} < G_{pl}$,

$$\frac{da}{dN} = c_3 \Delta G^{c_4},$$

where c_3 and c_4 are material constants.

At the end of cycle N, Abaqus/Standard extends the crack length, a_N, from the current cycle forward over an incremental number of cycles, ΔN to $a_{N+\Delta N}$ by releasing at least one element at the interface. Given the material constants c_3 and c_4, combined with the known node spacing $\Delta a_{N\,j} = a_{N+\Delta N} - a_N$ at the interface elements at the crack tips, the number of cycles necessary to fail each

interface element at the crack tip can be calculated as ΔN_j, where j represents the node at the $j$the crack tip. The analysis is set up to release at least one interface element after the loading cycle is stabilized. The element with the fewest cycles is identified to be released, and its $\Delta N_{min} = min(\Delta N_j)$ is represented as the number of cycles to grow the crack equal to its element length, $\Delta a_{N\,min} = min(\Delta a_{N\,j})$. The most critical element is completely released with a zero constraint and a zero stiffness at the end of the stabilized cycle. As the interface element is released, the load is redistributed and a new relative fracture energy release rate must be calculated for the interface elements at the crack tips for the next cycle. This capability allows at least one interface element at the crack tips to be released after each stabilized cycle and precisely accounts for the number of cycles needed to cause fatigue crack growth over that length.

If $G_{max} > G_{pl}$, the interface elements at the crack tips will be released by increasing the cycle number count, dN, by one only.

Specifying how a debonding force is released after a fracture criterion is met in Abaqus/Standard

After debonding, the traction between two surfaces is initially carried as equal and opposite forces at the slave node and the corresponding point on the master surface. The debonding force is released as the crack opens and advances. Once complete debonding has occurred at a point, the bond surfaces act like standard contact surfaces with associated interface characteristics. There are two different ways to release the debonding force, depending on the fracture criterion that you specify.

Specifying a debonding amplitude curve

When you use the critical stress, critical crack opening displacement, or crack length versus time fracture criteria, you can define how this force is to be reduced to zero with time after debonding starts at a particular node on the bonded surface. You specify a relative amplitude, a, as a function of time after debonding starts at a node. Thus, suppose the force transmitted between the surfaces at slave node N is $\mathbf{T}^N|_0$ when that node starts to debond, which occurs at time $t^N|_0$. Then, for any time $t > t^N|_0$ the force transmitted between the surfaces at node N is $a(t - t^N|_0)\mathbf{T}^N|_0$. The relative amplitude must be 1.0 at the relative time 0.0 and must reduce to 0.0 at the last relative time point given.

The best choice of the amplitude curve depends on the material properties, specified loading, and the crack propagation criterion. If the stresses are removed too rapidly, the resulting large changes in the strains near the crack tip can cause convergence difficulties. For large-strain problems severe mesh distortion can also occur. For problems with rate-independent materials a linear amplitude curve is normally adequate. For problems with rate-dependent materials the stresses should be ramped off more slowly at the beginning of debonding to avoid convergence and mesh distortion difficulties. To reduce the likelihood of convergence and mesh distortion difficulties, you can reduce the value of the debond stress by 25% in 50% of the time to debond. The solution should not be strongly influenced by the details of the unloading procedure; if it is, this usually indicates that the mesh should be refined in the debond region.

Input File Usage: ∗DEBOND, SLAVE=*slave*, MASTER=*master*
Data lines to define debonding amplitude curve

Abaqus/CAE Usage: Specifying a debonding amplitude curve is not supported in Abaqus/CAE.

Ramping down debonding force for the VCCT and the enhanced VCCT criteria

For the VCCT and the enhanced VCCT criteria, when the energy release rate exceeds the critical value at a crack tip, you can either release the traction between the two surfaces at the crack tip immediately during the following increment or release the traction gradually during succeeding increments with the reduction of the magnitude of the debonding force being governed by the critical fracture energy release rate. The latter approach is sometimes recommended to avoid sudden loss of stability when the crack tip is advanced. The enhanced VCCT criterion is meaningful only when used in conjunction with the latter approach. When the former approach is used, the results obtained by using the enhanced VCCT criterion are identical to those obtained by using the original VCCT criterion.

Input File Usage: Use the following option to release the traction immediately:

∗DEBOND, SLAVE=*slave*, MASTER=*master*,
DEBONDING FORCE=STEP

Use the following option to release the traction gradually:

∗DEBOND, SLAVE=*slave*, MASTER=*master*,
DEBONDING FORCE=RAMP

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: **Name:** *crack name*, **Type: Debond using VCCT**, select the step and the surface to surface (Standard) interaction, **Debonding force: Step** or **Ramp**

Procedures

Crack propagation analysis can be performed for static or dynamic overloadings using the following procedures:

- "Static stress analysis," Section 6.2.2
- "Quasi-static analysis," Section 6.2.5
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Explicit dynamic analysis," Section 6.3.3
- "Fully coupled thermal-stress analysis," Section 6.5.3

It can also be performed for sub-critical cyclic fatigue loadings using the following procedure:

- "Low-cycle fatigue analysis using the direct cyclic approach," Section 6.2.7

Controlling time incrementation during debonding in Abaqus/Standard

When automatic incrementation is used for any criteria other than VCCT, enhanced VCCT, or low-cycle fatigue, you can specify the size of the time increment used just after debonding starts. By default, the time increment is equal to the last relative time specified. However, if a fracture criterion is met at the beginning of an increment, the size of the time increment used just after debonding starts will be set equal to the minimum time increment allowed in this step.

For fixed time incrementation the specified time increment value will be used as the time increment size after debonding starts if Abaqus/Standard finds it needs a smaller time increment than the fixed time increment size. The time increment size will be modified as required until debonding is complete.

Input File Usage: *DEBOND, SLAVE=*slave*, MASTER=*master*, TIME INCREMENT=*t*

Abaqus/CAE Usage: Controlling time incrementation during debonding is not supported in Abaqus/CAE.

Viscous regularization for VCCT in Abaqus/Standard

The simulation of structures with unstable propagating cracks is challenging and difficult. Nonconvergent behavior may occur from time to time. While the usual stabilization techniques (such as contact pair stabilization and static stabilization) can be used to overcome some convergence difficulties, localized damping is included for VCCT or enhanced VCCT by using the viscous regularization technique. Viscous regularization damping causes the tangent stiffness matrix of the softening material to be positive for sufficiently small time increments.

Input File Usage: Use one of following options:

*FRACTURE CRITERION, TYPE=VCCT, VISCOSITY=μ

*FRACTURE CRITERION, TYPE=ENHANCED VCCT, VISCOSITY=μ

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Contact**, **Mechanical→Fracture Criterion**, **Type**: **VCCT** or **Enhanced VCCT**, **Viscosity**

Linear scaling to accelerate convergence for VCCT in Abaqus/Standard

For most crack propagation simulations using VCCT or the enhanced VCCT criterion, the deformation can be nearly linear up to the point of the onset of crack growth; past this point the analysis becomes very nonlinear. In this case a linear scaling method can be used to effectively reduce the solution time to reach the onset of crack growth.

Suppose that an applied "trial" load at increment $t = t_i$ is just a fraction of the critical load at the onset time of crack growth, $t = t_{crit}$. The following algorithm is used in Abaqus/Standard to quickly converge to the critical load state:

$$\Delta t_{i+1} = \left(\beta_i \sqrt{\frac{G_{equivC}}{G_{equiv}}} - 1 \right) t_i,$$

where initially β_i would be set between 0.7 and 0.9 depending on the degree of nonlinearity (the default value is 0.9). When Δt_{i+1} becomes smaller than 0.5% (indicating that the load is within 0.5% of its critical value), the next β_i is automatically set to 1.0 to cause the most critical crack-tip node to precisely reach the critical value at the next increment. After the first crack-tip node releases, the linear scaling calculations are no longer valid and the time increment is set to the default value. Cutback is then allowed.

Input File Usage: *CONTROLS, TYPE=VCCT LINEAR SCALING

Abaqus/CAE Usage: Step module: **Other→General Solution Controls→Edit**: *step name*, **VCCT Linear Scaling**

Tips for using the VCCT or enhanced VCCT criterion in Abaqus/Standard

Crack propagation problems using the VCCT or enhanced VCCT criterion are numerically challenging. The following tips will help you create a successful Abaqus/Standard model:

- An analysis with the VCCT or enhanced VCCT criterion requires small time increments. Abaqus/Standard tracks the location of the active crack front node by node when the VCCT or enhanced VCCT criterion is used. Therefore, the crack front is allowed to advance only a single node forward in any single increment (although such an advance may take place across the entire crack front in three-dimensional problems). Because an analysis using the VCCT or enhanced VCCT criterion provides detailed results of the growth of the crack, you will need small time increments, especially if the mesh is highly refined.
- Three different types of damping can be used to aid convergence for a model using the VCCT or enhanced VCCT criterion: contact stabilization, automatic or static stabilization, and viscous regularization. Contact and automatic stabilization are not specific to VCCT; they are built into Abaqus/Standard and are compatible with VCCT. Setting the value of the damping parameters is often an iterative procedure. If your VCCT model fails to converge due to unstable crack propagation, set the damping parameters to relatively high values and rerun the analysis. If the parameters are high enough, stable incrementation should return. However, the crack propagation behavior may have been modified by the damping forces and may not be physically correct. To monitor the energy absorbed by viscous damping, plot the damping energy and compare the results to the total strain energy in the model (ALLSE). When set properly, the value of the damping energy should be a small fraction of the total energy. Monitor the damping energy to ensure that the results of the VCCT simulation are reasonable in the presence of damping. When you use contact or automatic stabilization, Abaqus writes the damping energy to the variable ALLSD in the output database (**.odb**) file. When you use viscous regularization, Abaqus writes the damping energy to the variable ALLVD.
- To maximize the accuracy of the debonding simulation, try to use matched meshes between the slave and master surfaces of the debonding contact pair.
- If you do use a mismatched mesh, you can maximize the accuracy of the simulation by using the small-sliding, surface-to-surface formulation for the contact pair (see "Contact formulations in Abaqus/Standard," Section 38.1.1).
- Printing contact constraint information to the data (**.dat**) file allows you to review the status of the debonding contact pair at the beginning of the analysis. By printing detailed contact conditions to the message (**.msg**) file, you can track the incremental behavior of the advancing crack front during the analysis. For more information about these output requests, see "Output," Section 4.1.1.
- You can add a small clearance to the initially unbonded portion of the debonding contact pair ("Adjusting initial surface positions and specifying initial clearances in Abaqus/Standard contact pairs," Section 36.3.5). The small clearance will help to eliminate unnecessary severe discontinuity iterations during incrementation as the crack begins to progress.

- Do not use tie MPCs (“General multi-point constraints,” Section 35.2.2) for the slave surface in a debonding contact pair. Abaqus is unable to resolve the overconstraint presented by the MPC and the debonded contact state.
- You must have continuous master debonding surfaces.
- You may be able to help the analysis converge by adding geometric nonlinearity (even if small-sliding is used for the debonding contact pair). For more information, see “Geometric nonlinearity” in “General and linear perturbation procedures,” Section 6.1.3.
- For two-dimensional models with contact pairs involving higher-order underlying elements, the initially unbonded portion must extend over complete element faces. In other words, the crack tip in a two-dimensional, higher-order model must start at a corner node on the quadratic slave surfaces. The crack tip must not start at a midside node.
- When the surface-to-surface contact formulation is used, at least two rows of elements should be used behind the crack front.

Tips for using the VCCT criterion in Abaqus/Explicit

Crack propagation problems using the VCCT criterion analyzed in Abaqus/Explicit benefit from the robustness of the general contact algorithm in the context of an explicit time integrator. Nevertheless, as is the case in Abaqus/Standard, these analyses remain challenging given the discontinuous nature of the fracture phenomenon. The following tips will help you create a successful Abaqus/Explicit model:

- Dynamic effects are of utmost relevance when assessing the results from a debonding analysis using the VCCT criterion. In most cases experimental and/or theoretical data are available in quasi-static settings. You must ensure that the Abaqus/Explicit analysis generates low ratios of kinetic energy to internal energy (1% or less). In practical terms this requirement often translates into avoiding the use of mass scaling in the model. Use smooth amplitudes to drive the loading to help reduce the kinetic energy in the model. Running the analysis over a longer period of time will not help in most cases because bond breakage is an inherently fast and localized process.
- If appropriate, use damping-like behavior in the materials associated with the debonding plates to reduce dynamic vibrations. Unlike Abaqus/Standard, where a pure static equilibrium is achieved at the end of a converged increment, in Abaqus/Explicit the bond breakage at a given location is associated with a dynamic overshoot beyond the static equilibrium position. If the vibrations are significant (kinetic energy is clearly observable), the dynamic overshoot at nodes behind the crack tip may lead to premature debonding of the crack tip.
- To maximize the accuracy of the debonding simulation, use quad meshes between the slave and master surfaces of the debonding surfaces. Avoid using elements with aspect ratios greater than 2. In most cases mesh refinement will help with obtaining a realistic result.
- Highly mismatched critical energy values between modes tend to induce crack propagation in continuously changing directions in a manner that may be unstable and unrealistic, particularly for modes II and III. Do not use such values unless experimental data suggest so.

- Use frequent field output requests to evaluate the debonding evolution as the analysis progresses. In some cases this can point to nontrivial modeling deficiencies that are difficult to identify from a simple data check analysis.
- Avoid the use of other constraints involving nodes on both surfaces of the debonding interface because the cohesive contact forces will compete with the constraint forces to achieve global equilibrium. Bond breakage might be hard to interpret in these cases.

Comparing VCCT and cohesive elements

Using VCCT to solve delamination problems is very similar to using cohesive elements in Abaqus. Table 11.4.3–2 describes the advantages and disadvantages of the two approaches.

For an example of the use of cohesive elements, see "Delamination analysis of laminated composites," Section 2.7.1 of the Abaqus Benchmarks Guide. This example also shows the effect of viscous regularization on the predicted force-displacement response.

Table 11.4.3–2 Comparing VCCT and cohesive elements.

VCCT	Cohesive Elements
Simulation (mechanics)-driven crack propagation along a known crack surface.	Simulation (mechanics)-driven crack propagation along a known crack surface. However, cohesive elements can also be placed between element faces as a mechanism for allowing individual elements to separate.
Models brittle fracture using LEFM only.	Model brittle or ductile fracture for LEFM or EPFM. Very general interaction modeling capability is possible.
Uses a surface-based framework. Does not require additional elements.	Require definition of the connectivity and interconnectivity of cohesive elements with the rest of the structure. For accuracy, the mesh of cohesive elements may need to be smaller than the surrounding structural mesh and the associated "cohesive zone." As a result, cohesive elements may be more expensive.
Requires a pre-existing flaw at the beginning of the crack surface. Cannot model crack initiation from a surface that is not already cracked.	Can model crack initiation from initially uncracked surfaces. The crack initiates when the cohesive traction stress exceeds a critical value.

VCCT	Cohesive Elements
Crack propagates when strain energy release rate exceeds fracture toughness.	Crack propagates according to cohesive damage model, usually calibrated so that the energy released when the crack is fully open equals the critical strain energy release rate.
Multiple crack fronts/surfaces can be included.	Multiple crack fronts/surfaces can be included.
In Abaqus/Standard crack surfaces are rigidly bonded when uncracked.	Crack surfaces are joined elastically when uncracked in Abaqus/Standard.
Requires user-specified fracture toughness of the bond.	Require user-specified critical traction value and fracture toughness of the bond, as well as elasticity of the bonded surface.

Measuring the critical strain energy release properties for VCCT

You must obtain the critical strain energy release properties of the bonded surfaces for VCCT. The procedure to obtain the critical strain energy release properties is beyond the scope of this guide; however, you can refer to the following ASTM test specifications for guidance:

- ASTM D 5528-94a, "Standard Test Method for Mode I Interlaminar Fracture Toughness of Unidirectional Fiber-Reinforced Polymer Matrix Composites"
- ASTM D 6671-01, "Standard Test Method for Mixed Mode I-Mode II Interlaminar Fracture Toughness of Unidirectional Fiber-Reinforced Polymer Matrix Composites"
- ASTM D 6115-97, "Standard Test Method for Mode I Fatigue Delamination Growth Onset of Unidirectional Fiber-Reinforced Polymer Matrix Composites"

These test specifications can be found in the Annual Book of ASTM Standards, American Society for Testing and Materials, vol. 15.03, 2000.

Initial conditions

Initial contact conditions are used to identify which part of the slave surface is initially bonded, as explained earlier.

Boundary conditions

Boundary conditions should not be applied to any of the nodes on the master or slave crack surfaces, but they can be used to load the structure and cause crack propagation. Boundary conditions can be applied to any of the displacement degrees of freedom in a crack propagation analysis ("Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1). In a low-cycle fatigue analysis, prescribed

boundary conditions must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2).

Loads

The following types of loading can be prescribed in a crack propagation analysis:

- Concentrated nodal forces can be applied to the displacement degrees of freedom (1–6); see "Concentrated loads," Section 34.4.2.
- Distributed pressure forces or body forces can be applied; see "Distributed loads," Section 34.4.3. The distributed load types available with particular elements are described in Part VI, "Elements."

For a low-cycle fatigue analysis each load must have an amplitude definition that is cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2).

Predefined fields

The following predefined fields can be specified in a crack propagation analysis, as described in "Predefined fields," Section 34.6.1:

- Although temperature is not a degree of freedom in stress/displacement elements, nodal temperatures can be specified as predefined fields. The specified temperature affects temperature-dependent critical stress and crack opening displacement failure criteria, if specified.
- The values of user-defined field variables can be specified. These values affect field-variable-dependent critical stress and crack opening displacement failure criteria, if specified.

The temperatures and user-defined field variables on slave and master surfaces are averaged to determine the critical stresses and crack opening displacements.

In a low-cycle fatigue analysis, the temperature values specified must be cyclic over the step: the start value must be equal to the end value (see "Amplitude curves," Section 34.1.2). If the temperatures are read from the results file, you should specify initial temperature conditions equal to the temperature values at the end of the step (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). Alternatively, you can ramp the temperatures back to their initial condition values, as described in "Predefined fields," Section 34.6.1.

Material options

Any of the mechanical constitutive models in Abaqus/Standard can be used to model the mechanical behavior of the cracking material. See Part V, "Materials."

Elements

Regular, rectangular meshes give the best results in crack propagation analyses. Results with nonlinear materials are more sensitive to meshing than results with small-strain linear elasticity.

First-order elements generally work best for crack propagation analysis.

Line spring elements cannot be used in crack propagation analysis.

The VCCT, enhanced VCCT, and low-cycle fatigue criteria not only support two-dimensional models (planar and axisymmetric) but also three-dimensional models with contact pairs involving first-order underlying elements (solids, shells, and continuum shells). In Abaqus/Standard use of the VCCT or enhanced VCCT criterion in two-dimensional models with contact pairs involving higher-order underlying elements is limited to crack fronts that are aligned with the corner nodes of the higher-order element faces. Use of the low-cycle fatigue criterion with contact pairs involving higher-order underlying elements is not supported.

Output

Unless otherwise stated, the following discussions in this section are applied only to the critical stress, critical crack opening displacement, and crack length versus time criteria.

At the start of an analysis Abaqus/Standard will scan the partially bonded surfaces and identify all of the crack tips that are present in the model. The initial contact status of all of the slave surface nodes is printed in the data (`.dat`) file. At this stage Abaqus/Standard will explicitly identify all the crack tips and mark them as crack 1, crack 2, etc. The slave and master surfaces that are associated with these cracks are also identified.

The initial contact status of all of the slave surface nodes is also printed in the data (`.dat`) file for the VCCT, enhanced VCCT, and low-cycle fatigue criteria.

Printing crack propagation information to the data file

By default, crack propagation information will be printed to the data file during the analysis. For each crack that is identified Abaqus/Standard will print out the initial and current crack-tip node numbers, accumulated incremental crack length (distance from the initial crack tip to the current crack tip, measured along the slave surface), and the current value of the user-specified fracture criterion used. Crack propagation information cannot be printed to the data file in Abaqus/Explicit.

Input File Usage: *DEBOND, SLAVE=*slave*, MASTER=*master*

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: **Type: Debond using VCCT**, **Write output to DAT file every** *n* **increments**

For example, if the crack opening displacement criterion is used, the printed output during the analysis will appear as follows in the data file:

```
      CRACK TIP LOCATION AND ASSOCIATED QUANTITIES
CRACK   SLAVE   MASTER  INITIAL  CURRENT  CUMULATIVE  CRITICAL
NUMBER SURFACE SURFACE CRACKTIP CRACKTIP INCREMENTAL COD
                       NODE #   NODE #   LENGTH
   :       :       :        :        :        :          :
```

Writing crack propagation information to the results file

In Abaqus/Standard you can choose to write the crack propagation information to the results (`.fil`) file.

Input File Usage: *DEBOND, SLAVE=*slave*, MASTER=*master*, OUTPUT=FILE

Abaqus/CAE Usage: Writing crack propagation information to the results file is not supported in Abaqus/CAE.

Writing crack propagation information to both the data file and the results file

In Abaqus/Standard you can write the crack propagation information to both the data and the results files.

Input File Usage: *DEBOND, SLAVE=*slave*, MASTER=*master*, OUTPUT=BOTH

Abaqus/CAE Usage: Writing crack propagation information to both the data file and the results file is not supported in Abaqus/CAE.

Controlling the output frequency

In Abaqus/Standard you can control the output frequency in increments. By default, the crack-tip location and associated quantities will be printed every increment. Specify an output frequency of 0 to suppress crack propagation output.

Input File Usage: *DEBOND, SLAVE=*slave*, MASTER=*master*, FREQUENCY=*f*

Abaqus/CAE Usage: Interaction module: **Special→Crack→Create**: **Type: Debond using VCCT**, **Write output to DAT file every** n **increments**

Output variables

The following bond failure quantities can be requested as surface output (see "Output to the data and results files," Section 4.1.2; "Abaqus/Standard output variable identifiers," Section 4.2.1; and "Abaqus/Explicit output variable identifiers," Section 4.2.2) for all fracture criteria:

DBT	The time when bond failure occurred. For the VCCT, enhanced VCCT, and low-cycle fatigue criteria, this is the time when debonding initiates.
DBSF	Fraction of stress at bond failure that still remains.
DBS	All components of remaining stress in the failed bond.
DBS1i	1i component of stress in the failed bond that remains ($i = 1, 2$).

For the VCCT, enhanced VCCT, and low-cycle fatigue criteria, the following additional variables can be also requested as surface output (see "Output to the data and results files," Section 4.1.2):

CSDMG	Overall value of the scalar damage variable.
BDSTAT	Bond state. The bond state varies between 1.0 (fully bonded) and 0.0 (fully unbonded).
OPENBC	Relative displacement behind crack when the fracture criterion is met.
CRSTS	All components of critical stress at failure
CRSTS1i	1i component of critical stress at failure ($i = 1, 3$).
ENRRT	All components of strain energy release rate.
ENRRT1i	1i component of strain energy release rate ($i = 1, 3$).
EFENRRTR	Effective energy release rate ratio, $\frac{G_{equiv}}{G_{equivC}}$.

Surface output requests provide the usual output of contact variables in addition to the above quantities. The bond failure quantities must be requested explicitly; otherwise, only the default output for contact will be given.

Abaqus/CAE provides support for the visualization of time-history plots and X–Y plots of the variables that are written to the output database.

Contour integrals

Contour integrals can be requested for two-dimensional crack propagation analyses performed using the critical stress, critical crack opening displacement, or crack length versus time fracture criteria. If the contours are chosen so that the crack tip passes through the contour, the contour value will go to zero (as it should). Therefore, in crack propagation analysis contour integrals should be requested far enough from the crack tip that the crack tip does not pass through the contour, which is easily done by including all nodes along the bond surface in the crack-tip node set specified. See "Contour integral evaluation," Section 11.4.2, for details on contour integral output.

Input file template

Abaqus/Standard analysis

```
*HEADING
...
*BOUNDARY
Data lines to specify zero-valued boundary conditions
*INITIAL CONDITIONS, TYPE=CONTACT (, NORMAL)
Data lines to specify initial conditions
*SURFACE, NAME=slave
Data lines to define slave surface
*SURFACE, NAME=master
Data lines to define master surface
**
*CONTACT PAIR
slave, master
**
*STEP (, NLGEOM)
*STATIC or *VISCO or *COUPLED TEMPERATURE-DISPLACEMENT
*DEBOND, SLAVE=slave, MASTER=master
Data lines to define debonding amplitude curve
*FRACTURE CRITERION, TYPE=type, DISTANCE or NSET
Data lines to define fracture criterion
*BOUNDARY
Data lines to define zero-valued or nonzero boundary conditions
*CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD
```

Data lines to define loading
**
CONTOUR INTEGRAL, CONTOURS=n***, TYPE=***type*
****Contour integrals can be requested in a two-dimensional crack propagation analysis*
***CONTACT PRINT**
DBT, DBSF, DBS
***EL PRINT**
JK,
***END STEP**
**
***STEP**
***DIRECT CYCLIC, FATIGUE**
DEBOND, SLAVE=slave***, MASTER=***master*
***FRACTURE CRITERION, TYPE=FATIGUE**
Data lines to define material constants used in Paris law and fracture criterion
***BOUNDARY**
Data lines to define zero-valued or nonzero cyclic boundary conditions
***CLOAD and/or *DLOAD and/or *TEMPERATURE and/or *FIELD**
Data lines to define cyclic loading
**
***END STEP**
**

Abaqus/Explicit analysis

***HEADING**
…
***BOUNDARY**
Data lines to specify zero-valued boundary conditions
SURFACE, NAME=slave*
Data lines to define slave surface
SURFACE, NAME=master*
Data lines to define master surface
**
CONTACT CLEARANCE, NAME=clearance_name***,**
SEARCH NSET=*initially_bonded_nodeset_name*
SURFACE INTERACTION, NAME=interaction_name*
***COHESIVE BEHAVIOR**
Data lines to specify elastic behavior
***FRACTURE CRITERION, TYPE=VCCT, MIXED MODE BEHAVIOR=BK**
**
***STEP**
***DYNAMIC, EXPLICIT**

```
*CONTACT
*CONTACT CLEARANCE ASSIGNMENT
```
Data lines to assign a clearance name to a surface pair
```
*CONTACT PROPERTY ASSIGNMENT
```
Data lines to assign a surface interaction to a surface pair
```
*END STEP
**
```

Additional references

- Benzeggagh, M., and M. Kenane, "Measurement of Mixed-Mode Delamination Fracture Toughness of Unidirectional Glass/Epoxy Composites with Mixed-Mode Bending Apparatus," Composite Science and Technology, vol. 56, p. 439, 1996.
- Reeder, J., S. Kyongchan, P. B. Chunchu, and D. R.. Ambur, "Postbuckling and Growth of Delaminations in Composite Plates Subjected to Axial Compression" 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, Colorado, vol. 1746, p. 10, 2002.
- Wu, E. M., and R. C. Reuter Jr., "Crack Extension in Fiberglass Reinforced Plastics," T and M Report, University of Illinois, vol. 275, 1965.

11.5 Surface-based fluid modeling

- "Surface-based fluid cavities: overview," Section 11.5.1
- "Fluid cavity definition," Section 11.5.2
- "Fluid exchange definition," Section 11.5.3
- "Inflator definition," Section 11.5.4

11.5.1 SURFACE-BASED FLUID CAVITIES: OVERVIEW

Products: Abaqus/Standard Abaqus/Explicit

References

- "Fluid cavity definition," Section 11.5.2
- "Fluid exchange definition," Section 11.5.3
- "Inflator definition," Section 11.5.4

Overview

Surface-based fluid-filled cavities are modeled by:

- using standard finite elements to model the fluid-filled structure;
- using a surface definition to provide the coupling between the deformation of the fluid-filled structure and the pressure exerted by the contained fluid on the cavity boundary of the structure;
- defining the fluid behavior;
- using fluid exchange definitions to model the transfer of fluid between a cavity and the environment or between multiple cavities; and
- using inflator definitions to infuse a gas mixture into a fluid cavity to simulate the inflation of an automotive airbag.

The surface-based fluid cavity capability can be used to model a liquid or gas-filled structure. It supersedes the element-based hydrostatic fluid cavity capability in functionality and does not require the user to define fluid or fluid link elements.

Introduction

In certain applications it may be necessary to predict the mechanical response of a liquid-filled or a gas-filled structure. Examples include pressure vessels, hydraulic or pneumatic driving mechanisms, and automotive airbags. A primary difficulty in addressing such applications is the coupling between the deformation of the structure and the pressure exerted by the contained fluid on the structure. Figure 11.5.1–1 illustrates a simple example of a fluid-filled structure subjected to a system of external loads. The response of the structure depends not only on the external loads but also on the pressure exerted by the fluid, which, in turn, is affected by the deformation of the structure. The surface-based fluid cavity capability provides the coupling needed to analyze situations in which the cavity can be assumed completely filled by fluid with uniform properties and state. Applications with significant spatial variation within cavities cannot be modeled with this feature. For example, consider the fluid-structure interaction and coupled Eulerian-Lagrangian capabilities for applications involving sloshing and wave propagation through a fluid (see "Eulerian analysis," Section 14.1.1; "Fluid-structure interaction" in "Co-simulation: overview," Section 17.1.1; and "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2).

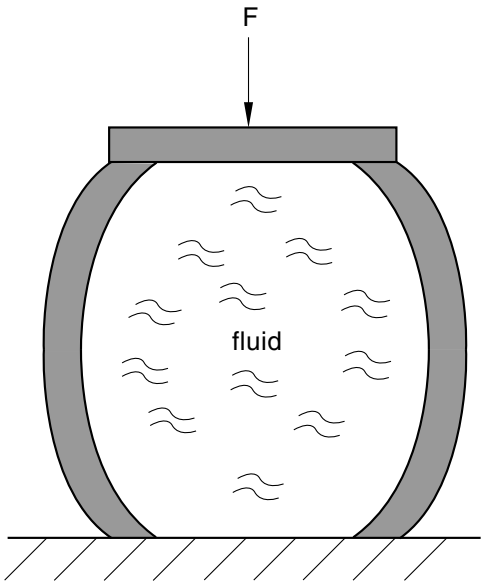


Figure 11.5.1–1 Fluid-filled structure.

Discretizing the fluid cavity

The boundary of the fluid cavity is defined by an element-based surface with normals pointing to the inside of the cavity. The underlying elements can be standard solid or structural elements as well as surface elements. Surface elements can be used to model holes in the structure or to fill in rigid regions where rigid or other load-carrying elements do not exist (see "Surface elements," Section 32.7.1). Care must be taken when using surface elements such that nodes completely surrounded by only surface elements have proper boundary conditions.

Consider the example presented in Figure 11.5.1–1. Solid elements are defined on the top and side of the cavity as indicated in Figure 11.5.1–2. A surface element is defined on the bottom rigid boundary of the cavity where no standard elements exist. The node located at the intersection of the axis of symmetry and the lower rigid boundary of the cavity must be restrained in the r- and z-directions because it is connected only to a surface element. The surface defining the cavity is based on the underlying solid and surface elements.

An additional user-defined volume can be added to the actual or geometric volume of the cavity. If the boundary of the cavity is not defined by an element-based surface, the fluid cavity is assumed to have a fixed volume that is equal to the added volume.

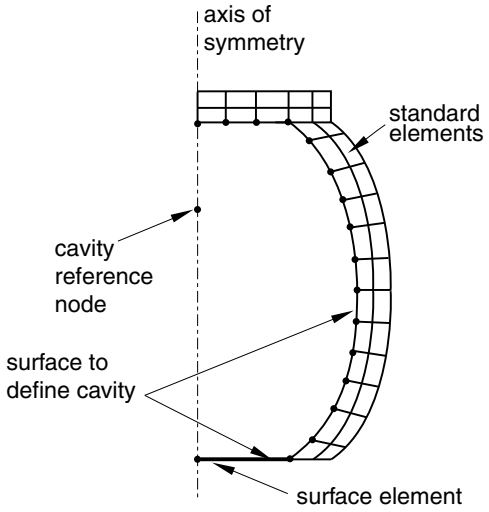


Figure 11.5.1–2 Axisymmetric model of fluid-filled structure.

Defining the location of the cavity reference node

A single node, known as the cavity reference node, is associated with a fluid cavity. This cavity reference node has a single degree of freedom representing the pressure inside the fluid cavity. The cavity reference node is also used in calculating the cavity volume.

If the cavity is not bounded by symmetry planes, the surface defining the cavity must completely enclose the cavity to ensure proper calculation of its volume. In this case the location of the cavity reference node is arbitrary and does not have to lie inside the cavity.

If, as a result of symmetry, only a portion of the cavity boundary is modeled with standard elements, the cavity reference node must be located on the symmetry plane or axis (Figure 11.5.1–2). If multiple symmetry planes exist, the cavity reference node must be located on the intersection of the symmetry planes (Figure 11.5.1–3). For an axisymmetric analysis the cavity reference node must be located on the axis of symmetry. These requirements are a consequence of the fluid cavity not being fully enclosed by the surface defining the cavity.

Finite element calculations

The finite element calculations for surface-based cavities are performed using volume elements as described in "Hydrostatic fluid calculations," Section 3.8.1 of the Abaqus Theory Guide. The volume elements for a cavity are created internally by Abaqus using the surface facet geometry and the cavity reference node that you define. In Abaqus/Standard the surface facets are represented with the following element types: FAX2 and F2D2 (which are linear, 2-node, axisymmetric and planar elements, respectively) and F3D3 and F3D4 (which are linear, 3-node and 4-node three-dimensional elements,

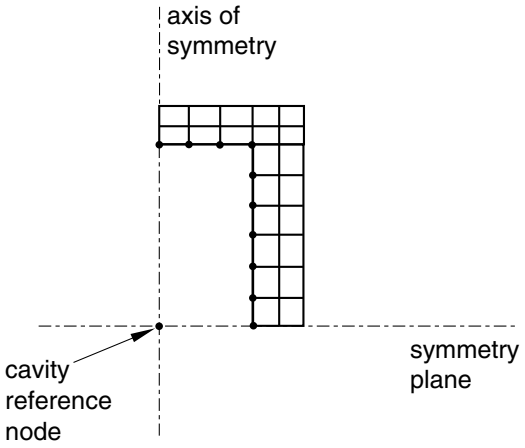


Figure 11.5.1–3 Axisymmetric model with additional symmetry plane.

respectively). Second-order facets in Abaqus are subdivided further into multiple linear facets or elements.

Fluid cavity behavior

The behavior of the fluid within the fluid-filled cavity can be based either on a hydraulic or a pneumatic model. The hydraulic model can simulate nearly incompressible fluid behavior and fully incompressible behavior in Abaqus/Standard. The compressibility is introduced by defining a bulk modulus. The pneumatic model is based on an ideal gas. The gas can be defined by multiple species in Abaqus/Explicit, and you can specify the temperature of the gas or have it calculated based on the assumption of adiabatic behavior. A multi-species ideal gas with an adiabatic temperature update is an appropriate model for automotive airbags.

Modeling flow into or out of a cavity

There are many ways in Abaqus to model the transfer of fluid into or out of a cavity. The flow can be specified as a prescribed mass or volume flux history or can model physical mechanisms due to a pressure differential such as venting through an exhaust orifice or leakage through a porous fabric. Fluid exchange definitions are used for this purpose and can model flow between a fluid cavity and its environment or between two fluid cavities (see "Fluid exchange definition," Section 11.5.3, for details). In addition, Abaqus/Explicit has the capability to model inflators used for the deployment of automotive airbags. Conditions at the inflator can be specified directly, or tank test data can be used (see "Inflator definition," Section 11.5.4, for details).

Modeling multiple chambers

Many fluid-filled systems such as airbags have multiple chambers with fluid flowing between chambers through holes or fabric leakage. In other cases it is advantageous to divide a single physical chamber into multiple chambers with fictitious walls to model a gradient in pressure across the physical chamber. Some fictitious leakage mechanisms through inter-chamber walls can be defined to obtain reasonable behavior. This can be a useful modeling technique when simulating the complex unfolding of an airbag. To model multiple chambers, define a fluid cavity for each chamber and link the fluid cavities together with the appropriate fluid exchange definitions. Averaged properties for the multi-chambered model can be output if requested (see "Fluid cavity definition," Section 11.5.2, for details).

Defining the fluid inertia in a dynamic procedure

The inertia of the fluid inside a fluid cavity or fluid exchanged between cavities is not automatically taken into account. To add the effect of inertia, use MASS elements on the boundary of the cavity. You should make sure that the total added mass corresponds to the mass of the fluid in the cavity and that the distribution of the MASS elements is a reasonable representation of the distributed fluid mass for the type of loading to which the structure is subjected. Only the overall effect of the fluid inertia can be modeled; the uniform pressure assumption in the cavity makes it impossible to model any pressure gradient-driven fluid motions. Thus, the approach assumes that the time scale of the excitation is very long compared to typical response times for the fluid.

Modeling contact involving the cavity boundary

If a large amount of fluid is removed from a cavity or the material surrounding the cavity is very flexible, the cavity may partially collapse and portions of the cavity walls may contact each other. Self-contact of the cavity walls and contact with surrounding structures can be handled effectively by using the standard techniques available in Abaqus for modeling contact. Abaqus/Explicit can also account for the blockage of flow out of a cavity due to contacting surfaces (see "Accounting for blockage due to contacting boundary surfaces" in "Fluid exchange definition," Section 11.5.3).

Interpreting negative eigenvalue messages

In some applications in Abaqus/Standard, negative eigenvalues may be encountered during the solution. These negative eigenvalues do not necessarily indicate that a bifurcation or buckling load has been exceeded. If the predicted response otherwise appears to be reasonable, these messages can be ignored. A detailed description of how negative eigenvalues can develop during the solution of hydrostatic fluid element problems is presented in "Hydrostatic fluid calculations," Section 3.8.1 of the Abaqus Theory Guide.

Procedures

The surface-based fluid cavity capability can be used in all procedures except coupled pore fluid diffusion/stress analysis (see "Coupled pore fluid diffusion and stress analysis," Section 6.8.1).

Initial conditions

The initial fluid pressure and temperature can be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). For an ideal gas the initial pressure represents the gauge pressure over and above the ambient pressure. The initial temperature should be given in the temperature scale used. Absolute zero in that temperature scale is specified separately for an ideal gas (see "Fluid cavity definition," Section 11.5.2).

If membrane elements are used as the underlying elements for the fluid cavity, the reference mesh (initial metric) can also be specified (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1).

Boundary conditions

The pressure degree of freedom at the cavity reference node (degree of freedom number 8) is a primary variable in the problem. Thus, it can be prescribed by defining a boundary condition (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1), similar to the way displacements of structural nodes can be prescribed. Prescribing the pressure at the cavity reference node is equivalent to applying a uniform pressure to the cavity boundary using a distributed load definition (see "Distributed loads," Section 34.4.3).

If the pressure is prescribed with a boundary condition, the fluid volume is adjusted automatically to fill the cavity (that is, fluid is assumed to enter and leave the cavity as needed to maintain the prescribed pressure). This behavior is useful in situations where a cavity is deformed prior to the introduction of the effect of the fluid. In a subsequent step you can remove the boundary condition on the pressure degree of freedom (see "Removing boundary conditions" in "Boundary conditions in Abaqus/CFD," Section 34.3.2), thus "sealing" the cavity with the current fluid volume.

Loads

Distributed pressures and body forces, as well as concentrated nodal forces, can be applied to the fluid-filled structure, as described in "Concentrated loads," Section 34.4.2, and "Distributed loads," Section 34.4.3.

Predefined fields

Predefined temperature fields and user-defined field variables can be defined for both fluid-filled structures and the enclosed fluids, as described in "Predefined fields," Section 34.6.1.

Temperatures

Fluid temperatures can be specified at all cavity reference nodes as predefined fields (see "Predefined temperature" in "Predefined fields," Section 34.6.1), unless an adiabatic process is specified or a coupled temperature-displacement procedure is used. Any difference between the applied and initial temperatures will cause thermal expansion for a pneumatic fluid and for a hydraulic fluid if a thermal

expansion coefficient is given. A specified temperature field can also affect temperature-dependent material properties, if any exist, for both fluid-filled structures and enclosed fluids.

Field variables

The values of user-defined field variables can be specified at all cavity reference nodes (see "Predefined field variables" in "Predefined fields," Section 34.6.1). These values will affect field-variable-dependent material properties for the enclosed fluid.

Output

The state of the fluid inside the cavity is available for history output using the nodal output variables PCAV and CVOL, which represent the gauge fluid pressure and cavity volume, respectively. In steady-state dynamic procedures the magnitude and phase angle of the fluid pressure can be obtained as nodal variable PPOR.

Abaqus/Explicit also provides output for the cavity temperature, cavity surface area, and mass of the fluid (nodal output variables CTEMP, CSAREA, and CMASS, respectively). Output variable CTEMP is available only when an ideal gas model is used under adiabatic conditions. If the node set for which the output request is made contains more than one fluid cavity, the time histories of the average fluid pressure, total volume, average fluid temperature, sum of all the external cavity surface areas, and total mass of these cavities will also be output by using the nodal output variables APCAV, TCVOL, ACTEMP, TCSAREA, and TCMASS, respectively.

In Abaqus/Explicit, when the model includes fluid exchange definitions, use nodal output variables CMFL and CMFLT to obtain history output of the total mass flow rate and total accumulated mass flow out of a cavity and CEFL and CEFLT to obtain history output of the total heat energy flow rate and total accumulated heat energy flow out of a cavity. If more than one fluid exchange is defined for a cavity, time histories of the mass or heat energy flow rate and accumulated mass or heat energy flow out of the cavity for each fluid exchange will also be output.

If the fluid cavity is modeled by a mixture of ideal gases, time histories of the molecular mass fraction of each fluid species inside the fluid cavity can be obtained by using nodal output variable CMF.

If inflators are used, use nodal output variables MINFL, MINFLT, and TINFL to obtain time histories of mass flow rate, accumulated mass flow, and inflator temperature for each inflator definition (see "Abaqus/Explicit output variable identifiers," Section 4.2.2).

Input file template

An analysis with hydrostatic fluid:

```
*HEADING
…
*FLUID CAVITY, NAME=cavity_name, BEHAVIOR=behavior_name,
REF NODE=cavity_reference_node, SURFACE=surface_name
*FLUID BEHAVIOR, NAME=behavior_name
*FLUID DENSITY
Data line to define density
```

```
*FLUID BULK MODULUS
Data line to define  bulk modulus
*FLUID EXPANSION
Data line to define thermal expansion
**
*FLUID EXCHANGE, NAME=exchange_name, PROPERTY=exchange_property_name
cavity_reference_node
*FLUID EXCHANGE PROPERTY, NAME=exchange_property_name, TYPE=MASS FLUX
Data line to define mass flow rate per unit area
**
*INITIAL CONDITIONS, TYPE=TEMPERATURE
Data line to define initial temperature
*INITIAL CONDITIONS, TYPE=FLUID PRESSURE
Data line to define initial pressure
**
*STEP
**
*TEMPERATURE
Data line to define temperature
*FLUID EXCHANGE ACTIVATION
exchange_name
**
*END STEP
```

An airbag analysis with a mixture of ideal gases:

```
*HEADING
…
*FLUID CAVITY, NAME=chamber_1, MIXTURE=MOLAR FRACTION, ADIABATIC,
REF NODE=chamber_1_reference_node, SURFACE=surface_name_1
blank line
Oxygen,  0.2
Nitrogen,  0.75
Carbon_dioxide,  0.05
**
*FLUID CAVITY, NAME=chamber_2, BEHAVIOR=Air, ADIABATIC,
REF NODE=chamber_2_reference_node, SURFACE=surface_name_2
blank line
**
*FLUID BEHAVIOR, NAME=Air
*CAPACITY, TYPE=POLYNOMIAL
Data line to define heat capacity coefficient
*MOLECULAR WEIGHT
```

Data line to define molecular weight
**
FLUID BEHAVIOR, NAME=Oxygen*
***CAPACITY, TYPE=POLYNOMIAL**
Data line to define heat capacity coefficient
***MOLECULAR WEIGHT**
Data line to define molecular weight
**
FLUID BEHAVIOR, NAME=Nitrogen*
***CAPACITY, TYPE=POLYNOMIAL**
Data line to define heat capacity coefficient
***MOLECULAR WEIGHT**
Data line to define molecular weight
**
FLUID BEHAVIOR, NAME=Carbon_dioxide*
***CAPACITY, TYPE=POLYNOMIAL**
Data line to define heat capacity coefficient
***MOLECULAR WEIGHT**
Data line to define molecular weight
**
FLUID INFLATOR, NAME=inflator*, **PROPERTY=***inflator_property*
chamber_1_reference_node
FLUID INFLATOR PROPERTY, NAME=inflator_property*,
TYPE=TEMPERATURE AND MASS
Data lines to define mass flow rate and gas temperature
FLUID INFLATOR MIXTURE, TYPE=MOLAR FRACTION, NUMBER SPECIES=2*
Carbon_dioxide, Nitrogen
Table to define molecular mass fraction
**
FLUID EXCHANGE, NAME=exhaust*, **PROPERTY=***exhaust_behavior*
chamber_1_reference_node
FLUID EXCHANGE PROPERTY, NAME=exhaust_behavior*, **TYPE=ORIFICE**
Data line to specify orifice behavior
FLUID EXCHANGE, NAME=leakage_1*, **PROPERTY=***fabric_behavior*
chamber_1_reference_node
FLUID EXCHANGE, NAME=leakage_2*, **PROPERTY=***fabric_behavior*
chamber_2_reference_node
FLUID EXCHANGE PROPERTY, NAME=fabric_behavior*, **TYPE=FABRIC LEAKAGE**
Data line to specify fabric leakage behavior
**
FLUID EXCHANGE, NAME=chamber_wall*, **PROPERTY=***wall_behavior*,
EFFECTIVE AREA=

chamber_1_reference_node, chamber_2_reference_node
FLUID EXCHANGE PROPERTY, NAME=wall_behavior***, TYPE=ORIFICE**
Data line to specify orifice behavior
******
AMPLITUDE, NAME=amplitude_name*
Data line to define amplitude variations
***PHYSICAL CONSTANTS, UNIVERSAL GAS CONSTANT=**
******
***INITIAL CONDITIONS, TYPE=FLUID PRESSURE**
Data line to define initial pressure
***INITIAL CONDITIONS, TYPE=TEMPERATURE**
Data line to define initial temperature
******
***STEP**
******
***FLUID EXCHANGE ACTIVATION**
exhaust, leakage_1, leakage_2, chamber_wall
FLUID INFLATOR ACTIVATION, INFLATION TIME AMPLITUDE=amplitude_name*
inflator
******
***END STEP**

11.5.2 FLUID CAVITY DEFINITION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Surface-based fluid cavities: overview," Section 11.5.1
- "Fluid exchange definition," Section 11.5.3
- *CAPACITY
- *FLUID BEHAVIOR
- *FLUID BULK MODULUS
- *FLUID CAVITY
- *FLUID DENSITY
- *MOLECULAR WEIGHT
- "Defining a fluid cavity interaction," Section 15.13.11 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a fluid cavity interaction property," Section 15.14.4 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A surface-based fluid cavity:

- can be used to model a liquid-filled or gas-filled structure;
- is associated with a node known as the cavity reference node;
- is defined by specifying a surface that fully encloses the cavity;
- is applicable only for situations where the pressure and temperature of the fluid in a particular cavity are uniform at any point in time;
- can be used to model an airbag using the assumptions of an ideal gas mixture under adiabatic conditions; and
- has a name that can be used to identify history output associated with the cavity.

Defining the fluid cavity

You must associate a name with each fluid cavity.

Input File Usage: *FLUID CAVITY, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**, **Name**: *name*

Specifying the cavity reference node

Every fluid cavity must have an associated cavity reference node. Along with the fluid cavity name, the reference node is used to identify the fluid cavity. In addition, it may be referenced by fluid exchange and inflator definitions. The reference node should not be connected to any elements in the model.

Input File Usage: *FLUID CAVITY, REF NODE=*n*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: select the fluid cavity reference node

Specifying the boundary of the fluid cavity

The fluid cavity must be completely enclosed by finite elements unless symmetry planes are modeled (see "Surface-based fluid cavities: overview," Section 11.5.1). Surface elements can be used for portions of the cavity surface that are not structural. The boundary of the cavity is specified using an element-based surface covering the elements that surround the cavity with surface normals pointing inward. By default, an error message is issued if the underlying elements of the surface do not have consistent normals. Alternatively, you can skip the consistency checking for the surface normals.

Input File Usage: Use the following option to define the surface with consistent normal checking:

*FLUID CAVITY, SURFACE=*surface_name*, CHECK NORMALS=YES

Use the following option to define the surface without consistent normal checking:

*FLUID CAVITY, SURFACE=*surface_name*, CHECK NORMALS=NO

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: select the fluid cavity boundary surface; toggle on or off **Check surface normals**

Specifying additional volume in a fluid cavity

An additional volume can be specified for a fluid cavity. The additional volume will be added to the actual volume when the boundary of the cavity is defined by a specified surface. If you do not specify a surface forming the boundary of the fluid cavity, the fluid cavity is assumed to have a fixed volume that is equal to the added volume. In Abaqus/Standard, along with the added volume, the surface forming the boundary of the fluid cavity must be specified.

Input File Usage: *FLUID CAVITY, ADDED VOLUME=*r*

Abaqus/CAE Usage: Specification of additional volume is not supported in Abaqus/CAE.

Specifying the minimum volume

When the volume of a fluid cavity is extremely small, transients in an explicit dynamic procedure can cause the volume to go to zero or even negative causing the effective cavity stiffness values to tend to infinity. To avoid numerical problems, you can specify a minimum volume for the fluid in Abaqus/Explicit. If the volume of the cavity (which is equal to the actual volume plus the added volume) drops below the minimum, the minimum value is used to evaluate the fluid pressure.

You can specify the minimum volume either directly or as the initial volume of the fluid cavity. If the latter method is used and the initial volume of the fluid cavity is a negative value, the minimum volume is set equal to zero.

Input File Usage: Use the following option to specify the minimum volume directly:

∗FLUID CAVITY, MINIMUM VOLUME=*minimum volume*

Use the following option to specify the minimum volume to be equal to the initial volume:

∗FLUID CAVITY, MINIMUM VOLUME=INITIAL VOLUME

Abaqus/CAE Usage: Specification of a minimum volume is not supported in Abaqus/CAE.

Defining the fluid cavity behavior

The fluid cavity behavior governs the relationship between cavity pressure, volume, and temperature. A fluid cavity in Abaqus/Standard can contain only a single fluid. In Abaqus/Explicit a cavity can contain a single fluid or a mixture of ideal gases.

Fluid behavior with a homogeneous fluid

To define a fluid cavity behavior made of a single fluid, specify a single fluid behavior to define the fluid properties. You must associate the fluid behavior with a name. This name can then be used to associate a certain behavior with a fluid cavity definition.

Input File Usage: Use the following options:

∗FLUID CAVITY, NAME=*fluid_cavity_name*, BEHAVIOR=*behavior_name*
∗FLUID BEHAVIOR, NAME=*behavior_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**, **Name**: *behavior_name*

Fluid behavior with a mixture of ideal gases in Abaqus/Explicit

In Abaqus/Explicit you can define a fluid cavity behavior made of multiple gas species. To define a fluid cavity behavior made of multiple gas species, you specify multiple fluid behaviors to define the fluid properties. Specify the names of the fluid behaviors and the initial mass or molar fractions defining the mixture to associate a certain group of behaviors with a fluid cavity definition.

Input File Usage: Use the following options to define the fluid cavity mixture in terms of the initial mass fraction:

∗FLUID BEHAVIOR, NAME=*behavior_name*
∗FLUID CAVITY, NAME=*fluid_cavity_name*,
MIXTURE=MASS FRACTION
out-of-plane surface thickness (if required; otherwise, blank)
behavior_name, *initial mass fraction*
...

Use the following options to define the fluid cavity mixture in terms of the initial molar fraction:

∗FLUID BEHAVIOR, NAME=*behavior_name*
∗FLUID CAVITY, NAME=*fluid_cavity_name*,
MIXTURE=MOLAR FRACTION
out-of-plane surface thickness (if required; otherwise, blank)
behavior_name, *initial molar fraction*
...

Abaqus/CAE Usage: Specification of ideal gas mixtures is not supported in Abaqus/CAE.

User-defined fluid behavior in Abaqus/Standard

In Abaqus/Standard the fluid behavior can be defined in user subroutine **UFLUID**.

Input File Usage: ∗FLUID BEHAVIOR, USER

Abaqus/CAE Usage: User subroutine **UFLUID** is not supported in Abaqus/CAE.

Defining the ambient pressure for a fluid cavity

For pneumatic fluids the equilibrium problem is generally expressed in terms of the "gauge" pressure in the fluid cavity (that is, ambient atmospheric pressure does not contribute to the loading of the solid and structural parts of the system). You can choose to convert gauge pressure to absolute pressure $\tilde{p}$ as used in the constitutive law. For hydraulic fluids you can define the ambient pressure, which can be used to calculate the pressure difference in the fluid exchange between a fluid cavity and its environment. The pressure value given as degree of freedom 8 at the cavity reference node is the value of the gauge pressure. The ambient pressure, p_A, is assumed to be zero if you do not specify it.

Input File Usage: ∗FLUID CAVITY, AMBIENT PRESSURE=p_A

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: toggle on **Specify ambient pressure**: p_A

Isothermal process

For hydraulic fluids and pneumatic fluids in problems of long time duration, it is reasonable to assume that the temperature is constant or a known function of the environment surrounding the cavity. In this case the temperature of the fluid can be defined by specifying initial conditions (see "Defining initial temperatures" in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) and predefined temperature fields (see "Predefined temperature" in "Predefined fields," Section 34.6.1) at the cavity reference node. For a pneumatic fluid the pressure and density of the gas are calculated from the ideal gas law, conservation of mass, and the predefined temperature field.

Defining the ambient temperature for a fluid cavity

For pneumatic fluids with adiabatic behavior the ambient temperature is needed when the heat energy flow is defined between a single cavity and its environment and the flow definition is based on analysis conditions. The ambient temperature, θ_A, is assumed to be zero if you do not specify it.

Input File Usage: *FLUID CAVITY, AMBIENT TEMPERATURE=θ_A

Abaqus/CAE Usage: Specification of ambient temperature is not supported in Abaqus/CAE.

Hydraulic fluids

The hydraulic fluid model is used to model nearly incompressible fluid behavior and fully incompressible fluid behavior in Abaqus/Standard. Compressibility is introduced by assuming a linear pressure-volume relationship. The required parameters for compressible behavior are the bulk modulus and the reference density. You omit the bulk modulus to specify fully incompressible behavior in Abaqus/Standard. Temperature dependence of the density can be modeled as a thermal expansion of the fluid.

Input File Usage: *FLUID CAVITY, BEHAVIOR=*behavior_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**

Defining the reference fluid density

The reference fluid density, ρ_R, is specified at zero pressure and the initial temperature, θ_I:

$$\rho_R = \rho(0, \theta_I).$$

Input File Usage: *FLUID DENSITY

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid density**: *density*

Defining the fluid bulk modulus for compressibility

The compressibility is described by the bulk modulus of the fluid:

$$p = -K\Big(\frac{V(p,\theta) - V_0(\theta)}{V_0(\theta_I)}\Big) = -K\rho_R(\rho^{-1}(p,\theta) - \rho_0^{-1}(\theta)),$$

where

- p is the current pressure,
- θ is the current temperature,
- K is the fluid bulk modulus,
- $V(p,\theta)$ is the current fluid volume,
- $\rho(p,\theta)$ is the density at current pressure and temperature,

$V_0(\theta)$ is the fluid volume at zero pressure and current temperature,
$V_0(\theta_I)$ is the fluid volume at zero pressure and initial temperature, and
$\rho_0(\theta)$ is the density at zero pressure and current temperature.

It is assumed that the bulk modulus is independent of the change in fluid density. However, the bulk modulus can be specified as a function of temperature or predefined field variables.

Input File Usage: *FLUID BULK MODULUS

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid Bulk Modulus** tabbed page: toggle on **Specify fluid bulk modulus**, and enter the modulus value in the table

Use the following options to include temperature and field variable dependence:

Toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Defining the fluid expansion

The thermal expansion coefficients are interpreted as total expansion coefficients from a reference temperature, which can be specified as a function of temperature or predefined field variables. The change in fluid volume due to thermal expansion is determined as follows:

$$V_0(\theta) = V_0(\theta_I)[1 + 3\alpha(\theta)(\theta - \theta_0) - 3\alpha(\theta_I)(\theta_I - \theta_0)],$$

where θ_0 is the reference temperature for the coefficient of thermal expansion and $\alpha(\theta)$ is the mean (secant) coefficient of thermal expansion.

If the coefficient of thermal expansion is not a function of temperature or field variables, the value of θ_0 is not needed.

Thermal expansion can also be expressed in terms of the fluid density:

$$\rho_0(\theta) = \rho_R/[1 + 3\alpha(\theta)(\theta - \theta_0) - 3\alpha(\theta_I)(\theta_I - \theta_0)].$$

Input File Usage: *FLUID EXPANSION, ZERO=θ_0

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Hydraulic**: **Fluid Expansion** tabbed page: toggle on **Specify fluid thermal expansion coefficients**, and enter the mean coefficient of thermal expansion in the table

Use the following options to include temperature and field variable dependence:

Toggle on **Use temperature-dependent data**, **Reference temperature**: θ_0, **Number of field variables**: *n*

Pneumatic fluids

Compressible or pneumatic fluids are modeled as an ideal gas (see "Equation of state," Section 25.2.1). The equation of state for an ideal gas (or the ideal gas law) is given as

$$\tilde{p} = \rho R \left(\theta - \theta^Z \right),$$

where the absolute (or total) pressure $\tilde{p}$ is defined as

$$\tilde{p} = p + p_A,$$

and p_A is the ambient pressure, $\boldsymbol{p}$ is the gauge pressure, $\boldsymbol{R}$ is the gas constant, θ is the current temperature, and θ^Z is absolute zero on the temperature scale being used. The gas constant, $\boldsymbol{R}$, can also be determined from the universal gas constant, $\tilde{R}$, and the molecular weight, MW, as follows:

$$R = \frac{\tilde{R}}{MW}.$$

Conservation of mass gives the change of mass in the fluid cavity as

$$\dot{m} = \dot{m}_{in} - \dot{m}_{out},$$

where $\boldsymbol{m}$ is the mass of the fluid, $\dot{m}_{in}$ is the mass flow rate into the fluid cavity, and $\dot{m}_{out}$ is the mass flow rate out of the fluid cavity.

Defining the molecular weight

You must specify the value of the molecular weight of the ideal gas, MW.

Input File Usage: *MOLECULAR WEIGHT
MW

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**, **Ideal gas molecular weight**: MW

Specifying the value of the universal gas constant

You can specify the value of the universal gas constant, $\tilde{R}$.

Input File Usage: *PHYSICAL CONSTANTS, UNIVERSAL GAS CONSTANT=$\tilde{R}$

Abaqus/CAE Usage: All modules: **Model→Edit attributes→*model name***: **Physical Constants**: toggle on **Universal gas constant**: $\tilde{R}$

Specifying the value of absolute zero

You can specify the value of absolute zero temperature, θ^Z.

Input File Usage: *PHYSICAL CONSTANTS, ABSOLUTE ZERO=θ^Z

Abaqus/CAE Usage: All modules: **Model→Edit attributes→*model name***: **Physical Constants**: toggle on **Absolute zero temperature**: θ^Z

Adiabatic process

By default, the fluid temperature is defined by the predefined temperature field at the cavity reference node. However, for rapid events the fluid temperature in Abaqus/Explicit can be determined from the conservation of energy assumed in an adiabatic process. With this assumption, no heat is added or removed from the cavity except by transport through fluid exchange definitions or inflators. An adiabatic process is typically well suited for modeling the deployment of an airbag. During deployment, the gas jets out of the inflator at high pressure and cools as it expands at atmospheric pressure. The expansion is so quick that no significant amount of heat can diffuse out of the cavity.

The energy equation can be obtained from the first law of thermodynamics. By neglecting the kinetic and potential energy, the energy equation for a fluid cavity is given by

$$\frac{d(mE)}{dt} = \dot{m}_{in} H_{in} - \dot{m}_{out} H_{out} - \dot{W} - \dot{Q},$$

where the work done by the fluid cavity expansion is given as

$$\dot{W} = p\dot{V}$$

and $\dot{Q}$ is the heat energy flow rate due to the heat transfer through the surface of the fluid cavity. A positive value for $\dot{Q}$ will generate the heat energy flow out of the primary fluid cavity. The specific energy is given by

$$E = E_I + \int_{\theta_I}^{\theta} c_v(T) dT,$$

where E_I is the initial specific energy at the initial temperature θ_I, c_v is the specific heat at constant volume (or the constant volume heat capacity), which depends only upon temperature for an ideal gas, H is the specific enthalpy, and V is the volume occupied by the gas. By definition, the specific enthalpy is

$$H = H_I + \int_{\theta_I}^{\theta} c_p(T) dT,$$

where H_I is the initial specific enthalpy at the initial (or reference) temperature θ_I and c_p is the specific heat at constant pressure, which depends only upon temperature for an ideal gas. The pressure, temperature, and density of the gas are obtained by solving the ideal gas law, the energy balance, and mass conservation.

Adiabatic behavior will always be used for the fluid cavity if an adiabatic or coupled procedure is used.

Input File Usage: *FLUID CAVITY, ADIABATIC

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid cavity**: **Property definition**: **Pneumatic**, toggle on **Use adiabatic behavior**

Defining the heat capacity at constant pressure

You must define the heat capacity at constant pressure for the ideal gas. It can be defined either in polynomial or tabular form. The polynomial form is based on the Shomate equation according to the National Institute of Standards and Technology. The constant pressure molar heat capacity can be expressed as

$$\tilde{c}_p = \tilde{a} + \tilde{b}(\theta - \theta^Z) + \tilde{c}(\theta - \theta^Z)^2 + \tilde{d}(\theta - \theta^Z)^3 + \frac{\tilde{e}}{(\theta - \theta^Z)^2},$$

where the coefficients $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, and $\tilde{e}$ are gas constants. These gas constants together with molecular weight are listed in Table 11.5.2–1 for some gases that are often used in airbag simulations. The constant pressure heat capacity can then be obtained by

$$c_p = \frac{\tilde{c}_p}{MW}.$$

The constant volume heat capacity, c_v, can be determined by

$$c_v = c_p - R.$$

Table 11.5.2–1 Properties of some commonly used gases (SI units).

Gas	MW	$\tilde{a}$	$\tilde{b}$ (× 10^{-3})	$\tilde{c}$ (× 10^{-6})	$\tilde{d}$ (× 10^{-9})	$\tilde{e}$ (× 10^{6})	θ (kelvin)
Air	0.0289	28.110	1.967	4.802	−1.966	0.0	273–1800
Nitrogen	0.028	26.092	8.218	–1.976	0.1592	0.0444	298–6000
Oxygen	0.032	29.659	6.137	–1.186	0.0957	–0.219	298–6000
Hydrogen	0.00202	33.066	−11.36	11.432	–2.772	–0.158	273–1000
Carbon monoxide	0.028	25.567	6.096	4.054	−2.671	0.131	298–1300
Carbon dioxide	0.044	24.997	55.186	−33.691	7.948	–0.136	298–1200
Water vapor	0.0180	32.240	1.923	0.105	−3.595	0.0	273–1800

You can use the polynomial form for specifying the heat capacity at constant pressure, in which case you enter the coefficients $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, and $\tilde{e}$. Alternatively, you can define a table of constant pressure heat capacity versus temperature and any predefined field variables.

Input File Usage: Use the following option to specify the heat capacity in polynomial form:

∗CAPACITY, TYPE=POLYNOMIAL
$\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, $\tilde{e}$

Use the following option to specify the heat capacity in tabular form:

∗CAPACITY, TYPE=TABULAR, DEPENDENCIES=*n*
$\tilde{c}_p$, *temperature, field_variable_1, etc...*
...

Abaqus/CAE Usage: Use the following option to specify the heat capacity in polynomial form:

Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**, toggle on **Specify molar heat capacity**: **Polynomial**, **Polynomial Coefficients**: $\tilde{a}$, $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, $\tilde{e}$

Use the following option to specify the heat capacity in tabular form:

Interaction module: **Create Interaction Property**: **Fluid cavity**: **Definition**: **Pneumatic**: toggle on **Specify molar heat capacity**: **Tabular**: enter the molar heat capacity

Use the following options to include temperature and field variable dependence in the table:

Toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

A mixture of ideal gases

Abaqus/Explicit can model a mixture of ideal gases in the fluid cavity. For ideal gas mixtures the Amagat-Leduc rule of partial volumes is used to obtain an estimate of the molar-averaged thermal properties (Van Wylen and Sonntag, 1985). Let each species have constant pressure and volume heat capacities, c_{p_i} and c_{v_i}; molecular weight, MW_i; and mass fraction, f_i. The constant pressure and volume heat capacities for the mixed gas are then given by

$$c_p = \sum_i f_i c_{p_i},$$

$$c_v = \sum_i f_i c_{v_i},$$

and the molecular weight is given by

$$MW = 1/\sum_i \frac{f_i}{MW_i}.$$

The specific energy and enthalpy for the mixed gas are then given by

$$E = \sum_i f_i E_i,$$

$$H = \sum_i f_i H_i.$$

The energy flow entering the fluid cavity is given by

$$\dot{m}_{in} H_{in} = \sum_i \dot{m}_{in_i} H_{in_i},$$

and the energy flow out of the fluid cavity is given by

$$\dot{m}_{out} H_{out} = \sum_i \dot{m}_{out_i} H_{out_i}.$$

Using the properties of a mixture of ideal gases as given above, the pressure and temperature can be obtained from the ideal gas law and the energy equation.

Averaged properties for multiple fluid cavities

If the output of the state of the fluid inside the cavity is requested for a node set that contains more than one fluid cavity, the averaged properties of the multiple fluid cavities will also be output automatically. The average pressure is calculated by volume weighting cavity pressure contributions. The average temperature for an adiabatic ideal gas is obtained by mass weighting cavity temperature contributions. Let each fluid cavity have pressure p_k, temperature θ_k, volume V_k, gas constant R_k, and mass m_k. The average pressure of the fluid cavity cluster is then defined as

$$p_{avg} = \sum_k p_k V_k / \sum_k V_k,$$

and the average temperature is

$$\theta_{avg} = \sum_k R_k \theta_k m_k / \sum_k R_k m_k.$$

Additional reference

- Van Wylen, G. J., and R. E. Sonntag, *Fundamentals of Classical Thermodynamics,* Wiley, New York, 1985.

11.5.3 FLUID EXCHANGE DEFINITION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Surface-based fluid cavities: overview," Section 11.5.1
- "Fluid cavity definition," Section 11.5.2
- *FLUID EXCHANGE
- *FLUID EXCHANGE PROPERTY
- *FLUID EXCHANGE ACTIVATION
- "VUFLUIDEXCH," Section 1.2.17 of the Abaqus User Subroutines Reference Guide
- "VUFLUIDEXCHEFFAREA," Section 1.2.18 of the Abaqus User Subroutines Reference Guide
- "Defining a fluid exchange interaction," Section 15.13.12 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a fluid exchange interaction property," Section 15.14.5 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A fluid exchange definition:

- can be used to model flow between a single fluid cavity and its environment or flow between two fluid cavities;
- can be used to prescribe mass- or volume-based flux into or out of a cavity;
- can model the venting of a cavity through an exhaust orifice;
- can model flow through cavity walls such as leakage through a porous fabric;
- can be used to prescribe heat loss through a cavity surface due to heat transfer;
- can take the local material state into account;
- can account for blockage due to contacting boundary surfaces; and
- has a name that can be used to identify history output of mass flow rates out of a cavity.

Defining fluid exchange

The fluid exchange capability is very general and can be used to define flow in and out of a cavity either as a prescribed function or based on the pressure difference arising from analysis conditions. The flow behavior in Abaqus/Standard is based on mass fluid flow, and the behavior in Abaqus/Explicit can be based on mass fluid flow or heat energy flow. You must associate the fluid exchange definition with a name.

Input File Usage: *FLUID EXCHANGE, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**, **Name**: *name*

Flow between a single cavity and its environment

To define flow between a fluid cavity and its environment in Abaqus/Explicit, specify the single reference node associated with the fluid cavity. In the discussion that follows this fluid cavity is referred to as the primary cavity. When the flow is defined as a prescribed function, the flow can either be into or out of the primary cavity. If the flow is into the cavity, the properties of the material flowing in are assumed to be the instantaneous properties of the material in the cavity itself. When the flow behavior is based on analysis conditions, the mass flow can occur only out of the primary cavity but the heat energy flow can be either into or out of the primary cavity. For the case of mass flow Abaqus will use the fluid cavity pressure and the specified constant ambient pressure to calculate the pressure difference used to determine the mass flow rate. For the case of heat energy flow Abaqus/Explicit will use the fluid cavity temperature and the specified constant ambient temperature to calculate the temperature difference used to determine the heat energy flow rate.

Input File Usage: Use the following options:

∗FLUID CAVITY, NAME=*primary_cavity_name*,
REF NODE=*primary_cavity_reference_node*
∗FLUID EXCHANGE, NAME=*fluid_exchange_name*
primary_cavity_reference_node

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Definition**: **To environment**, **Fluid cavity interaction**: *name*, **Fluid exchange property**: *name*

Flow between two fluid cavities

To define flow between two fluid cavities, specify the reference nodes associated with the primary and secondary fluid cavities. When the flow is based on analysis conditions, the fluid will flow from the high pressure or upstream cavity to the low pressure or downstream cavity and the heat energy will flow from the high temperature to the low temperature.

Input File Usage: Use the following options:

∗FLUID CAVITY, NAME=*primary_cavity_name*,
REF NODE=*primary_cavity_reference_node*
∗FLUID CAVITY, NAME=*secondary_cavity_name*,
REF NODE=*secondary_cavity_reference_node*
∗FLUID EXCHANGE, NAME=*fluid_exchange_name*
primary_cavity_reference_node, secondary_cavity_reference_node

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Definition**: **Between cavities**, **Fluid cavity interaction 1**: *name*, **Fluid cavity interaction 2**: *name*, **Fluid exchange property**: *name*

Specifying the effective area in an Abaqus/Explicit analysis

The flow rate from the primary cavity for any fluid exchange property is proportional to the effective leakage area. The leakage area may represent the size of an exhaust orifice, the area of a porous fabric enclosing the cavity, or the size of a pipe between cavities.

In an Abaqus/Explicit analysis you can specify the value of the effective leakage area directly. Alternatively, you can define a surface that represents the leakage area by specifying the name of the surface on the boundary enclosing the primary fluid cavity. The effective area for fluid exchange is based on the area of the surface unless you specify the area directly or define the effective area with user subroutine **VUFLUIDEXCHEFFAREA**. If both the effective area and a surface are specified, the area of the surface is used only to determine blockage; see "Accounting for blockage due to contacting boundary surfaces," below. If neither area is specified, the effective area defaults to 1.0.

You can also define the effective leakage area with user subroutine **VUFLUIDEXCHEFFAREA** (see "VUFLUIDEXCHEFFAREA," Section 1.2.18 of the Abaqus User Subroutines Reference Guide) if leakage needs to be modeled as a function of the material state in the underlying elements of the specified surface. For example, this subroutine can be used to define the leakage area at an element level for modeling fabric permeability in uncoated airbags where the leakage can vary locally depending on the strains in the yarn directions and the angle between the fabric yarns. Only membrane elements are supported for use with **VUFLUIDEXCHEFFAREA**.

Input File Usage: Use the following option to specify the effective leakage area directly and to specify a surface that represents the leakage area:

∗FLUID EXCHANGE, EFFECTIVE AREA=*effective_area*, SURFACE=*surface_name*

Use the following option to define the effective leakage area with a user subroutine:

∗FLUID EXCHANGE, EFFECTIVE AREA=USER, SURFACE=*surface_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid exchange**: **Effective exchange area**: *effective_area*

User subroutine **VUFLUIDEXCHEFFAREA** is not supported in Abaqus/CAE.

Application of fluid cavity pressure on a fluid exchange surface

You can control how the effect of the cavity pressure on a fluid exchange surface is accounted for in Abaqus/Explicit. By default, the cavity pressure generates forces at all of the fluid exchange surface nodes, using the same method as for other portions of the fluid cavity. Optionally, the resultant force of the cavity pressure on the fluid exchange surface can be distributed among only the nodes that lie on the perimeter of the fluid exchange surface (for example, of the nodes shown on the fluid exchange surface in Figure 11.5.3–1, only the nodes at locations A and B lie on the perimeter). This option can be used to avoid local bulging of a vent surface that will cause inaccurate computation of the leakage area. Figure 11.5.3–2 shows an example of bulging when cavity pressure forces are distributed among all nodes of a vent surface.

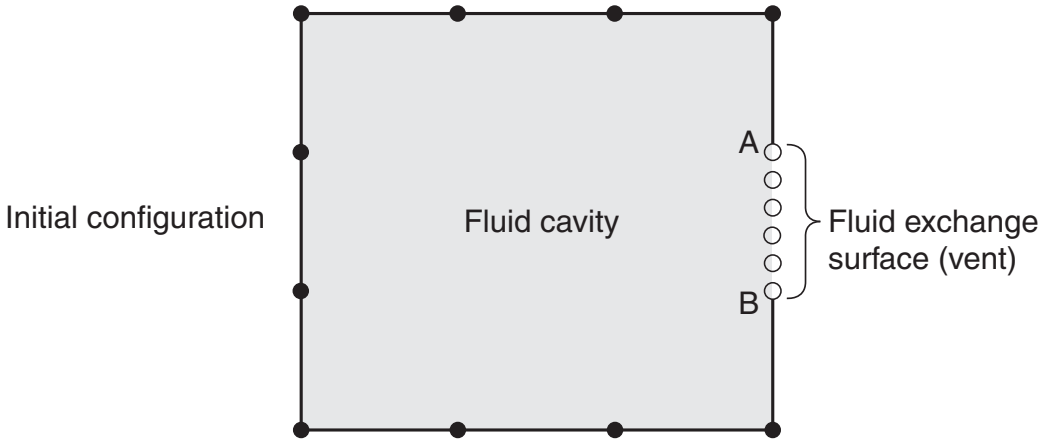


Figure 11.5.3–1 Initial configuration of a fluid exchange surface.

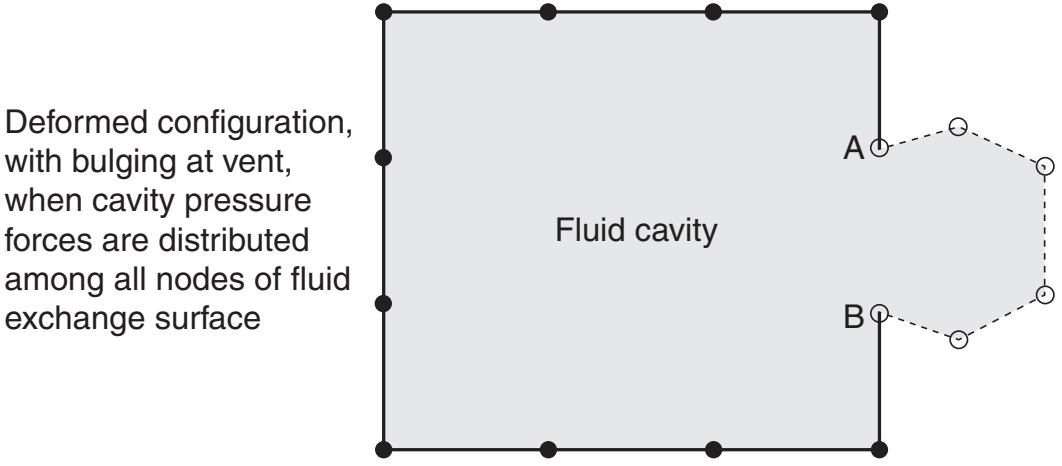


Figure 11.5.3–2 Deformed configuration of a fluid exchange surface.

Input File Usage: Use the following option (default) to indicate that the fluid pressure should generate forces on all nodes of a fluid exchange surface:

∗FLUID EXCHANGE, CAVITY PRESSURE=SURFACE, SURFACE=*surface_name*

Use the following option to indicate that the fluid pressure should generate force only on perimeter nodes of a fluid exchange:

*FLUID EXCHANGE, CAVITY PRESSURE=PERIMETER, SURFACE=*surface_name*

Abaqus/CAE Usage: You cannot change the default pressure application in Abaqus/CAE. The pressure is always applied to all of the fluid exchange surface nodes.

Defining the fluid exchange property

There are several different types of fluid exchange properties available in Abaqus to define the rate flow from a fluid cavity to the environment or between two cavities. The fluid exchange property can be as simple as prescribing the mass or volume flow rate directly. More complex leakage mechanisms such as those found on automotive airbags can be modeled by defining the mass or volume leakage rate as a function of the pressure difference, Δp; the absolute pressure, $\tilde{p}$; and the temperature, θ. The heat loss due to heat transfer through the surface of the cavity can be modeled in Abaqus/Explicit by prescribing the heat energy flow rate directly or by defining the heat energy flow rate as a function of the temperature difference, $\Delta\theta$; the absolute pressure, $\tilde{p}$; and the temperature, θ. Alternatively, in Abaqus/Explicit the mass flow rate and/or heat energy flow rate can be specified in user subroutine **VUFLUIDEXCH**.

For the purposes of evaluating the mass flow rate between two cavities, the absolute pressure and temperature are taken from the high pressure or upstream cavity. The mass flow is always in the direction from the high pressure cavity to the low pressure or downstream cavity, and the heat energy flow is always in the direction from the high temperature cavity to the low temperature cavity. The cavity absolute pressure and temperature are always used to calculate the flow between a cavity and the environment.

You must associate the fluid exchange property with a name. This name can then be used to associate a certain property with a fluid exchange definition.

Input File Usage: Use the following options:

*FLUID EXCHANGE, NAME=*fluid_exchange_name*, PROPERTY=*property_name*
*FLUID EXCHANGE PROPERTY, NAME=*property_name*

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**, **Name**: *property_name*

Specifying a mass or volume flux

Fluid flux into or out of the primary fluid cavity can be defined directly by prescribing the mass flow rate per unit area, $\dot{\tilde{m}}$. The mass flow rate is

$$\dot{m} = \dot{\tilde{m}} A,$$

where A is the effective area.

Fluid flux can also be defined by prescribing a volumetric flow rate per unit area, $\dot{\tilde{V}}$. The mass flow rate is

$$\dot{m} = \rho \dot{\bar{V}} A,$$

where ρ is the density.

A negative value for $\dot{m}$ or $\dot{\bar{V}}$ will generate flux into the primary fluid cavity. When a second fluid cavity is not defined, the state of the fluid flowing into the primary cavity is assumed to be that of the fluid already present in the primary cavity.

Input File Usage: To prescribe a flux based on mass flow rate:

∗FLUID EXCHANGE PROPERTY, TYPE=MASS FLUX

To prescribe a flux based on volumetric flow rate:

∗FLUID EXCHANGE PROPERTY, TYPE=VOLUME FLUX

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Mass flux** or **Volume flux**

Specifying the flow rate using the viscous and hydrodynamic resistance coefficients

The mass flow rate, $\dot{m}$, can be related to pressure difference by both viscous and hydrodynamic resistance coefficients such as

$$\Delta p A = C_V \dot{m} + C_H \dot{m} |\dot{m}|,$$

where Δp is the pressure difference, A is the effective area, C_V is the viscous resistance coefficient, and C_H is the hydrodynamic resistance coefficient. The resistance coefficients can be functions of the average absolute pressure, average temperature, and average of any user-defined field variables. A positive value of $\dot{m}$ corresponds to flow out of the first cavity.

Input File Usage: ∗FLUID EXCHANGE PROPERTY, TYPE=BULK VISCOSITY, DEPENDENCIES=*n*
viscous resistance coefficient (C_V), hydrodynamic resistance coefficient (C_H)

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Bulk viscosity**: **Viscous coefficient**: C_V: **Hydrodynamic coefficient**: C_H

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Specifying the flow rate through a vent or exhaust orifice

The mass flow rate through a vent or exhaust orifice that can be approximated by one-dimensional, quasi-steady, and isentropic flow is given (Bird, Stewart and Lightfoot, 2002) by

$$|\dot{m}| = CA\frac{\tilde{p}_e}{\sqrt{R(\theta-\theta^Z)}}\sqrt{\frac{2\gamma}{\gamma-1}(q^{\frac{2}{\gamma}}-q^{\frac{\gamma+1}{\gamma}})},$$

where C is the dimensionless discharge coefficient, A is the vent or exhaust orifice area, θ is the temperature in the upstream fluid cavity, θ^Z is the absolute zero on the temperature scale being used, and $\tilde{p}_e$ is the absolute pressure in the upstream fluid cavity. The pressure ratio, q, is defined as

$$q = \frac{\tilde{p}}{\tilde{p}_e},$$

where $\tilde{p}$ is the absolute pressure in the orifice. The critical pressure, p_c, at which choked or sonic flow occurs is defined as

$$p_c = \tilde{p}_e\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}},$$

where γ is the ratio of the constant pressure heat capacity, c_p, and the constant volume heat capacity, c_v:

$$\gamma = \frac{c_p}{c_v}.$$

The orifice pressure, $\tilde{p}$, is then given by

$$\tilde{p} = p_a \quad \text{if } p_a \geq p_c,$$
$$\tilde{p} = p_c \quad \text{if } p_a < p_c,$$

where p_a is equal to the ambient pressure for flow out of a single fluid cavity or the downstream cavity pressure for flow between two fluid cavities.

The value of the discharge coefficient can be a function of the absolute upstream pressure, upstream temperature, and any user-defined field variables. Fluid exchange through a vent or exhaust orifice is valid only for pneumatic fluids and is available only in Abaqus/Explicit.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=ORIFICE, DEPENDENCIES=*n*
discharge coefficient

Abaqus/CAE Usage: Fluid exchange through vents or orifices is not supported in Abaqus/CAE.

Specifying the flow rate due to fabric leakage

The mass flow rate due to leakage through fabric can be expressed as

$$|\dot{m}| = CA\sqrt{2\rho|\Delta p|},$$

where C is the dimensionless fabric leakage or discharge coefficient and A is the effective fabric leakage area.

The value of the discharge coefficient can be a function of absolute upstream pressure, upstream temperature, and any user-defined field variables.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=FABRIC LEAKAGE, DEPENDENCIES=*n*
discharge coefficient

Abaqus/CAE Usage: Defining fluid exchange due to fabric leakage is not supported in Abaqus/CAE.

Specifying a table of mass flow rate versus pressure difference

The overall mass flow rate can be calculated from a specified mass flow rate per unit area, $\dot{\tilde{m}}$, by

$$|\dot{m}| = \dot{\tilde{m}}(|\Delta p|, \tilde{p}, \theta)A,$$

where A is the effective area.

In this case you can define the mass flow rate per unit area in a table depending on the absolute value of pressure difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\tilde{m}}$ and $|\Delta p|$ must be positive and start from zero.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=MASS RATE LEAKAGE, DEPENDENCIES=*n*
0, 0
$\dot{\tilde{m}}_1$, $|\Delta p|_1$
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Mass rate leakage**: **Mass Flow Rate**: $\dot{\tilde{m}}$, **Pressure Difference**: $|\Delta p|$

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: *n*

Specifying a table of volumetric flow rate versus pressure difference

The overall mass flow rate can be calculated from a specified volumetric flow rate per unit area, $\dot{\tilde{V}}$, by

$$|\dot{m}| = \rho\dot{\tilde{V}}(|\Delta p|, \tilde{p}, \theta)A,$$

where A is the effective area and ρ is the density.

In this case you can define the volumetric flow rate per unit area in a table depending on the absolute value of pressure difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\tilde{V}}$ and $|\Delta p|$ must be positive and start from zero.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=VOLUME RATE LEAKAGE, DEPENDENCIES=*n*

$0, 0$
$\dot{\bar{V}}_1, |\Delta p|_1$
...

Abaqus/CAE Usage: Interaction module: **Create Interaction Property**: **Fluid exchange**: **Definition**: **Volume rate leakage**: **Volumetric Flow Rate**: $\dot{\bar{V}}$, **Pressure Difference**: $|\Delta p|_1$

Use the following options to include pressure, temperature, and field variable dependence:

Toggle on **Use pressure-dependent data**, toggle on **Use temperature-dependent data**, **Number of field variables**: n

Specifying a heat energy flux

In Abaqus/Explicit heat energy flux into or out of the primary fluid cavity can be defined directly by prescribing the heat energy flow rate per unit area, $\dot{\bar{Q}}$. The heat energy flow rate is

$$\dot{Q} = \dot{\bar{Q}} A,$$

where A is the effective area. A positive value for $\dot{\bar{Q}}$ generates heat flux out of the primary fluid cavity.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=ENERGY FLUX

Abaqus/CAE Usage: Defining fluid exchange by specifying the heat energy flow rate explicitly is not supported in Abaqus/CAE.

Specifying a table of heat energy flow rate versus temperature difference

The overall heat energy flow rate can be calculated from a specified heat energy flow rate per unit area, $\dot{\bar{Q}}$, by

$$\dot{Q} = \dot{\bar{Q}}(|\Delta\theta|, \tilde{p}, \theta) A,$$

where A is the effective area.

In this case in Abaqus/Explicit you can define the heat energy flow rate per unit area in a table depending on the absolute value of temperature difference and, optionally, on the average absolute pressure, average temperature, and average value of any user-defined field variables. Values for $\dot{\bar{Q}}$ and $|\Delta\theta|$ must be positive and start from zero.

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=ENERGY RATE LEAKAGE, DEPENDENCIES=n
$0, 0$
$\dot{\bar{Q}}_1, |\Delta\theta|_1$
...

Abaqus/CAE Usage: Defining fluid exchange by specifying the heat energy flow rate as a function of temperature difference and pressure is not supported in Abaqus/CAE.

Specifying mass flow rate and/or heat energy flow rate with a user subroutine

The mass flow rate, $\dot{m}$, or the overall heat energy flow rate, $\dot{Q}$, can be defined in Abaqus/Explicit using user subroutine **VUFLUIDEXCH** (see "VUFLUIDEXCHEFFAREA," Section 1.2.18 of the Abaqus User Subroutines Reference Guide).

Input File Usage: *FLUID EXCHANGE PROPERTY, TYPE=USER

Abaqus/CAE Usage: User subroutine **VUFLUIDEXCH** is not supported in Abaqus/CAE.

Activating the fluid exchange definition in Abaqus/Explicit

Fluid exchange will not occur in Abaqus/Explicit unless the fluid exchange definition is activated in an analysis step.

Input File Usage: Use the following options to activate a fluid exchange for a given analysis step:

*FLUID EXCHANGE, NAME=*fluid_exchange_name*
*FLUID EXCHANGE ACTIVATION
fluid_exchange_name

Abaqus/CAE Usage: Fluid exchange is activated automatically for Abaqus/Explicit steps in Abaqus/CAE.

Varying the magnitude of the flow

By default, the magnitude of the flow is based on the specified flow behavior. A time variation of flow magnitude during a step can be introduced by an amplitude curve. The magnitude based on the specified flow behavior is multiplied by the amplitude value to obtain the actual mass or heat energy flow rate. For example, a time variation of prescribed mass or volumetric flux can be defined.

An amplitude curve may be used to trigger an event for fluid exchange in the middle of a step. For example, an airbag may deploy at some predetermined time during a step, and it may be desirable to close off all exhaust orifices until the actual deployment. A step amplitude curve that starts at zero and steps up at deployment time could be used for this purpose.

Input File Usage: Use the following options:

*AMPLITUDE, NAME=*amplitude_name*
*FLUID EXCHANGE ACTIVATION, AMPLITUDE=*amplitude_name*

Abaqus/CAE Usage: The use of an amplitude to activate a fluid exchange is not supported in Abaqus/CAE.

Accounting for blockage due to contacting boundary surfaces

Abaqus/Explicit can account for the blockage of flow out of a cavity due to an obstruction caused by contacting surfaces. For example, flow out of an exhaust orifice may be fully or partially blocked because it is covered by another contacting surface.

Blockage can be considered for any fluid exchange property. However, a surface must be defined on the boundary of the fluid cavity to be checked for contact obstruction. Abaqus/Explicit will calculate

the area fraction of the surface not blocked by contacting surfaces and apply this fraction to the mass or energy flow rate out of the cavity. You can control the combination of surfaces that can cause blockage. Abaqus/Explicit will not consider contacting surfaces to cause blockage unless you specify that they can potentially cause blockage (see "Contact blockage," Section 37.1.4).

Input File Usage: *FLUID EXCHANGE ACTIVATION, BLOCKAGE=YES

Abaqus/CAE Usage: Accounting for blockage due to contacting boundary surfaces is not supported in Abaqus/CAE.

Limiting the flow direction

By default, flow can occur both in and out of the primary fluid cavity when a second node is included in the fluid exchange definition. In addition, heat energy flow can occur in both directions when flow is defined between a single cavity and its environment. You can limit the flow direction in Abaqus/Explicit in these cases such that fluid or heat energy flows only out of the primary fluid cavity. This method is relevant only for a fluid exchange definition based on analysis conditions and not on prescribed mass, volume, or heat energy flux.

Input File Usage: *FLUID EXCHANGE ACTIVATION, OUTFLOW ONLY

Abaqus/CAE Usage: Limiting the flow direction is not supported by Abaqus/CAE.

Activating the fluid exchange based on the change in the leakage area

The flow between cavities can be activated in Abaqus/Explicit based on a change in the area of the surface defining the effective area. You need to specify the ratio of the actual surface area to the initial effective area, which represents the threshold value for triggering the fluid exchange. The effective area used for the fluid exchange between the cavities (or between the cavity and the ambient) is the area difference between the actual area and the initial area.

Input File Usage: Use the following options:

*FLUID EXCHANGE, SURFACE=*surface_name*
*FLUID EXCHANGE ACTIVATION, DELTA LEAKAGE AREA=*surface_ratio*

Abaqus/CAE Usage: Activating the fluid exchange based on the change in the leakage area is not supported by Abaqus/CAE.

Activation in multiple steps

By default, when you modify the activation of a fluid exchange definition or activate a new fluid exchange definition, all existing fluid exchange activations in the step remain. When modifying an existing activation, all applicable data must be respecified.

Activated fluid exchange definitions remain active in subsequent steps unless deactivated. You can choose to deactivate all fluid exchange definitions in the model and optionally reactivate new ones. If you deactivate any fluid exchange definition in a step, all fluid exchange definitions must be respecified.

Input File Usage: Use the following option to modify an existing fluid exchange activation or to specify an additional fluid exchange activation (default):

*FLUID EXCHANGE ACTIVATION, OP=MOD

Use the following option to deactivate all fluid exchange definitions in the model and optionally reactivate new ones:

*FLUID EXCHANGE ACTIVATION, OP=NEW

Abaqus/CAE Usage: Fluid exchange activation is automatic for all fluid exchange interactions in all steps in Abaqus/CAE. No modifications or additions are allowed.

Specifying mass flux in Abaqus/Standard

In Abaqus/Standard the amount of fluid in a cavity can be varied in a step. An amplitude curve can be used to define the mass flow rate during the particular step.

Input File Usage: Use the following options:

*AMPLITUDE, NAME=*amplitude_name*
*FLUID FLUX, AMPLITUDE=*amplitude_name*

Use the following option to modify an existing fluid flux or to specify an additional fluid flux to a cavity (default):

*FLUID FLUX, OP=MOD

Use the following option to deactivate all fluid flux definitions in the model and optionally reactivate new ones:

*FLUID FLUX, OP=NEW

Abaqus/CAE Usage: The use of fluid flux to modify mass flow rates is not supported in Abaqus/CAE.

Additional reference

- Bird, R. B., W. E. Stewart, and E. N. Lightfoot, *Transport Phenomena,* Wiley, New York, 2002.

11.5.4 INFLATOR DEFINITION

Product: Abaqus/Explicit

References

- "Surface-based fluid cavities: overview," Section 11.5.1
- "Fluid cavity definition," Section 11.5.2
- "Fluid exchange definition," Section 11.5.3
- *FLUID INFLATOR
- *FLUID INFLATOR PROPERTY
- *FLUID INFLATOR ACTIVATION

Overview

An inflator definition:

- can be used to inflate a fluid cavity to simulate actual inflators used for airbag supplemental restraint systems;
- can inflate a fluid cavity with an ideal gas mixture different from that present in the fluid cavity;
- can be specified directly or by defining data from a tank test;
- has a name that can be used to identify history output of mass flow rates; and
- can be activated at any time during the analysis.

Defining an inflator

The inflator capability in Abaqus/Explicit is suited for modeling the flow characteristics of inflators used for airbag systems. You must associate the inflator definition with a name. You specify the reference node of the fluid cavity that the inflator will fill with gas. A single fluid cavity can have any number of inflators.

Input File Usage: *FLUID INFLATOR, NAME=*name*
fluid_cavity_reference_node

Defining the inflator property

The inflator property defines the mass flow rate and temperature as a function of inflation time either directly or by entering tank test data. It also defines the mixture of gases entering the fluid cavity. You must associate the inflator property with a name. This name can then be used to associate a certain property with an inflator definition.

Input File Usage: Use the following options:

∗FLUID INFLATOR, NAME=*fluid_inflator_name*,
PROPERTY=*property_name*
∗FLUID INFLATOR PROPERTY, NAME=*property_name*

Specifying the gas temperature and mass flow rate directly

The temperature and the mass flow rate of the gas entering the fluid cavity can be given directly as functions of inflation time. Enter a table of mass flow rate and temperature versus inflation time.

Input File Usage: ∗FLUID INFLATOR PROPERTY, TYPE=TEMPERATURE AND MASS
inflation time, inflator gas temperature, inflator mass flow rate
...

Using tank test data

The mass flow rate and the temperature of the gas entering the fluid cavity can be determined by the results of a tank test. In the test the inflator is discharged into a closed, fixed volume tank, and the time history of pressure in the tank is measured. The inflator mass flow rate can then be calculated from the pressure history using the equations of gas dynamics. For an ideal gas, conservation of energy for an adiabatic process is given by

$$\dot{m}_{tank} c_v \theta_{tank} + m_{tank} c_v \dot{\theta}_{tank} = \dot{m}_{in} c_p (\theta_{in} - \theta^Z),$$

where θ is the temperature, θ^Z is the absolute zero on the temperature scale being used, and the subscripts in and $tank$ refer to quantities in the inflator and the rigid tank, respectively. Using mass balance

$$\dot{m}_{tank} = \dot{m}_{in},$$

and the equation of state for an ideal gas with constant volume gives

$$\dot{p}_{tank} V_{tank} = R(\dot{m}_{tank} \theta_{tank} + m_{tank} \dot{\theta}_{tank}).$$

The mass flow rate can be found by combining the above equations

$$\dot{m}_{in} = \frac{\dot{p}_{tank} V_{tank}}{\gamma R(\theta_{in} - \theta^Z)},$$

where γ is the ratio of the constant pressure heat capacity, c_p, and the constant volume heat capacity, c_v:

$$\gamma = \frac{c_p}{c_v}.$$

To calculate the mass flow rate using the results of a tank test, enter a table of tank pressure and inflator temperature versus inflation time, and specify the volume of the tank.

Input File Usage: *FLUID INFLATOR PROPERTY, TYPE=TANK TEST, TANK VOLUME=V_{tank}
inflation time, inflator gas temperature, tank pressure
...

Using the dual pressure method

If both the inflator pressure, $\tilde{p}_{in}$, and tank pressure, p_{tank}, time history curves can be measured during a tank test, the inflator mass flow rate and temperature can then be calculated using the assumption of isentropic flow (Wang and Nefske, 1988). The mass flow rate through the inflator orifice can be described by

$$\dot{m}_{in} = CA\frac{\tilde{p}_{in},}{\sqrt{R(\theta_{in} - \theta^Z)}}C_{tank},$$

where C is the discharge coefficient, A is the effective area, and the coefficient C_{tank} is determined by assuming choked or sonic flow as

$$C_{tank} = (\frac{2}{\gamma+1})^{\frac{1}{\gamma-1}}\sqrt{\frac{2\gamma}{\gamma+1}}.$$

Comparing the expression for inflator mass flow rate obtained in a rigid tank with that given above, the inflator temperature is given by

$$\theta_{in} - \theta^Z = \frac{1}{R}\left(\frac{\dot{p}_{tank}V_{tank}}{\gamma CAC_{tank}\tilde{p}_{in}}\right)^2,$$

and the inflator mass flow rate is

$$\dot{m}_{in} = \frac{\gamma(CAC_{tank}\tilde{p}_{in})^2}{V_{tank}\dot{p}_{tank}}.$$

To calculate the inflator mass flow rate and temperature using the dual pressure method, enter a table of tank pressure and inflator pressure versus inflation time; and specify the volume of the tank, the effective area, and the discharge coefficient. The tank volume and effective area must be specified. The discharge coefficient has a default value of 0.4.

Input File Usage: *FLUID INFLATOR PROPERTY, TYPE=DUAL PRESSURE, TANK VOLUME=V_{tank}, EFFECTIVE AREA=A, DISCHARGE COEFFICIENT=C
inflation time, inflator pressure, tank pressure
...

Specifying the inflator pressure and mass flow rate directly

You can enter a table of the mass flow rate and inflator pressure versus inflation time and specify the effective area and discharge coefficient. The gas temperature in the inflator will be calculated by using

the assumption of isentropic flow. The effective area must be specified. The discharge coefficient has a default value of 0.4.

Input File Usage: ∗FLUID INFLATOR PROPERTY, TYPE=PRESSURE AND MASS, EFFECTIVE AREA=*A*, DISCHARGE COEFFICIENT=*C*
inflation time, inflator pressure, inflator mass flow rate
...

Specifying the gas mixture

To define the inflator gas mixture, specify the number of gas species used for the inflator, and enter a list of names of fluid behaviors and a table of the mass fraction or molar fraction of the species. The mass fraction or molar fraction of the species may be a function of inflation time. The sum of the mass fractions or molar fractions for the species should be equal to one at any given time.

Input File Usage: Use the following options to specify the gas mixture in terms of the mass fractions:

∗FLUID INFLATOR PROPERTY
∗FLUID INFLATOR MIXTURE, NUMBER SPECIES=*k*, TYPE=MASS FRACTION
fluid_behavior_name_1, fluid_behavior_name_2, etc.
inflation time, mass fraction 1, mass fraction 2, etc.
...

Use the following options to specify the gas mixture in terms of the molar fractions:

∗FLUID INFLATOR PROPERTY
∗FLUID INFLATOR MIXTURE, NUMBER SPECIES=*k*, TYPE=MOLAR FRACTION
fluid_behavior_name_1, fluid_behavior_name_2, etc.
inflation time, molar fraction 1, molar fraction 2, etc.
...

Activating the inflator definition

Inflation will not occur unless the inflation definition is activated in an analysis step.

Input File Usage: Use the following options to activate a fluid inflator for a given analysis step:

∗FLUID INFLATOR, NAME=*fluid_inflator_name*
∗FLUID INFLATOR ACTIVATION
fluid_inflator_name

Relating inflation time to analysis time

Inflator property definition consists of specifying tables of gas variables versus inflation time. In Abaqus/Explicit the inflation time, t_{in}, is related to the value of an amplitude curve $f(t)$ by

$$t_{in} = \int_0^t f(t)dt.$$

Typically the amplitude variation is a step function stepping from zero to one at the time the airbag should be deployed. This amplitude variation has the effect of offsetting the inflation time from the analysis time.

Input File Usage: Use the following options:

∗AMPLITUDE, NAME=*amplitude_name*
∗FLUID INFLATOR ACTIVATION, INFLATION TIME
AMPLITUDE=*amplitude_name*

Modifying the mass flow rate

If the mass flow rate is prescribed directly in the inflator property definition, you can modify it by specifying an amplitude definition during a step. However, if the mass flow rate is calculated by using tank test data or the dual pressure method, the amplitude definition will be ignored.

Input File Usage: Use the following options:

∗AMPLITUDE, NAME=*amplitude_name*
∗FLUID INFLATOR ACTIVATION, MASS FLOW
AMPLITUDE=*amplitude_name*

Activation in multiple steps

By default, when you modify the activation of a fluid inflator definition or activate a new fluid inflator definition, all existing fluid inflator activations in the step remain. When modifying an existing activation, all applicable parameters must be respecified.

Activated inflator definitions remain active in subsequent steps unless deactivated. You can choose to deactivate all fluid inflator definitions in the model and optionally reactivate new ones. If you deactivate any fluid inflator definition in a step, all fluid inflator definitions must be respecified.

Input File Usage: Use the following option to modify an existing fluid inflator activation or to specify an additional fluid inflator activation (default):

∗FLUID INFLATOR ACTIVATION, OP=MOD

Use the following option to deactivate all fluid inflator definitions in the model and optionally reactivate new ones:

∗FLUID INFLATOR ACTIVATION, OP=NEW

Additional reference

- Wang, J. T., and O. J. Nefske, “A New CAL3D Airbag Inflation Model,” SAE paper 880654, 1988.

11.6 Mass scaling

- “Mass scaling,” Section 11.6.1

11.6.1 MASS SCALING

Products: Abaqus/Explicit Abaqus/CAE

References

- "Explicit dynamic analysis," Section 6.3.3
- "Adjust and/or redistribute mass of an element set," Section 2.6.1
- "Output," Section 4.1.1
- *FIXED MASS SCALING
- *VARIABLE MASS SCALING
- "Configuring a dynamic, explicit procedure" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Configuring a dynamic fully coupled thermal-stress procedure using explicit integration" in "Configuring general analysis procedures," Section 14.11.1 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

Mass scaling is often used in Abaqus/Explicit for computational efficiency in quasi-static analyses and in some dynamic analyses that contain a few very small elements that control the stable time increment. Mass scaling can be used to:

- scale the mass of the entire model or scale the masses of individual elements and/or element sets;
- scale the mass on a per step basis in a multistep analysis; and
- scale the mass at the beginning of the step and/or throughout the step.

Mass scaling can be performed by:

- scaling the masses of all specified elements by a user-supplied constant factor;
- scaling the masses of all specified elements by the same value so that the minimum stable time increment for any element in the element set is equal to a user-supplied time increment;
- scaling the masses of only the elements in the element set whose element stable time increments are less than a user-supplied time increment so that the element stable time increment for these elements becomes equal to the user-supplied time increment;
- scaling the masses of all specified elements so that their element stable time increments each become equal to the user-supplied time increment; and
- scaling automatically based on mesh geometry and initial conditions for bulk metal rolling analyses.

Introduction

The explicit dynamics procedure is typically used to solve two classes of problems: transient dynamic response calculations and quasi-static simulations involving complex nonlinear effects (most commonly problems involving complex contact conditions). Because the explicit central difference method is used to integrate the equations in time (see "Explicit dynamic analysis," Section 6.3.3), the discrete mass matrix used in the equilibrium equations plays a crucial role in both computational efficiency and accuracy for both classes of problems. When used appropriately, mass scaling can often improve the computational efficiency while retaining the necessary degree of accuracy required for a particular problem class. However, the mass scaling techniques most appropriate for quasi-static simulations may be very different from those that should be used for dynamic analyses.

Quasi-static analysis

For quasi-static simulations incorporating rate-independent material behavior, the natural time scale is generally not important. To achieve an economical solution, it is often useful to reduce the time period of the analysis or to increase the mass of the model artificially ("mass scaling"). Both alternatives yield similar results for rate-independent materials, although mass scaling is the preferred means of reducing the solution time if rate dependencies are included in the model because the natural time scale is preserved.

Mass scaling for quasi-static analysis is usually performed on the entire model. However, when different parts of a model have different stiffness and mass properties, it may be useful to scale only selected parts of the model or to scale each of the parts independently. In any case, it is never necessary to reduce the mass of the model from its physical value, and it is generally not possible to increase the mass arbitrarily without degrading accuracy. A limited amount of mass scaling is usually possible for most quasi-static cases and will result in a corresponding increase in the time increment used by Abaqus/Explicit and a corresponding reduction in computational time. However, you must ensure that changes in the mass and consequent increases in the inertial forces do not alter the solution significantly.

Although mass scaling can be achieved by modifying the densities of the materials in the model, the methods described in this section offer much more flexibility, especially in multistep analyses.

See "Rolling of thick plates," Section 1.3.6 of the Abaqus Example Problems Guide, for a discussion of using mass scaling in a quasi-static analysis.

Dynamic analysis

The natural time scale is always important in dynamic analysis, and an accurate representation of the physical mass and inertia in the model is required to capture the transient response. However, many complex dynamic models contain a few very small elements, which will force Abaqus/Explicit to use a small time increment to integrate the entire model in time. These small elements are often the result of a difficult mesh generation task. By scaling the masses of these controlling elements at the beginning of the step, the stable time increment can be increased significantly, yet the effect on the overall dynamic behavior of the model may be negligible.

During an impact analysis, elements near the impact zone typically experience large amounts of deformation. The reduced characteristic lengths of these elements result in a smaller global time

increment. Scaling the mass of these elements as required throughout the simulation can significantly decrease the computation time. For cases in which the compressed elements are impacting a stationary rigid body, increases in mass for these small elements during the simulation will have very little effect on the overall dynamic response.

Mass scaling for truly dynamic events should almost always occur only for a limited number of elements and should never significantly increase the overall mass properties of the model, which would degrade the accuracy of the dynamic solution.

See "Impact of a copper rod," Section 1.3.10 of the Abaqus Benchmarks Guide, for a discussion of using mass scaling in a dynamic analysis.

Stable time increments

Throughout this section the term "element stable time increment" refers to the stable time increment of a single element. The term "element-by-element stable time increment" refers to the minimum element stable time increment within a specific element set. The term "stable time increment" refers to the stable time increment of the entire model, regardless of whether the global estimator or the element-by-element estimator is used.

Introducing mass scaling into a model

Two types of mass scaling are available in Abaqus/Explicit: fixed mass scaling and variable mass scaling. These two types of mass scaling can be applied separately, or they can be applied together to define an overall mass scaling strategy. The mass scaling can also apply globally to the entire model or, alternatively, on an element set by element set basis.

Fixed mass scaling

Fixed mass scaling is performed once at the beginning of the step for which it is specified. Two basic approaches are available for fixed mass scaling: you can define a mass scaling factor directly, or you can define a desired minimum stable time increment for which the mass scaling factors are determined by Abaqus/Explicit.

If both variable mass scaling and fixed mass scaling are specified in a step, the element original mass is scaled once at beginning of that step based on the specified fixed mass scaling. It is then further scaled at the beginning and periodically during that step based on the specified variable mass scaling.

Fixed mass scaling provides a simple means to modify the mass properties of a quasi-static model at the beginning of an analysis or to modify the masses of a few small elements in a dynamic model so that they do not control the stable time increment size. Since the scaling operation is performed only once at the beginning of the step for which the mass scaling is defined, fixed mass scaling is computationally efficient.

Input File Usage: *FIXED MASS SCALING

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**

Variable mass scaling

Variable mass scaling is used to scale the mass of elements at the beginning of a step and periodically during that step. When using this type of mass scaling, you define a desired minimum stable time increment: mass scaling factors will be calculated automatically and applied, as required, throughout the step.

If both variable mass scaling and fixed mass scaling are specified in a step, the element original mass is scaled once at beginning of that step based on the specified fixed mass scaling. It is then further scaled at the beginning and periodically during that step based on the specified variable mass scaling.

Variable mass scaling is most useful when the stiffness properties that control the stable time increment change drastically during a step. This situation can occur in both quasi-static bulk forming and dynamic simulations in which elements are highly compressed or crushed.

Input File Usage: *VARIABLE MASS SCALING

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**

Defining a scale factor directly

Defining a scale factor directly is useful for quasi-static analyses in which the kinetic energy in the model should remain small. You can define a fixed mass scaling factor that is applied to the original mass of all elements in a specified element set. The masses of the elements will be scaled at the beginning of the step and held fixed throughout the step unless further modified by variable mass scaling.

Input File Usage: *FIXED MASS SCALING, FACTOR=*scale_factor*

For example, the following option scales the masses of elements contained in element set *elset* by a factor of 10:

```
*FIXED MASS SCALING, FACTOR=10., ELSET=elset
```

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**, **Scale by factor:** *scale_factor*

Defining a desired element-by-element stable time increment

You can define a desired element-by-element stable time increment for an element set for fixed or variable mass scaling. Abaqus/Explicit will then determine the necessary mass scaling factors. There are three mutually exclusive methods available to scale the mass of the model when a desired element-by-element stable time increment is defined. Each method is described in detail later in this section.

To determine the stable time increment used during an increment, Abaqus/Explicit first determines the smallest stable time increment on an element-by-element basis. Then, a global estimation algorithm determines a stable time increment based on the highest frequency of the model. The larger of the two estimates determines the stable time increment used. In general, the stable time increment determined

by the global estimator will be greater than the stable time increment determined by the element-by-element estimator. When fixed or variable mass scaling is used with a specified element-by-element stable time increment to scale the mass of a set of elements, the element-by-element stable time increment estimate is being affected directly. If all of the elements in the model are being scaled by a single mass scaling definition, the element-by-element estimate will equal the value assigned to the element-by-element stable time increment unless the penalty method is being used to enforce contact constraints. Penalty contact can cause the element-by-element estimate to be slightly below the value assigned to the element-by-element stable time increment (see "Contact controls for general contact in Abaqus/Explicit," Section 36.4.5, and "Contact constraint enforcement methods in Abaqus/Explicit," Section 38.2.3). The actual stable time increment used may be greater than the value assigned to the element-by-element stable time increment because of the use of the global estimator. If mass scaling is performed on only a portion of the model, the elements that are not scaled may have element stable time increments that are less than the value assigned to the element-by-element stable time increment and in that case will control the element-by-element stable time increment estimate. As a result, if only portions of the model are being scaled, the time increment used will generally not equal the value assigned to the element-by-element stable time increment.

If the fixed time increment size for the explicit dynamic step is based on the initial element-by-element stability limit or is specified directly, the time increment used will be calculated according to the rules described in "Explicit dynamic analysis," Section 6.3.3.

Scaling the mass uniformly

Scaling the mass uniformly is useful for quasi-static analyses in which the kinetic energy in the model should remain small. This approach is similar to defining a scale factor directly. In both cases the masses of all the elements specified are scaled uniformly by a single factor. However, with this method the mass scaling factor is determined by Abaqus/Explicit instead of being user specified. A single mass scaling factor is applied uniformly to all the elements so that the minimum stable time increment within these elements is equal to the value assigned to the element-by-element stable time increment, *dt*.

Input File Usage: Use either of the following options:

*FIXED MASS SCALING, TYPE=UNIFORM, DT=*dt*
*VARIABLE MASS SCALING, TYPE=UNIFORM, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: Uniformly to satisfy target**

Scaling only elements with element stable time increments below the specified element-by-element stable time increment

Scaling elements with element stable time increments below a user-specified value is appropriate for both quasi-static and dynamic analyses. It is useful for increasing the element stable time increment of the most critical elements.

When the mesh at the beginning of an analysis or a step contains a few very small elements that control the stable time increment size, use fixed mass scaling to scale the masses of those elements and start the step with a desired time increment value. Increasing the mass of only these controlling elements means that the stable time increment can be increased significantly, yet the effect on the overall behavior of the model may be negligible.

For analyses in which evolving deformation creates a limited number of small elements, use variable mass scaling to scale the masses of those elements, thereby limiting the reduction in the stable time increment.

Input File Usage: Use either of the following options:

∗FIXED MASS SCALING, TYPE=BELOW MIN, DT=*dt*
∗VARIABLE MASS SCALING, TYPE=BELOW MIN, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: If below minimum target**

Scaling all elements to have equal element stable time increments

Scaling all elements such that they have the same stable time increment effectively contracts the eigenspectrum of the model; that is, it reduces the range between the lowest and highest natural frequency of the model. Because of the drastic change in mass properties, this approach is appropriate only for quasi-static analyses. It implies that some elements may have mass scaling factors that are less than one.

Input File Usage: Use either of the following options:

∗FIXED MASS SCALING, TYPE=SET EQUAL DT, DT=*dt*
∗VARIABLE MASS SCALING, TYPE=SET EQUAL DT, DT=*dt*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Scale to target time increment of:** *dt*, **Scale element mass: Nonuniformly to equal target**

Global and local mass scaling

Specifying an element set for either fixed or variable mass scaling scales the mass of a localized region of the model. Omitting an element set implies that mass scaling will be performed for all elements. A global definition can be overwritten by a local definition for a given element set by repeating the mass scaling definition with an element set specified.

Input File Usage: Use either of the following options:

∗FIXED MASS SCALING, ELSET=*elset*
∗VARIABLE MASS SCALING, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step** or **Throughout step**, **Region: Set:** *elset*

Example 1

Different mass scaling factors may be useful when materials with vastly different wave speeds or mesh refinements are present in an analysis. In this example a scale factor of 50 may be desirable for the masses of all elements in a quasi-static analysis, except for a few elements for which a mass scaling factor of 500 is used.

```
*FIXED MASS SCALING, FACTOR=50.0
*FIXED MASS SCALING, FACTOR=500.0, ELSET=elset1
```

The first fixed mass scaling definition scales the masses of all elements in the model by a factor of 50. The second fixed mass scaling definition overrides the first definition for the elements contained in element set *elset1* by scaling their masses by a factor of 500.

Example 2

An alternative method of scaling the masses of elements in *elset1* is to assign a stable time increment to them and allow Abaqus/Explicit to determine the mass scaling factors.

```
*FIXED MASS SCALING, FACTOR=50.0
*FIXED MASS SCALING, DT=.5E-6, TYPE=BELOW MIN, ELSET=elset1
```

The first fixed mass scaling definition scales the masses in the entire model by a factor of 50. The second fixed mass scaling definition overrides the first definition by scaling the masses of any elements in *elset1* whose stable time increments are less than $.5 \times 10^{-6}$.

Mass scaling at the beginning of the step

Fixed mass scaling is used to prescribe mass scaling only at the beginning of a step and always scales the original element masses. When the scale factor is defined directly, the mass is scaled by the value assigned to the scale factor. If the element-by-element stable time increment, *dt*, is specified, the mass scaling is based on this value. If both the scale factor and the element-by-element stable time increment are specified, the mass is first scaled by the value assigned to the scale factor and then possibly scaled again, depending on the value assigned to the element-by-element stable time increment and the type of fixed mass scaling chosen.

Local mass scaling can be defined for a specific element set. If no element set is specified, the fixed mass scaling definition will apply to all elements in the model. Only one fixed mass scaling definition is permitted per element set. Multiple fixed mass scaling definitions cannot contain overlapping element sets. Local mass scaling definitions will overwrite global definitions for the specified element sets.

Input File Usage: *FIXED MASS SCALING, FACTOR=*factor*, DT=*dt*, TYPE=*type*, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: At beginning of step**, **Scale by factor:** *factor*, **Scale to target time increment of:** *dt*

Example

Assume that for a quasi-static analysis a mass scaling factor of 50 is applied to all the elements in the model. Furthermore, assume that even after being scaled by a factor of 50, a few extremely small or poorly shaped elements are causing the stable time increment to be less than a desired minimum. To increase the stable time increment, the following option is used:

```
*FIXED MASS SCALING, FACTOR=50., TYPE=BELOW MIN, DT=.5E-6
```

The specified scale factor causes the masses of all the elements in the model to be scaled by a factor of 50. If any element's stable time increment is still below 0.5×10^{-6} after being scaled by a factor of 50.0, its mass will be scaled such that its stable time increment is equal to 0.5×10^{-6}.

Mass scaling throughout the step

Variable mass scaling with a specified element-by-element stable time increment is used to define mass scaling that is to be performed at the beginning and throughout the step. Either the frequency in increments or the number of intervals must be specified to define how frequently mass scaling is to be performed. In increments other than those in which mass scaling is performed, the time increment used will generally be different from the value assigned to the element-by-element stable time increment.

Local mass scaling can be defined for a specific element set. If no element set is specified, the variable mass scaling definition will apply to all elements in the model. Only one variable mass scaling definition is permitted per element set. Multiple variable mass scaling definitions cannot contain overlapping element sets. Local mass scaling definitions will overwrite global definitions for the specified element sets.

Input File Usage: *VARIABLE MASS SCALING, DT=*dt*, TYPE=*type*, ELSET=*elset*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*

Calculating the mass scaling at equally spaced increments

You can specify the number of increments between mass scaling calculations. For example, specifying a frequency of 5 will cause mass scaling to be performed at the beginning of the step and at increments 5, 10, 15, etc.

Care should be taken when choosing the value of the frequency, since performing mass scaling every few increments during an analysis may result in noticeable additional computational cost per increment.

Input File Usage: *VARIABLE MASS SCALING, TYPE=*type*, DT=*dt*, FREQUENCY=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*, **Scale: Every *n* increments**

Calculating the mass scaling at equally spaced time intervals

Alternatively, you can specify the number of equally spaced time intervals at which the mass scaling calculations are to be performed. For example, specifying 5 intervals in a step with a duration of one second will cause mass scaling to be performed at the beginning of the step and at times of .2 , .4, .6, .8, and 1.0 seconds.

Input File Usage: *VARIABLE MASS SCALING, TYPE=*type*, DT=*dt*, NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Semi-automatic mass scaling**, **Scale: Throughout step**, **Scale to target time increment of:** *dt*, **Scale: At *n* equal intervals**

Different mass scaling at the beginning and during the step

There are cases where it is desirable to include mass scaling at the beginning of a step that may be modified further throughout the step.

Input File Usage: Use both of the following options:

*FIXED MASS SCALING, FACTOR=*factor*, TYPE=*type*, DT=*dt_init*
*VARIABLE MASS SCALING, TYPE=*type*, DT=*dt_min*, FREQUENCY=*n*
or NUMBER INTERVAL=*n*

Abaqus/CAE Usage: Create both of the following mass scaling definitions:

Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**:
Semi-automatic mass scaling, **Scale: At beginning of step**
Semi-automatic mass scaling, **Scale: Throughout step**

Example

Assume that in a dynamic impact analysis, a few extremely small or poorly shaped elements exist in the mesh and consequently control the stable time increment. To prevent these elements from controlling the stable time increment, it is desirable to scale their masses at the beginning of the step. In addition, elements in a region of the mesh will develop severe distortions as a result of impact with a fixed rigid surface. Consequently, elements in the impact zone may eventually control the stable time increment.

Since the elements in the impact zone are essentially stationary against the rigid surface, selectively scaling their masses will guarantee that the overall dynamic response is not adversely affected. Mass scaling these elements by prescribing a time increment to limit the reduction in the element-by-element stable time increment may decrease run time substantially.

For example, specify fixed mass scaling for all elements in the model with stable time increments below a value of 1.0×10^{-6}. In addition, specify variable mass scaling for the elements in the impact zone (*elset1*) with stable time increments below a value of 0.5×10^{-6}. In this case all the elements in the model are checked at the beginning of the step. If any have stable time increments less than 1.0×10^{-6}, their masses are scaled (independently) such that the element-by-element stable time increment equals 1.0×10^{-6}. This scaling remains in effect throughout the step and is not further modified, except for those elements in *elset1*. The variable mass scaling definition causes the elements contained in *elset1* to be scaled throughout the step so that their stable time increments do not become less than 0.5×10^{-6}. Because only elements in *elset1* are scaled during the step, it is possible that a stable time increment less than 0.5×10^{-6} may result.

Mass scaling in a multiple step analysis

The scaled element masses at the end of one step and any variable mass scaling methods specified in that step are carried forward automatically to the subsequent step, ensuring continuity in the mass matrix at the step boundaries and continued application of the variable mass scaling methods. However, you can reset the element masses to their original values or recompute the element masses by using a new fixed mass scaling method at the beginning of the subsequent step. You can also remove the variable mass scaling methods inherited from the prior step or replace an inherited method with a new variable mass scaling method.

To reset the initial mass matrix, specify a fixed mass scaling method in the subsequent step. Similarly, specify a variable mass scaling method in the subsequent step to discontinue all of the variable mass scaling methods of the prior step. The examples below illustrate the following special cases: (a) continuous mass matrix with no further mass scaling, and (b) reverting the mass matrix to the original state with no further mass scaling.

Very large changes in element mass across the steps due to mass scaling may lead to precision problems in the mass calculations. These precision problems may give rise to erroneous or misleading results. When large changes in element masses are desired in such situations, it is recommended that fixed mass scaling be used in the new step to reset the element masses to their original values before using additional mass scaling definitions, as required, to scale the element masses to their desired values.

Continuous mass matrix with no further scaling

To define a continuous mass matrix with no further scaling, remove any variable mass scaling definitions inherited from the prior step by redefining a new variable mass scaling definition.

Input File Usage: Use the following option without any parameters in a new step:

*VARIABLE MASS SCALING

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Disable mass scaling throughout step**

Example

Assume that during the first step of a quasi-static analysis, elements will experience distortions that will cause the stable time increment to decrease dramatically. Furthermore, assume that the deformation during the second step will not be large enough to have any further effect on the stable time increment.

```
*HEADING
…
*STEP
…
*FIXED MASS SCALING, FACTOR=1.1
*VARIABLE MASS SCALING, TYPE=BELOW MIN, DT=1.E-5, FREQUENCY=10
…
*END STEP
*STEP
…
*VARIABLE MASS SCALING
…
*END STEP
```

During the first step the fixed mass scaling increases the element mass by the factor 1.1. The variable mass scaling definition scales the mass of the entire model at the beginning of the step and every tenth increment such that the element-by-element stable time increment equals at least 1×10^{-5}. The variable mass scaling definition in the second step replaces the one continued from the first step. This particular definition of variable mass scaling without any parameters in the second step also prevents any further mass scaling during the second step. The scaled mass matrix from the first step is carried over to be used during the entire second step.

Reverting the mass matrix to the original state

You can introduce a fixed mass scaling method in the subsequent step to discontinue all of the mass scaling methods of the prior step. Further, if the default specification of fixed mass scaling is used, element masses revert to their original values at the beginning of the subsequent step. Thus, specify just the default fixed mass scaling method to prevent the scaled mass of the previous step from being used in a new step. This is useful going from a quasi-static simulation step where mass scaling is appropriate to a dynamic step in which no scaling is desired.

Input File Usage: Use both of the following options without any parameters:

*FIXED MASS SCALING
*VARIABLE MASS SCALING

Abaqus/CAE Usage: Create both of the following mass scaling definitions:

Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**:
Reinitialize mass
Disable mass scaling throughout step

Example

Assume that an analysis contains a quasi-static step followed by a dynamic step. Mass scaling can be performed during the quasi-static step but turned off during the dynamic step.

```
*HEADING
*STEP
…
*FIXED MASS SCALING, FACTOR=1.1
*VARIABLE MASS SCALING, TYPE=BELOW MIN, DT=1.E-5, FREQUENCY=10
*END STEP
*STEP
*FIXED MASS SCALING
*VARIABLE MASS SCALING
*END STEP
```

During the first step the fixed mass scaling increases the element mass by the factor 1.1. The variable mass scaling definition scales the mass of the entire model at the beginning of the step and every tenth increment such that the element-by-element stable time increment equals at least 1×10^{-5}. The new fixed mass scaling definition without any parameters in the second step then reverts the mass matrix back to the original state. The new variable mass scaling definition replaces all the variable mass scaling definitions inherited from the first step. Further, since the new variable mass scaling definition has no parameters, no mass scaling is applied during the second step. Thus, the mass matrix for the second step reverts to that of the original state.

Mass contribution from external programs connected to Abaqus via co-simulation

Co-simulation can lead to mass and/or rotary inertia from external programs being added to the Abaqus model during a step. However, that contribution along with other quantities imported from the external program must be removed once the co-simulation step is completed. If co-simulation is expected to add mass and/or rotary inertia to the Abaqus model, Abaqus automatically reverts the mass matrix back to the original state once such a co-simulation step is completed. You need to respecify any mass scaling that must be continued beyond the co-simulation step.

When mass scaling is or is not used

The following entities are not affected by mass scaling:

- Thermal solution response in a fully coupled thermal-stress analysis
- Gravity loads, viscous pressure loads
- Adiabatic heat calculations
- Equation of state materials
- Fluid and fluid link elements
- Surface-based fluid cavities
- Spring and dashpot elements

Densities associated with any of the relevant items in this list will remain unscaled. Mass, rotary inertia, infinite, and rigid elements can be scaled. However, because none of the elements has an associated stable time increment, they can be scaled only using either a user-specified scale factor or an element-

by-element stable time increment applied uniformly. If the element-by-element stable time increment is specified, at least one element with a stable time increment must be included in the mass scaling definition. Rotary inertia in shell, beam, and pipe elements is based on the scaled mass.

The mass of infinite elements can be scaled; however, the infinite elements will not act as quiet boundaries unless the densities of each adjacent deformable element are scaled by the same factor. The mass of both elements will be scaled by the same factor if they are both included in the same fixed or variable mass scaling definition.

Automatic mass scaling for analysis of bulk metal rolling

Bulk metal rolling is generally considered a quasi-static process, but the process is often modeled with Abaqus/Explicit because of its ability to handle the contact problem well. To achieve an economical solution with Abaqus/Explicit, it is often useful to increase the mass of the product artificially. However, the mass scaling factor must be chosen such that the changes in the mass and the corresponding changes in the inertial forces do not alter the solutions significantly. Choosing too high a scaling factor will not produce quasi-static results. Choosing too low a scaling factor, while conservative, will result in long run times. Rolling variable mass scaling can be used to make the choice of the optimal scaling factor automatic for this process.

The automatic strategy is based on the semi-automatic method of scaling all elements to have equal element stable time increments. The method is made automatic by determining the appropriate value for the target stable time increment from several parameters of the rolling process. The value used for the target stable time increment, Δt, is based on the average element length in the rolling direction, L_e; the feed rate, V; and the number of nodes in the cross-section of the product, n. The feed rate is defined as the average velocity of the product in the rolling direction during steady-state conditions. The value of Δt is adjusted during the analysis to account for the actual value of the feed rate. You must specify estimated values for the average velocity, the average element length in the rolling direction, and the number of nodes in the cross-section of the product.

The mass of any element will never drop below its original mass. This is different from the method of scaling all elements to have equal element stable time increments. Imposing this restriction means that rolling problems that have significant inertial effects will not have their mass adjusted automatically when they are analyzed as quasi-static.

To achieve a good result, it is recommended that:

- the product be meshed by extruding a two-dimensional cross-section of the product;
- the average element length in the rolling direction not vary significantly along the length of the product;
- the product have an initial velocity in the rolling direction approximately equal to the steady-state feed rate;
- the element size in the cross-section be equal to or less than the size in the rolling direction; and
- no other mass scaling be used on elements scaled with rolling automatic variable mass scaling.

Input File Usage: ∗VARIABLE MASS SCALING, ELSET=*elset1*, FREQUENCY=*n*,
TYPE=ROLLING, FEED RATE=*V*, EXTRUDED LENGTH=L_e,
CROSS SECTION NODES=*n*

Abaqus/CAE Usage: Step module: **Create Step**: **General**, **Dynamic, Explicit** or **Dynamic, Temp-disp, Explicit**: **Mass scaling**: **Use scaling definitions below**: **Create**: **Automatic mass scaling**, **Feed rate:** V, **Extruded element length:** L_e, **Nodes in cross section:** n

Output

Output variable EMSF provides the element mass scaling factor. Abaqus/CAE can be used to obtain contour and history plots of EMSF. Output variable DMASS provides the total percent change in mass of the model as a result of mass scaling and is available for history plotting in Abaqus/CAE. Output variable DMASS is not available on an element set basis.

Output variable EDT provides the element stable time increment. The element stable time increment includes the effect of mass scaling. Abaqus/CAE can be used to obtain history plots of EDT.

11.7 Selective subcycling

- “Selective subcycling,” Section 11.7.1

11.7.1 SELECTIVE SUBCYCLING

Product: Abaqus/Explicit

References

- "Explicit dynamic analysis," Section 6.3.3
- "Fully coupled thermal-stress analysis," Section 6.5.3
- *SUBCYCLING

Overview

Selective subcycling:

- allows different time increments to be used for different groups of elements;
- reduces run time for an analysis when a small region of elements in the model controls the stable time increment; and
- is invoked by defining the subcycling zones.

Introduction

The selective subcycling method in Abaqus/Explicit is based on domain decomposition. In this method subcycling zones are defined that remain unchanged during the analysis. The domain-level parallelization method ("Parallel execution in Abaqus/Explicit," Section 3.5.3) is invoked automatically when subcycling zones are defined. Each subcycling zone, as well as the non-subcycling zone, is independently decomposed into the user-specified number of parallel domains. The "master" domains are defined as the parallel domains that are derived from the non-subcycling zone and are integrated with the largest stable time increment. The remaining parallel domains derived from the subcycling zones are integrated using smaller time increments, or "subcycles."

The subcycle time increment sizes are chosen as integer divisors of the time increment used in the master parallel domains. Therefore, all parallel domains exactly reach the same time points as the master parallel domains. During subcycling, nodes that lie on the interface with the non-subcycling zone require special treatment. The velocity at the interface nodes is taken from the non-subcycling zone and is constant during subcycles. This produces an interface node displacement field that varies linearly during the subcycles.

Defining subcycling zones

Subcycling zones are defined by element sets. You can include all element types in these sets except Eulerian element types EC3D8R and EC3D8RT. However, all parallel domains must have at least one deformable element to provide the stable time increment. Abaqus/Explicit issues an error message if there is no deformable element in a parallel domain. You can define an arbitrary number of subcycling zones. However, some modeling features cannot be split between subcycling zones. Abaqus/Explicit

automatically merges subcycling zones that contain features that cannot be split. Subcycling zones are merged together when:

- the zones overlap;
- the zones share the same nodes;
- a node is in one subcycling zone, but its adjacent nodes are in a different subcycling zone;
- subcycling zones are involved in the same constraint equation, connector, or rigid body; or
- general contact is specified in the analysis.

When subcycling zones are merged, the smallest stable time increment among the merged zones is used. The constraint, connector, or rigid body is always assigned to the subcycling zone if any one of its nodes is involved in that subcycling zone. Since the domain-level parallelization method is used, all restrictions on parallel domain decomposition apply to subcycling zones. These restrictions prevent certain features from being split across master parallel domains, as well as parallel domains that contain the subcycling zones (See "Parallel execution in Abaqus/Explicit," Section 3.5.3). Analytical rigid surfaces cannot be included in the general contact domain when a subcycling zone is defined.

Efficient selective subcycling requires proper choice of subcycling zones. For each subcycling zone, the time increment size should be small compared to the non-subcycling zone, producing a large number of subcycles. The number of subcycles is the ratio of the stable time increment size in the non-subcycling zone to the stable time increment sizes in the subcycling zones. In addition to a large number of subcycles, the number of elements in a subcycling zone should generally be small compared to the total number of elements in the model for optimal performance benefit. If a majority of elements in the model are in subcycling zones, there will not be much performance benefit.

Input File Usage: Use the following option to define a subcycling zone:

∗SUBCYCLING, ELSET=*element_set_name*

Accuracy of results

The subcycling algorithm used in Abaqus/Explicit provides sufficient accuracy for most complex dynamic models. However, because of the relatively large time increment size used in the non-subcycling zone and the interpolation used on zone interface nodes, subcycling solutions can introduce a truncation error, which may slightly alter results compared with traditional solutions. This error should not affect the overall dynamic behavior of the model. Special attention should be given to the interface between the subcycling zone and non-subcycling zone when general contact (see "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1) is involved. It is not necessary to define a pair of surfaces that have the potential for contacting each other within the same zone. However, to minimize truncation errors, it is highly recommended that a single surface that has the potential for contacting others not be split across the zones.

Output and mass scaling

Output (see "Output," Section 4.1.1) and mass scaling (see "Mass scaling," Section 11.6.1) are always performed at the same time points reached by all parallel domains.

Input file template

```
*HEADING
 …
*ELSET, ELSET=ZONE1
 …
*SUBCYCLING, ELSET=ZONE1
*************************
*STEP
*DYNAMIC, EXPLICIT
```
Data line to specify the time period of the step
```
...
*END STEP
```

11.8 Steady-state detection

- “Steady-state detection,” Section 11.8.1

11.8.1 STEADY-STATE DETECTION

Product: Abaqus/Explicit

References

- "Output," Section 4.1.1
- *STEADY STATE DETECTION
- *STEADY STATE CRITERIA

Overview

Steady-state detection:

- can be used to detect the time in a quasi-static uni-directional Abaqus/Explicit simulation when a steady-state condition has been reached and then terminate the simulation;
- can be used to output quantities that are useful in tracking the progress of a uni-directional Abaqus/Explicit simulation; and
- is available only for three-dimensional analysis.

Introduction

Many types of uni-directional processes are used to transform preformed shapes into forms more suitable for further processing. The most common examples are rolling, wire drawing, and extrusion processes. Since the processes are usually carried out at low speeds, explicit dynamic procedures such as those in Abaqus/Explicit are often used to model the processes as quasi-static. The analyses usually consist of a workpiece that is formed into a desired shape by any number of rollers or other forming surfaces along a primary direction. The forming surfaces are usually modeled as rigid bodies. For rolling simulations the rigid body reference node is usually defined at the center of the roller. The mesh of the workpiece is often extruded and may be constructed of multiple layers of material. As the workpiece progresses through the forming surfaces, the shape eventually reaches a constant state. The position where the workpiece exits the final forming surface is referred to as the exit plane and is usually aligned with the rigid body reference node of the final forming surface. As soon as this constant shape is reached, the analysis is considered to have reached steady state. The force and torque on the final forming surfaces at this steady-state condition have also reached constant values or oscillate about constant values. A significant computational savings can be achieved by detecting the steady-state condition and halting the analysis either immediately or as soon as the steady-state cross-section progresses beyond the exit plane to a position referred to as the cutting plane.

Mesh requirements

The workpiece mesh is required to meet certain conditions for use with the steady-state detection capability. First, the mesh must be topologically regular in the primary direction. In other words, the

mesh should consist of multiple planes of elements with each plane being similar to its adjacent leading and trailing planes in that it contains the same number of elements and the same element topology in the cross-section. Furthermore, each element in a plane is connected to elements in leading and trailing planes that reference the same material and section properties. Therefore, meshes with multiple materials and section properties are permitted, but any row of elements in the primary direction must be of the same type and must reference the same material and section properties (see Figure 11.8.1–1).

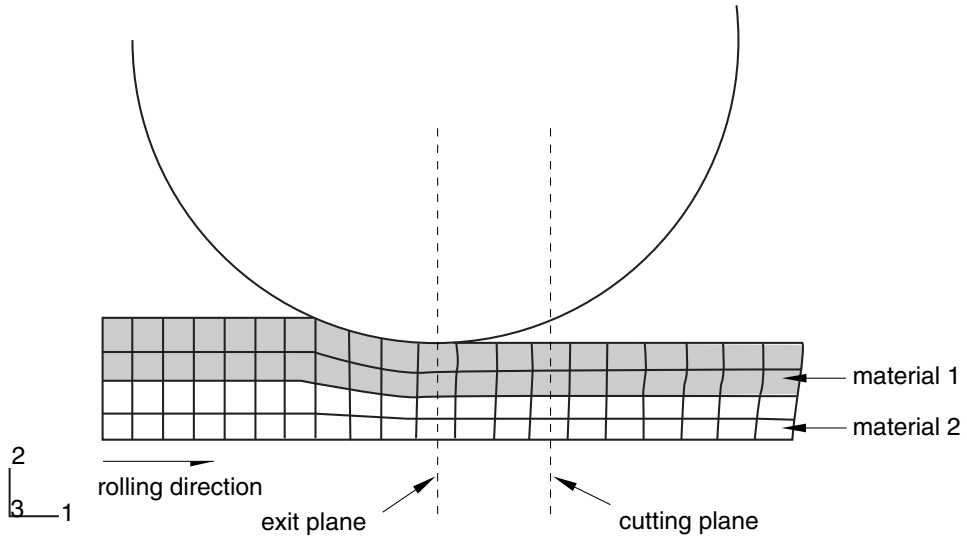


Figure 11.8.1–1 Acceptable multiple-material extruded mesh for a rolling analysis.

Steady-state detection criteria sampling

To determine if steady state has been reached, steady-state detection "norms" are calculated, which represent an averaged value of a variable of interest over the cross-section of the workpiece as material passes through a given position along the primary direction. This position is referred to as the exit plane and usually coincides with the position of the last rigid forming tool (e.g., roller) that the workpiece passes through. The normal of the exit plane is by definition coincident with the primary direction. The time intervals at which the norms are sampled vary depending on whether the rolling analysis is modeled in an Eulerian or Lagrangian manner.

Sampling in a Lagrangian analysis

In a Lagrangian-based analysis (which may include adaptive meshing employed on a Lagrangian domain) the steady-state norms are calculated as the trailing control node of each plane of elements passes the exit plane. Figure 11.8.1–2 illustrates the control node definitions.

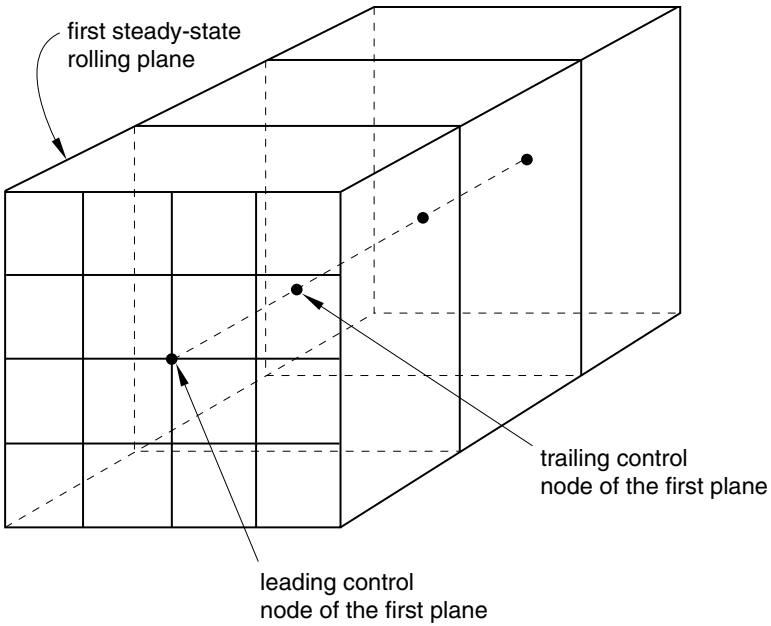


Figure 11.8.1–2 Control node positioning.

The time period of norm sampling is, therefore, based on the frequency at which the planes of elements cross the exit plane. For output purposes the values of the norms are assumed to remain constant between the times at which successive control nodes pass the exit plane.

Sampling in an Eulerian analysis

An Eulerian analysis employs a control volume approach in which material is drawn from an inflow Eulerian boundary and is pushed or pulled out through an outflow boundary. Adaptive mesh domains are defined on the workpiece, and sliding boundary regions are defined to model contact between the workpiece and forming tools such as rollers. See "ALE adaptive meshing: overview," Section 12.2.1, for details of adaptive meshing techniques. The mesh remains relatively stationary while the material moves through the exit plane. The time period between sampling is, therefore, based on the progress of the material moving through the exit plane. To determine a time period in a manner consistent with the Lagrangian case, the sampling period is determined by dividing the characteristic element length of the workpiece by the speed of the material flow. This period is roughly the time it takes for material to pass through an element of typical size.

Steady-state detection norm definitions

An individual norm is considered to have achieved steady state if its relative change in value over three consecutive planes does not exceed a tolerance. You can provide the norm tolerances when you define

the steady-state criteria, or default values of tolerances can be chosen by Abaqus/Explicit. The norms can be output by requesting their identifiers listed in the definitions below.

Equivalent plastic strain norm

The plastic strain norm of a plane of elements is defined by summing the product of the equivalent plastic strain and the element volume of each element on the plane, then dividing by the total volume of the elements on the plane. This norm provides a weighted average of the equivalent plastic strain for the plane. The identifier for the equivalent plastic strain norm is SSPEEQ.

Spread norm

The spread norm of a plane of elements is computed as the largest of the area moments of inertia of the cross-section of the plane. In determining the spread norm, the cross-section of the plane of elements is determined by projecting the element faces whose normals originally coincided with the primary direction onto the exit plane. The area moments of inertia are then determined about the centroid of the section in the directions of the original principal axes of the cross-section. The identifier for the spread norm is SSSPRD.

Force norm

The force norm is computed by averaging the magnitude of the force at the rigid body reference node of a forming tool, such as the exit roller, over the time period between sampling points. You provide the rigid body reference node and force direction. The identifier for the force norm is SSFORC.

Torque norm

The torque norm is computed by averaging the magnitude of the torque at the rigid body reference node of a forming tool, such as the exit roller, over the time period between sampling points. You provide the rigid body reference node and torque direction. The identifier for the torque norm is SSTORQ.

Requesting steady-state detection during an analysis

You must define the criteria that are used to determine if steady state has been reached. Abaqus/Explicit will halt the analysis based on the achievement of steady state.

Steady-state detection

A steady-state detection definition is used to define the elements in the workpiece, the primary direction of the workpiece, the cutting position, and the type of sampling used. The primary direction is defined by specifying the direction cosines with respect to the global Cartesian coordinate system. The cutting position is defined by specifying the global coordinates of a point lying in the cutting plane. The normal to the cutting plane is assumed to coincide with the primary direction. Once steady state has been detected, the analysis is terminated when the plane of the workpiece that was first detected to have reached steady state has progressed to the cutting plane. You can choose the sampling method used, as described below.

Requesting sampling as elements pass the exit plane for a Lagrangian analysis

You can request that all steady-state norms be calculated as each plane of elements crosses the exit plane.

Input File Usage: ∗STEADY STATE DETECTION, ELSET=*elset*,
SAMPLING=PLANE BY PLANE

Requesting sampling at uniform intervals for an Eulerian analysis

Alternatively, you can request that all steady-state norms be calculated at an interval based on the time required for material to progress the length of an average element.

Input File Usage: ∗STEADY STATE DETECTION, ELSET=*elset*, SAMPLING=UNIFORM

Steady-state criteria

Any number of steady-state criteria definitions can be specified. Only when all of the criteria specified under any one steady-state criteria definition have been satisfied will the analysis be considered to have reached steady state.

To define the criteria, you specify the norm type identifier, the norm tolerance, and the global coordinates of a point on the exit plane. For force and torque norms, you also specify the rigid body reference node of the forming tool at the exit plane and the direction cosines of the force or torque. Exit planes can be defined separately for each norm definition.

Input File Usage: Use the following options to define the criteria needed to achieve steady state:

∗STEADY STATE DETECTION, ELSET=*elset*,
SAMPLING=PLANE BY PLANE or UNIFORM
∗STEADY STATE CRITERIA
∗STEADY STATE CRITERIA
...

For example, assume that two sets of criteria are of interest and that the analysis can be terminated as soon as either is satisfied. The input might be as follows:

```
*STEADY STATE DETECTION, ELSET=sheet,
SAMPLING=PLANE BY PLANE
1.0, 0.0, 0.0,    6.0, 0.0, 0.0
*STEADY STATE CRITERIA
SSPEEQ,.002,  5.0, 0.0, 0.0
SSSPRD,.002,  5.0, 0.0, 0.0
SSFORC,.005,  5.0, 0.0, 0.0, 1000, 1.0, 0.0, 0.0
SSFORC,.005,  5.0, 0.0, 0.0, 1000, 0.0, 1.0, 0.0
SSTORQ,.005,  5.0, 0.0, 0.0, 1000, 0.0, 0.0, 1.0
*STEADY STATE CRITERIA
SSPEEQ,.001,  5.0, 0.0, 0.0
SSSPRD,.001,  5.0, 0.0, 0.0
SSFORC,.010,  5.0, 0.0, 0.0, 1000, 0.0, 1.0, 0.0
```

Materials

Steady-state detection is intended to be used with plasticity-based materials since the equivalent plastic strain norm would be zero for nonplasticity-based material models.

Procedures

One steady-state detection definition is allowed per analysis. The definition can be entered in any step and is continued through subsequent steps in an analysis. A steady-state detection definition cannot be entered in an annealing step or continued through an annealing step.

Elements

The current steady-state detection capabilities support the use of C3D8R and C3D8RT elements only.

Output

The output variables SSPEEQ*n*, SSSPRD*n*, SSFORC*n*, and SSTORQ*n* are used to output the equivalent plastic strain, spread, force, and torque norms, respectively. Abaqus/CAE can be used to obtain history plots of each of the steady-state detection norm variables. Individual norms can be output by requesting the norm number *n*, which is based on the order in which the norms are specified. Referring to the example above, if the force norm of the second steady-state criteria definition were to be requested, the output identifier would be SSFORC3. If a steady-state detection norm is requested that does not include a norm number, SSFORC for example, all norms of that type are output.

Once steady state has been detected, an element set is created automatically by Abaqus/Explicit and written to the output database consisting of the plane of elements that first satisfied the steady-state criteria. The element set created is named *SteadyStatePlane-StepN*, where *N* is the step number; and it can be viewed with Abaqus/CAE. If no output requests are made to the output database, the element set *SteadyStatePlane-StepN* is not created.

Input file template

```
*HEADING
 …
*ELSET, ELSET=WORKPIECE
************************
*STEP
*DYNAMIC, EXPLICIT
```
Data line to specify the time period of the step
```
...
*STEADY STATE DETECTION, ELSET=WORKPIECE, SAMPLING=PLANE BY PLANE
```
Data line specifying rolling direction and cutting plane position
```
*STEADY STATE CRITERIA
```
Data lines specifying steady-state detection norm criteria

```
...
*OUTPUT, HISTORY, TIME INTERVAL=1.E-6
*INCREMENTATION OUTPUT
SSPEEQ, SSSPRD, SSFORC, SSTORQ
...
*END STEP
```

12. Adaptivity Techniques

Adaptivity techniques: overview 12.1
ALE adaptive meshing 12.2
Adaptive remeshing 12.3
Analysis continuation after mesh replacement 12.4

12.1 Adaptivity techniques: overview

- “Adaptivity techniques,” Section 12.1.1

12.1.1 ADAPTIVITY TECHNIQUES

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "ALE adaptive meshing: overview," Section 12.2.1
- "Adaptive remeshing: overview," Section 12.3.1
- "Mesh-to-mesh solution mapping," Section 12.4.1
- *ADAPTIVE MESH
- "Understanding adaptive remeshing," Section 17.13 of the Abaqus/CAE User's Guide

Overview

The finite element discretization that results from suboptimal meshing of models can limit your ability to obtain adequate analysis results at a reasonable CPU cost. This section provides an overview of the adaptivity techniques available in Abaqus that help you optimize a mesh and, therefore, obtain quality solutions while controlling the cost of your analysis. The term "adaptivity" reflects the adaptive, or solution-dependent, processes that Abaqus uses to adapt your mesh to meet your analysis goals.

Selecting an adaptivity technique

Three adaptivity techniques are available in Abaqus: Arbitrary Lagrangian-Eulerian (ALE) adaptive meshing; varying topology adaptive remeshing; and mesh-to-mesh solution mapping, to enable rezoning analysis. Table 12.1.1–1 shows that the adaptivity techniques can be classified according to

- their applicability to achieving particular goals, either accuracy or control of mesh distortion;
- their impact on mesh definitions, either through smoothing a single mesh or through generating multiple dissimilar meshes; and
- when adaptivity occurs with respect to analysis steps.

Table 12.1.1–1 The characteristics of the adaptivity techniques.

	Accuracy	Distortion control	Single mesh	Multiple meshes	Adaptivity occurs
ALE adaptive meshing		√	√		Throughout a step
Adaptive remeshing	√			√	Separately from analysis steps
Mesh-to-mesh solution mapping		√		√	Between analysis steps

ALE adaptive meshing

Arbitrary Lagrangian-Eulerian (ALE) adaptive meshing provides control of mesh distortion. ALE adaptive meshing uses a single mesh definition that is gradually smoothed within analysis steps. ALE adaptive meshing is available for limited applications in Abaqus/Standard and is more generally applicable in Abaqus/Explicit. The term ALE implies a broad range of analysis approaches, from purely Lagrangian analysis, in which the node motion corresponds to material motion, to purely Eulerian analysis, in which the nodes remain fixed in space and material "flows" through the elements. Typically ALE analyses use an approach between these two extremes. The ALE feature is distinct from the Eulerian analysis capability in Abaqus/Explicit, which is described in "Eulerian analysis," Section 14.1.1.

You can use adaptive meshing to control element distortion in cases where large deformation or loss of material occurs. Figure 12.1.1–1 illustrates a case where adaptive meshing limits mesh distortion in a bulk forming simulation.

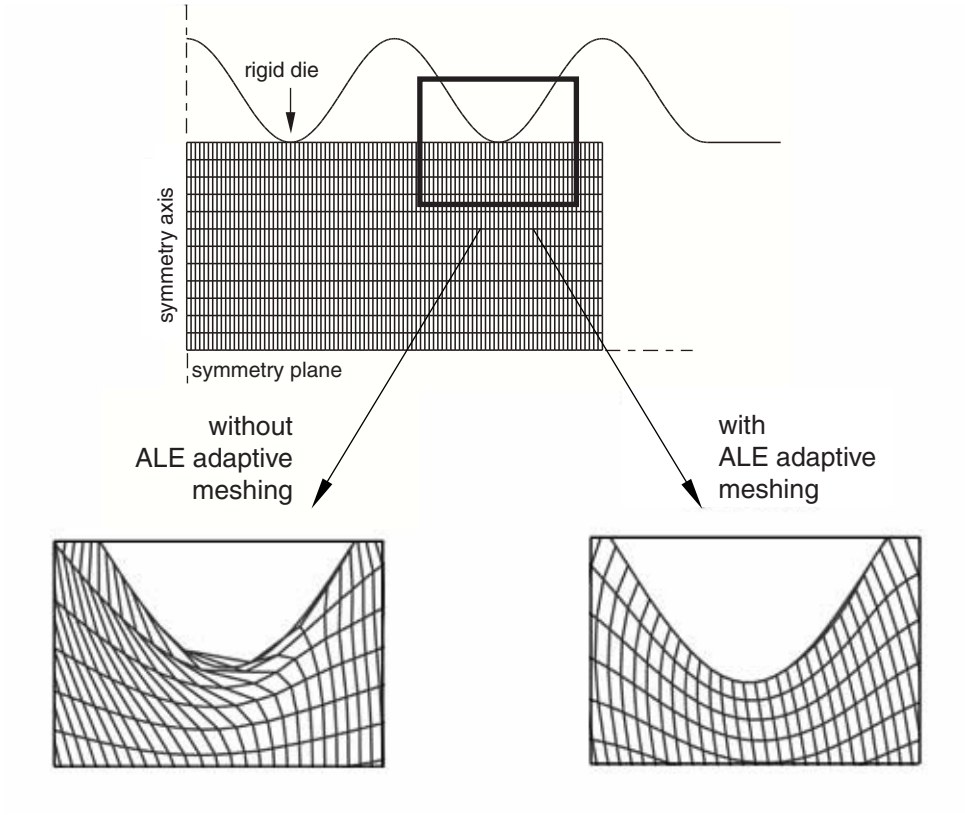


Figure 12.1.1–1 Use of ALE adaptive meshing to control element distortion.

Unlike other adaptivity techniques, adaptive meshing operates on your original mesh definition and is, therefore, useful only when a single mesh can be effective for the duration of a simulation. The mesh is adapted through smoothing of the mesh nodes. This smoothing is typically applied frequently within analysis steps. ALE adaptive meshing requires only one analysis job. See "ALE adaptive meshing: overview," Section 12.2.1, for details.

Adaptive remeshing (varying topology adaptivity)

Adaptive remeshing is typically used for accuracy control, although it can also be used for distortion control in some situations. The adaptive remeshing process involves the iterative generation of multiple dissimilar meshes to determine a single, optimized mesh that is used throughout an analysis. Adaptive remeshing is available only for Abaqus/Standard analyses submitted from Abaqus/CAE. The goal of adaptive remeshing is to obtain a solution that satisfies mesh discretization error indicator targets that you set, while minimizing the number of elements and, hence, the cost of your analysis. You can use adaptive remeshing to obtain a mesh that provides a balance between solution cost and desired accuracy. Figure 12.1.1–2 illustrates a case where adaptive remeshing improves the quality of the stress result around a fillet with targeted mesh refinement.

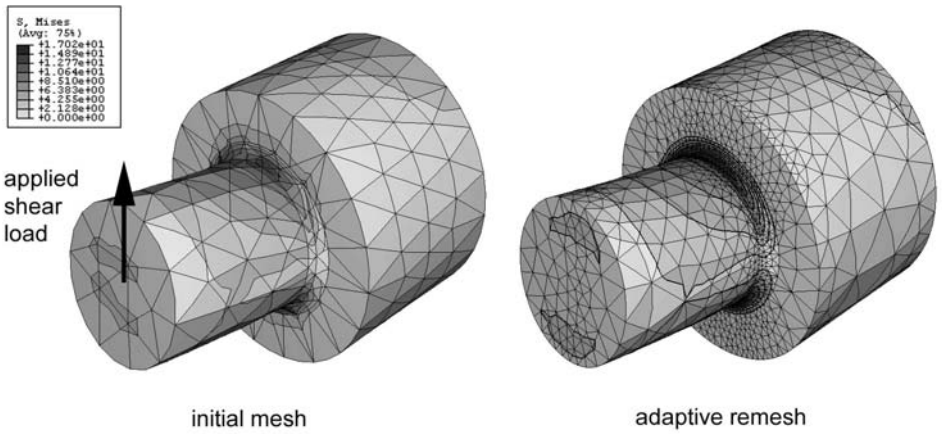


Figure 12.1.1–2 Use of adaptive remeshing to improve the quality of a stress result.

Adaptive remeshing involves an iterative process to determine a single, optimized mesh that is used through an analysis. The iterative process and the remeshing are controlled in Abaqus/CAE. Each successive analysis job covers the same simulation history time period but uses a different mesh. Once the adaptive remeshing process is complete, a single mesh and a single analysis job represent your entire analysis history. See "Adaptive remeshing: overview," Section 12.3.1, and "Understanding adaptive remeshing," Section 17.13 of the Abaqus/CAE User's Guide.

Mesh-to-mesh solution mapping

Mesh-to-mesh solution mapping is available only in Abaqus/Standard. You can use this technique to control element distortion in cases where large deformation occurs by replacing the mesh and continuing the analysis. Figure 12.1.1–3 illustrates a case where solution mapping is used in conjunction with a new mesh to overcome difficulties associated with element distortion.

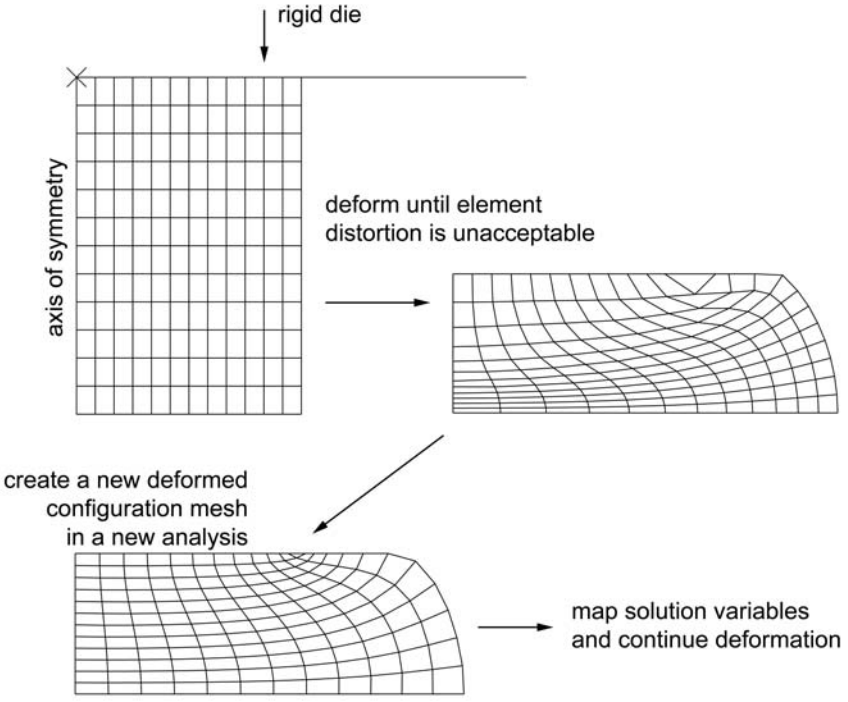


Figure 12.1.1–3 Use of mesh-to-mesh solution mapping as a component of a rezoning technique.

Mesh replacement, or rezoning, involves the creation of multiple Abaqus jobs, each of which represents the configuration of the model in distinct, sequential periods of the simulation history. You use mesh replacement when a single mesh cannot be effective for the duration of a simulation. Each mesh subsequent to the initial configuration reflects a solution-dependent deformed configuration of the model. Therefore, analyses that use mesh replacement are sequentially dependent, and Abaqus uses mesh-to-mesh solution mapping to propagate solution variables from one analysis to the next. In contrast to adaptive remeshing, each mesh replacement job represents a component of the overall analysis history—no single mesh and no single analysis job represent your entire analysis. See "Mesh-to-mesh solution mapping," Section 12.4.1, for details.

12.2 ALE adaptive meshing

- "ALE adaptive meshing: overview," Section 12.2.1
- "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2
- "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3
- "Modeling techniques for Eulerian adaptive mesh domains in Abaqus/Explicit," Section 12.2.4
- "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5
- "Defining ALE adaptive mesh domains in Abaqus/Standard," Section 12.2.6
- "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7

12.2.1 ALE ADAPTIVE MESHING: OVERVIEW

The adaptive meshing technique in Abaqus combines the features of pure Lagrangian analysis and pure Eulerian analysis. This type of adaptive meshing is often referred to as Arbitrary Lagrangian-Eulerian (ALE) analysis. The Abaqus documentation often refers to "ALE adaptive meshing" simply as "adaptive meshing."

ALE adaptive meshing is a tool that makes it possible to maintain a high-quality mesh throughout an analysis, even when large deformation or loss of material occurs, by allowing the mesh to move independently of the material. ALE adaptive meshing does not alter the topology (elements and connectivity) of the mesh, which implies some limitations on the ability of this method to maintain a high-quality mesh upon extreme deformation. Refer to "Adaptivity techniques," Section 12.1.1, for a comparison between ALE adaptive meshing and other Abaqus adaptivity methods.

ALE adaptive meshing is distinct from the pure Eulerian analysis capability in Abaqus/Explicit. The pure Eulerian capability supports multiple materials and voids within a single element, which allows effective handling of analyses involving extreme deformation (such as fluid flow). In contrast, ALE elements are always 100% full of a single material; while this formulation limits the deformation of material in the model to the deformation of the elements, it allows more precise definitions of material boundaries and more complex contact interactions. For more information on pure Eulerian analysis, see "Eulerian analysis," Section 14.1.1.

Although the adaptive meshing techniques and the user interface are similar in Abaqus/Explicit and Abaqus/Standard, the use-cases and the level of functionality are different. Adaptive meshing in Abaqus/Explicit is intended to model large-deformation problems. It does not attempt to minimize discretization errors in small-deformation analyses. Adaptive meshing in Abaqus/Standard is intended for use in acoustic domains and for modeling the effects of ablation, or wear, of material. A comparison between the adaptive remeshing functionality in Abaqus/Explicit and Abaqus/Standard is provided in this section.

Features of ALE adaptive meshing

ALE adaptive meshing:

- can often maintain a high-quality mesh under severe material deformation by allowing the mesh to move independently of the underlying material; and
- maintains a topologically similar mesh throughout the analysis (i.e., elements are not created or destroyed).

In Abaqus/Explicit ALE adaptive meshing:

- can be used to analyze Lagrangian problems (in which no material leaves the mesh) and Eulerian problems (in which material flows through the mesh);
- can be used as a continuous adaptive meshing tool for transient analysis problems undergoing large deformations (such as dynamic impact, penetration, and forging problems);
- can be used as a solution technique to model steady-state processes (such as extrusion or rolling);
- can be used as a tool to analyze the transient phase in a steady-state process; and

- can be used in explicit dynamics (including adiabatic thermal analysis) and fully coupled thermal-stress procedures.

In Abaqus/Standard ALE adaptive meshing:

- can be used to solve Lagrangian problems (in which no material leaves the mesh) and to model effects of ablation, or wear (in which material is eroded at the boundary);
- can be used to update the acoustic mesh when structural preloading causes significant geometric changes in the acoustic domain; and
- can be used in geometrically nonlinear static, steady-state transport, coupled pore fluid flow and stress, and coupled temperature-displacement procedures.

Activating ALE adaptive meshing

Adaptive meshing can be applied to an entire model or to individual parts of a model. A Lagrangian adaptive mesh domain will be created, so that the domain as a whole will follow the material originally inside it, which is the proper physical interpretation for most structural analyses. Additional options are provided for controlling the mesh. In Abaqus/Explicit analyses you can define Eulerian boundaries to allow material to flow into or out of the domain modeled.

The subsequent sections of "ALE adaptive meshing," Section 12.2, describe the various options that can be used with adaptive meshing. Although these options give you the ability to exercise detailed control over adaptive meshing, they are not necessary for many Lagrangian problems.

- To take full advantage of all the adaptive mesh features in Abaqus, it is important to understand the concepts of adaptive mesh domains, boundary regions, boundary edges, geometric features, and mesh constraints. These concepts are explained in "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2, and "Defining ALE adaptive mesh domains in Abaqus/Standard," Section 12.2.6. Instructions for applying boundary conditions, loads, and surfaces to adaptive mesh boundaries are also provided in those sections.
- "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3, and "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7, outline the methods used to move the mesh and to remap solution variables to the new mesh. These sections also present options for controlling these algorithms. Although the default methods have been chosen to work well for a wide variety of problems, you may wish to override the defaults to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing to relatively difficult or unusual applications.
- Various output and diagnostics are available for verifying the formation of adaptive mesh domains and for interpreting the results of an analysis. These options are explained in "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5.
- "Modeling techniques for Eulerian adaptive mesh domains in Abaqus/Explicit," Section 12.2.4, gives advice, in the form of examples and modeling hints, on setting up and interpreting Eulerian problems in Abaqus/Explicit using adaptive meshing.

Input File Usage: *ADAPTIVE MESH, ELSET=*elset_name*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Uses for ALE adaptive meshing

Adaptive meshing is of great value in a variety of problems. Abaqus/Explicit and Abaqus/Standard each employ adaptive meshing in ways that provide the greatest value within the particular solver.

Uses in Abaqus/Explicit

In problems where large deformation is anticipated the improved mesh quality resulting from adaptive meshing can prevent the analysis from terminating as a result of severe mesh distortion. In these situations you can use adaptive meshing to obtain faster, more accurate, and more robust solutions than with pure Lagrangian analyses.

Adaptive meshing is particularly effective for simulations of metal forming processes such as forging, extrusion, and rolling because these types of problems usually involve large amounts of nonrecoverable deformation. Because the final shape of the product can be drastically different from the original shape, a mesh that is optimal for the original product geometry can become unsuitable in later stages of the process when large material deformation leads to severe element distortion and entanglement. Element aspect ratios can also degrade in zones with high strain concentrations. Both of these factors can lead to a loss of accuracy, a reduction in the size of the stable time increment, or even termination of the problem.

Uses in Abaqus/Standard

You can use adaptive meshing to enable acoustic domain meshes to follow the large deformations of the bounding structure. In other applications you can use adaptive meshing and adaptive mesh constraints to model arbitrarily large amounts of ablation of material away from the domain.

Adaptive meshing of acoustic regions greatly extends the utility of acoustic analysis procedures. Abaqus can be used to model the response of a coupled structural-acoustic system subjected to structural preloads. By default, the structural-acoustic calculations are based on the original configuration of the acoustic domain. This approximation is adequate as long as the boundary between the fluid and structure does not experience large deformation during application of the preload. However, when the geometry of the acoustic domain changes significantly as a result of structural loading, the original acoustic configuration must be updated. An example is the interior cavity of a tire subjected to inflation, rim mounting, and footprint pressure loads.

The acoustic elements in Abaqus do not have mechanical behavior and, therefore, cannot model the deformation of the fluid when the structure undergoes large deformation. Abaqus/Standard solves the problem of computing the current configuration of the acoustic domain by periodically creating a new acoustic mesh that uses the same topology as the original mesh but with the nodal locations adjusted so that the deformation of the structural-acoustic boundary does not lead to severe distortion of the acoustic elements.

The geometric changes associated with the new acoustic mesh are then taken into account in a subsequent coupled structural-acoustic analysis. However, it is assumed that the material properties of the fluid, such as the density, do not change as a result of mesh smoothing.

Adaptive meshing can also model effects of ablation, or wear, by enabling you to define boundary mesh motions independent of the underlying material motion. An example is the wearing of a tire during its life, an effect that can significantly affect the performance of the structure.

Comparison of ALE adaptive meshing in Abaqus/Explicit and Abaqus/Standard

Adaptive meshing in Abaqus/Explicit is generally more robust and provides more features for controlling the mesh than does adaptive meshing in Abaqus/Standard.

ALE adaptive meshing in Abaqus/Explicit

Adaptive meshing in Abaqus/Explicit is designed to handle a large variety of problem classes, and employs a variety of smoothing methods, with controls that you can use to tailor the adaptivity to specific problems. The Abaqus/Explicit implementation allows you to do the following:

- to create entirely Eulerian models;
- to improve the quality of the mesh initially, before deformation begins; and
- to define tracer particles, which enable tracking and output of material-based results quantities.

ALE adaptive meshing in Abaqus/Standard

Adaptive meshing in Abaqus/Standard uses a single smoothing algorithm that works well for structural acoustic analyses and the modeling of ablation processes. The Abaqus/Standard implementation of adaptive meshing has the following limitations:

- Initial mesh sweeps cannot be used to improve the quality of the initial mesh definition.
- The method is not intended to be used in general classes of large-deformation problems, such as bulk forming.
- Diagnostics capabilities are currently limited.

Illustrative examples

To illustrate the value of adaptive meshing, simple examples of transient and steady-state forming applications follow. For simplicity, two-dimensional cases are shown. In each case Abaqus/Explicit is used in the simulation.

Axisymmetric forging

In this example a well-lubricated rigid die of sinusoidal shape moves down to deform a blank of rectangular cross-section (see Figure 12.2.1–1). The indentation depth is 80% of the original blank thickness. Material extrudes upward and outward (radially) as the blank is indented. The die is modeled with an analytical rigid surface, and the blank is modeled with axisymmetric continuum elements in a regular mesh configuration. The blank is assumed to have elastic-plastic material properties.

A pure Lagrangian analysis of this problem does not run to completion because of excessive distortion in several elements, as shown in Figure 12.2.1–2. The contact surface cannot be treated correctly because of the gross distortion of the elements at the troughs of the sinusoidal rigid surface.

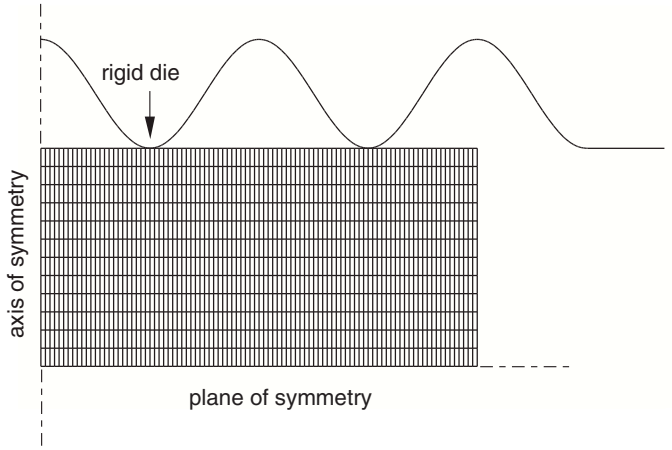


Figure 12.2.1–1 A blank and a sinusoidal die.

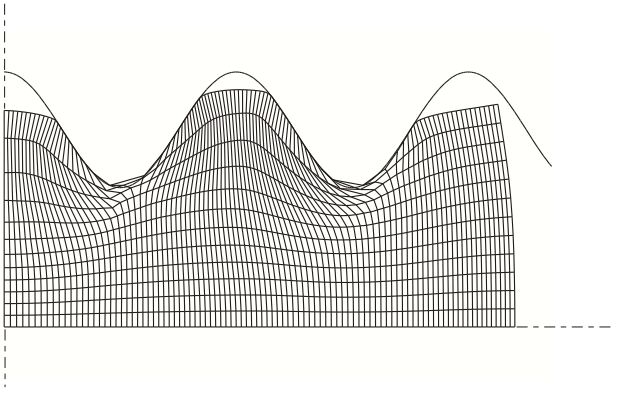


Figure 12.2.1–2 Eventually, the purely Lagrangian analysis will terminate because of excessive element distortion.

Adaptive meshing allows the problem to run to completion. A Lagrangian adaptive mesh domain is created for the entire blank. Abaqus/Explicit automatically chooses suitable defaults for adaptive meshing; hence, the adaptive mesh approach requires only two additional input lines:

```
*HEADING
 ...
*ELSET, ELSET=BLANK
**************************
*STEP
```

```
*DYNAMIC, EXPLICIT
 ...
*ADAPTIVE MESH, ELSET=BLANK
 ...
*END STEP
```

Figure 12.2.1–3 and Figure 12.2.1–4 show the deformed mesh at various stages of the forming analysis. Because the mesh refinement is maintained on the areas of the slave surface that contact the die troughs as the material flows radially, contact conditions are resolved correctly throughout the analysis.

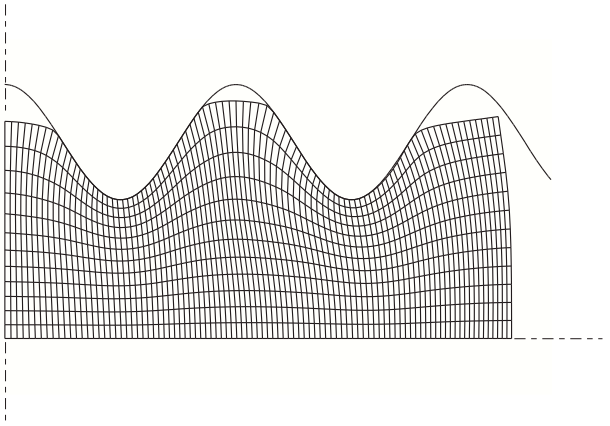


Figure 12.2.1–3 Deformed configuration at an intermediate stage of the analysis.

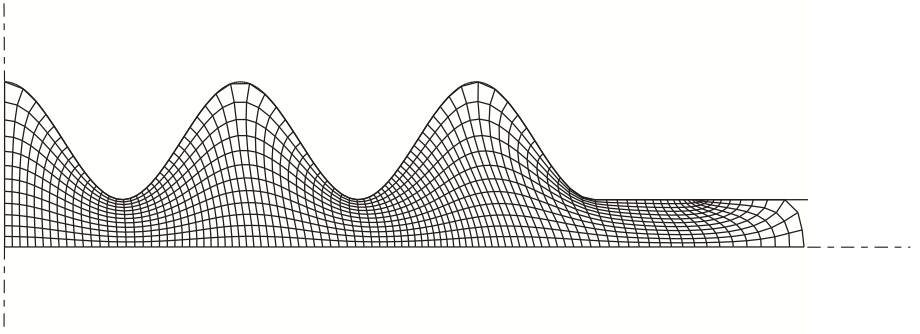


Figure 12.2.1–4 Deformed configuration upon completion of the analysis.

Steady-state rolling example

This example shows how adaptive meshing can be used in a steady-state simulation to allow the flow of material through Eulerian boundaries on the problem domain. A steel plate is passed through a symmetric roll stand to reduce its height by 50%. This simulation is run until it reaches steady-state conditions.

Figure 12.2.1–5 and Figure 12.2.1–6 show the initial and final (steady-state) configurations in a purely Lagrangian model of this problem.

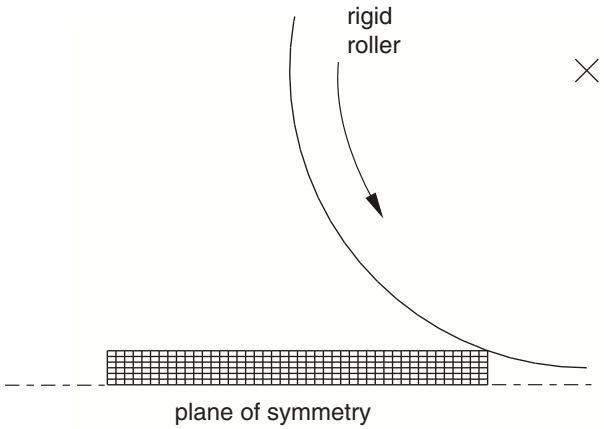


Figure 12.2.1–5 The initial configuration of the roller and the undeformed blank in the pure Lagrangian model.

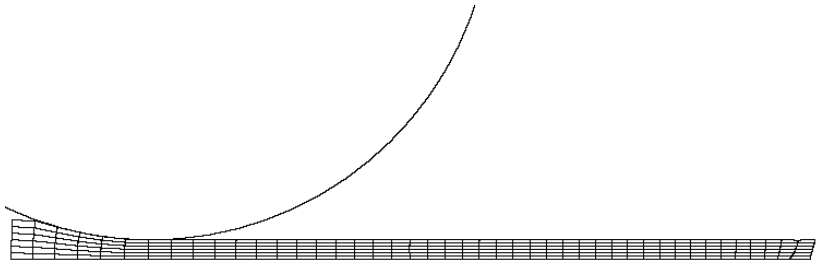


Figure 12.2.1–6 The final steady-state configuration in the pure Lagrangian model.

Figure 12.2.1–7 shows this problem modeled using an Eulerian adaptive mesh domain, where material flows through the mesh. Only the region near the roller is modeled. The exact location of the free surface does not need to be known to set up the problem: it is created in a likely location, and the final steady-state position is found as part of the solution. Although not shown, a focused mesh can be

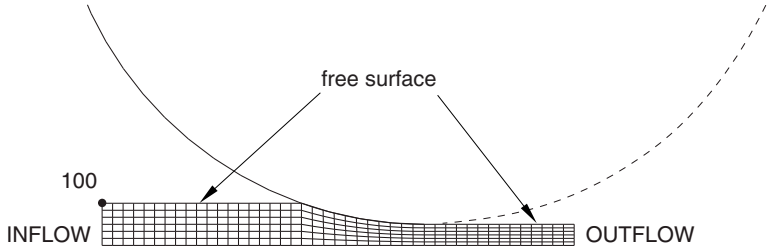


Figure 12.2.1–7 The initial Eulerian adaptive mesh domain.

used to capture steep strain gradients directly beneath the roller. The Eulerian domain reaches the same steady-state solution as obtained with the Lagrangian approach.

The Eulerian adaptive mesh domain is created by defining an inflow and an outflow boundary on the adaptive mesh domain. Adaptive mesh constraints are applied normal to these boundaries so that material will flow through the mesh (see "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2). Frictional contact between the roller and the blank pulls material through the adaptive mesh domain.

The problem is set up by making the following modifications to the input file for the pure Lagrangian analysis:

```
*HEADING
 ...
*ELSET, ELSET=BILLET
 ...
*ELSET, ELSET=INFLOW
 ...
*ELSET, ELSET=OUTFLOW
 ...
*NSET, NSET=INFLOW
 ...
*NSET, NSET=OUTFLOW
 ...
*SURFACE, NAME=INFLOW, REGION TYPE=EULERIAN
 INFLOW, S1
*SURFACE, NAME=OUTFLOW, REGION TYPE=EULERIAN
 OUTFLOW, S2
**************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
 ...
*ADAPTIVE MESH, ELSET=BILLET, CONTROLS=ADAPT
```

```
*ADAPTIVE MESH CONTROLS, NAME=ADAPT
*ADAPTIVE MESH CONSTRAINT, TYPE=DISPLACEMENT
 INFLOW, 1, 1, 0.0
 100, 2, 2, 0.0
 OUTFLOW, 1, 1, 0.0
 ...
*END STEP
```

Adaptive mesh controls were not required to solve this problem; they were included for illustrative purposes (see “ALE adaptive meshing and remapping in Abaqus/Explicit,” Section 12.2.3, for details).

12.2.2 DEFINING ALE ADAPTIVE MESH DOMAINS IN Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References

- "ALE adaptive meshing: overview," Section 12.2.1
- "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3
- *ADAPTIVE MESH
- "Understanding ALE adaptive meshing," Section 14.6 of the Abaqus/CAE User's Guide

Overview

Arbitrary Lagrangian-Eulerian (ALE) adaptive mesh domains:

- define the portions of a finite element model where mesh movement is independent of material deformation;
- can be used to analyze Lagrangian or Eulerian problems;
- can contain only first-order, reduced-integration, solid elements (4-node quadrilaterals, 3-node triangles, 8-node hexahedra, 6-node wedges, and 4-node tetrahedra);
- can be used in planar, axisymmetric, and three-dimensional geometries;
- have boundary regions where loads, boundary conditions, and surfaces can be defined; and
- are active only for geometrically nonlinear steps.

Defining an ALE adaptive mesh domain

ALE adaptive meshing is performed in adaptive mesh domains, which can be either Lagrangian or Eulerian. Within either type of adaptive mesh domain the mesh will move independently of the material. Lagrangian adaptive mesh domains are usually used to analyze transient problems with large deformations. On the boundary of a Lagrangian domain the mesh will follow the material in the direction normal to the boundary, so that the mesh covers the same material domain at all times. Eulerian adaptive mesh domains are usually used to analyze steady-state processes involving material flow. On certain user-defined boundaries of an Eulerian domain, material can flow into or out of the mesh. By default, the mesh is not fixed spatially on these boundaries; mesh constraints must be applied to prevent the mesh from moving with the material, as described in "Mesh constraints," presented later in this section. There can never be any "empty" elements; all elements in the domain must be filled completely with material at all times.

You must specify the region of the original mesh that will be subject to adaptive meshing.

Input File Usage: *ADAPTIVE MESH, ELSET=*name*

Multiple adaptive mesh domains can be defined in a step by reusing the *ADAPTIVE MESH option (for example, to prevent material from flowing

from one domain to another or to apply adaptive meshing to unconnected domains). The element sets used to create adaptive mesh domains cannot overlap.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Only one adaptive mesh domain can be defined in Abaqus/CAE for any particular step.

Modifying an ALE adaptive mesh domain

By default, all adaptive mesh domains defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh domains in effect for a given step relative to the preexisting adaptive mesh domains. At each new step the existing adaptive mesh domains can be modified and additional adaptive mesh domains can be specified (except in Abaqus/CAE, where only one adaptive mesh domain can be in effect for a given step).

Input File Usage: Use either of the following options to modify an existing adaptive mesh domain or to specify an additional adaptive mesh domain:

*ADAPTIVE MESH, ELSET=*name*
*ADAPTIVE MESH, ELSET=*name*, OP=MOD

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**

Removing an ALE adaptive mesh domain

If you choose to remove any adaptive mesh domain in a step, no adaptive mesh domains will be propagated from the previous step. Therefore, all adaptive mesh domains that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh domains and to specify new adaptive mesh domains:

*ADAPTIVE MESH, ELSET=*name*, OP=NEW

If the OP=NEW parameter is used on any *ADAPTIVE MESH option within a step, it must be used on all *ADAPTIVE MESH options in the step.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **No adaptive mesh domain for this step**

Splitting ALE adaptive mesh domains

User-defined adaptive mesh domains are examined by Abaqus/Explicit. The user-defined domain will be modeled using a single adaptive mesh if the domain:

- consists of a single element type;
- consists of a single connected region;
- consists of a single material;

- is subject to a uniform body force (including zero body force); and
- has identical section controls.

The user-defined domain will be split into multiple adaptive mesh domains, separated by boundary regions, if the domain:

- consists of multiple element types;
- spans part instances;
- consists of multiple regions (including regions that are connected by less than a single element face, only by contact conditions, or only by connectors such as MPCs);
- consists of multiple materials;
- is subject to multiple body force definitions; or
- is subject to multiple section control definitions.

In this documentation the term "adaptive mesh domain" refers to a single domain after splitting by Abaqus/Explicit. On the rare occasion that a reference is made to an adaptive mesh domain prior to the automatic splitting, it is referred to as a "user-defined adaptive mesh domain." Since adaptive mesh domains are split across element types, degenerate elements should be used for mixed domains that include both triangles and quadrilaterals (or tetrahedron and bricks). For example, when defining a mixed plane strain domain with quadrilateral and triangular elements, the CPE4R element type should be used to define both quadrilaterals and degenerated quadrilaterals. Using the CPE3 element will result in split domains, which is generally not desirable.

ALE adaptive mesh boundary regions

Each ALE adaptive mesh domain has a boundary, which can consist of one or more regions. (Regions, in this context, are surfaces in three-dimensional models or lines in two-dimensional or axisymmetric models.) A boundary region can be either Lagrangian, sliding, or Eulerian. Some boundary regions are created automatically by Abaqus/Explicit; others can be created by defining boundary conditions, loads, and surfaces. Adaptive mesh boundary regions are separated by edges in three dimensions and by corners in two dimensions. Both edges and corners are referred to as "boundary region edges" throughout this documentation.

Boundary region edges

Two types of boundary region edges can exist: Lagrangian and sliding. Lagrangian edges are always associated with a material line. Material can never flow past a Lagrangian edge, and nodes can move only along a Lagrangian edge (like beads on a string). Sliding edges are associated only with the mesh. Material can flow past a sliding edge (that is, sliding edges are free to slide over the underlying material).

Lagrangian edges can be viewed with Abaqus/CAE; see "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5.

Lagrangian boundary regions

Lagrangian boundary regions are the most common boundary regions in structural finite element analysis; therefore, with the exception of contact surfaces, they are always the default in Abaqus/Explicit. A Lagrangian boundary region has the most constraints of all the boundary region types. The mesh is constrained to move with the material in the direction normal to the surface of the boundary region and in the directions perpendicular to the boundary region edges.

Lagrangian boundary regions have Lagrangian edges: the edges follow the material. On the interior of a Lagrangian boundary region, the mesh can move independently of the material in the surface of the boundary region. Thus, a Lagrangian boundary region can be thought of as a "mesh patch" that follows the material. Nodes are free to move within and along the edges of the patch but cannot leave the patch.

Lagrangian corners

A Lagrangian corner is formed where two Lagrangian edges meet. The node at a Lagrangian corner is constrained to move with the material in all directions; it is nonadaptive.

Sliding boundary regions

A sliding boundary region is the same as a Lagrangian boundary region except that it has a sliding edge. Sliding boundary regions are created by default when you define a surface on the boundary of an adaptive mesh domain (see "Surfaces: overview," Section 2.3.1).

The mesh is constrained to move with the material in the direction normal to the boundary region, but it is completely unconstrained in the directions tangential to the boundary region. Thus, a sliding boundary region can be thought of as a "mesh patch" that moves independently of the underlying material.

Sliding boundary regions can be created by defining a surface, boundary condition, or load on the boundary of an adaptive mesh domain (as explained later in this section). Since the mesh is totally unconstrained in the directions tangential to a sliding boundary region, the location of an applied boundary condition or load may not be physically meaningful as the mesh moves over the material. Therefore, to retain the spatial meaning of an applied boundary condition or load, spatial mesh constraints (described in "Mesh constraints," presented later in this section) are usually applied tangential to sliding boundary regions.

Eulerian boundary regions

Eulerian boundary regions can be defined on the exterior of a model where it makes physical sense to let material flow across the boundary (for example, at the inlet and outlet of a steady-state extrusion or rolling problem). This flow across the boundary distinguishes Eulerian boundary regions from Lagrangian or sliding boundary regions.

Eulerian boundary regions have sliding edges and must lie completely on an exterior surface of a model. It makes no physical sense to allow material flow to originate on an interior surface. You must explicitly define Eulerian boundary regions since, by default, Abaqus/Explicit assumes that no material flows into or out of an adaptive mesh domain.

Eulerian boundary regions are created by defining a surface, a boundary condition, or a load on the boundary of an adaptive mesh domain. On Eulerian boundary regions the mesh motion usually should

be constrained in the direction normal to the material motion; therefore, the surface mesh should be fixed in space using spatial mesh constraints (described in "Mesh constraints," presented later in this section). Applying these constraints normal to an Eulerian boundary region allows material to flow into or out of the mesh, as in a fluid flow problem, while allowing adaptive meshing to occur on the surface of the boundary region to maximize mesh quality.

The material flowing into an Eulerian boundary region is assumed to have the same properties as the material that is inside the adaptive mesh domain.

Techniques for modeling Eulerian domains are presented in "Modeling techniques for Eulerian adaptive mesh domains in Abaqus/Explicit," Section 12.2.4.

Creation of boundary regions

Abaqus/Explicit will create adaptive mesh boundary regions automatically on

- the exterior of a model,
- the boundary between different adaptive mesh domains, or
- the boundary between an adaptive mesh domain and a nonadaptive domain.

By default, a boundary region on the exterior of a model will be Lagrangian, so that the boundary region follows the material, and loads, boundary conditions, etc. will retain their Lagrangian interpretation. A boundary region between different adaptive mesh domains is always Lagrangian: no material can flow through such a boundary region. An additional constraint is applied when the model contains multiple parallel domains (see "Parallel execution in Abaqus/Explicit," Section 3.5.3). In this case the boundary region is nonadaptive: no material can flow through the boundary region, and the nodes on this boundary are constrained to move exactly with the underlying material in all directions. A boundary region between an adaptive mesh domain and a nonadaptive domain is always nonadaptive. The only exception to this occurs if an Eulerian boundary region is defined on the boundary between an adaptive mesh domain and a nonadaptive domain that comprises displacement-based infinite elements. In this case the nodes on the boundary behave as in Eulerian boundary regions (see the description under "Eulerian boundary regions," presented earlier in this section), and the mesh motion at the boundary nodes can be constrained using spatial mesh constraints.

The boundary between two different materials can never "flow" through the mesh; such a physical boundary is always associated with a Lagrangian boundary region or a nonadaptive mesh boundary.

Figure 12.2.2–1 shows some boundary regions that will be created automatically by Abaqus/Explicit. In the model shown in this figure Abaqus/Explicit splits the user-defined adaptive mesh domain into two adaptive mesh domains because the original domain is composed of two different materials.

In addition to the boundary regions created automatically by Abaqus/Explicit, Lagrangian, sliding, and Eulerian boundary regions can be created by the definition of surfaces, boundary conditions, and loads, as described later in this section.

Geometric features

Many models include distinct geometric kinks that take the form of geometric edges or corners. It is usually not desirable to perform adaptive meshing across such geometric features unless they flatten.

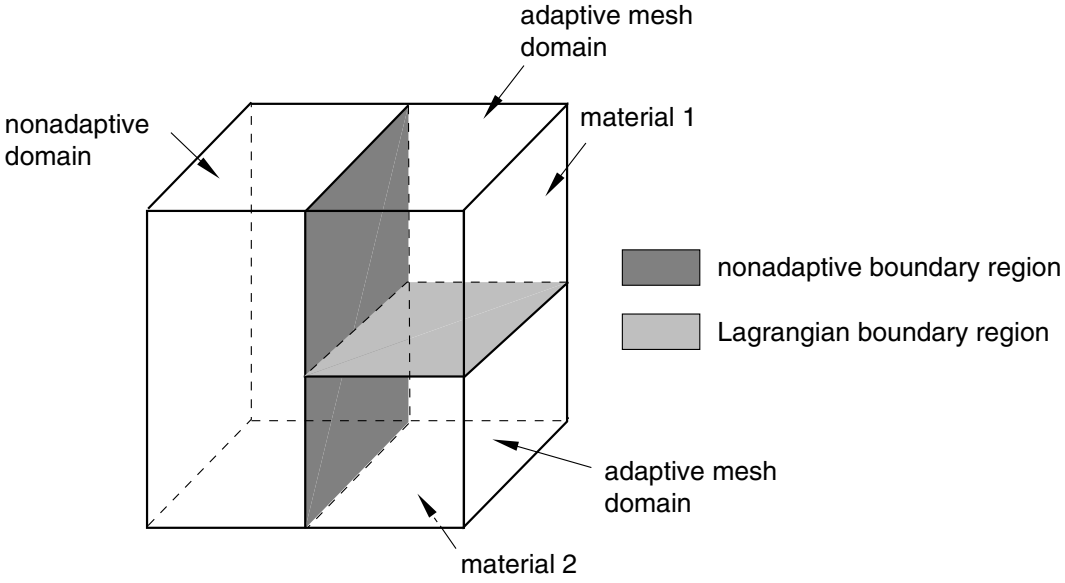


Figure 12.2.2–1 Automatic splitting of mesh domains and creation of boundary regions.

Once a geometric feature does flatten, it is usually best if the feature is deactivated so that adaptive meshing will occur across it. This is especially true when adaptive mesh domains are subject to large deformation.

The adaptive meshing algorithm in Abaqus/Explicit will respect geometric features on Lagrangian and sliding boundaries. In three dimensions geometric features consist of edges and corners (see Figure 12.2.2–2), while in two dimensions they consist of only corners. If a geometric edge coincides with the edge of a Lagrangian boundary region, the presence of the geometric feature has no effect on the treatment of the edge: material cannot flow perpendicular to a Lagrangian edge.

Geometric features are not detected or tracked on Eulerian boundary regions because they generally are not physically meaningful.

Output options are available for viewing the formation of geometric edges and corners—see "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5.

Controlling the detection of geometric edges and corners

Geometric features are identified initially as edges on boundary regions where the angle between the normals on adjacent element faces is greater than the initial geometric feature angle, θ_I ($0^\circ \leq \theta_I \leq 180^\circ$). See Figure 12.2.2–3. The default value for the initial geometric feature angle is $\theta_I = 30^\circ$.

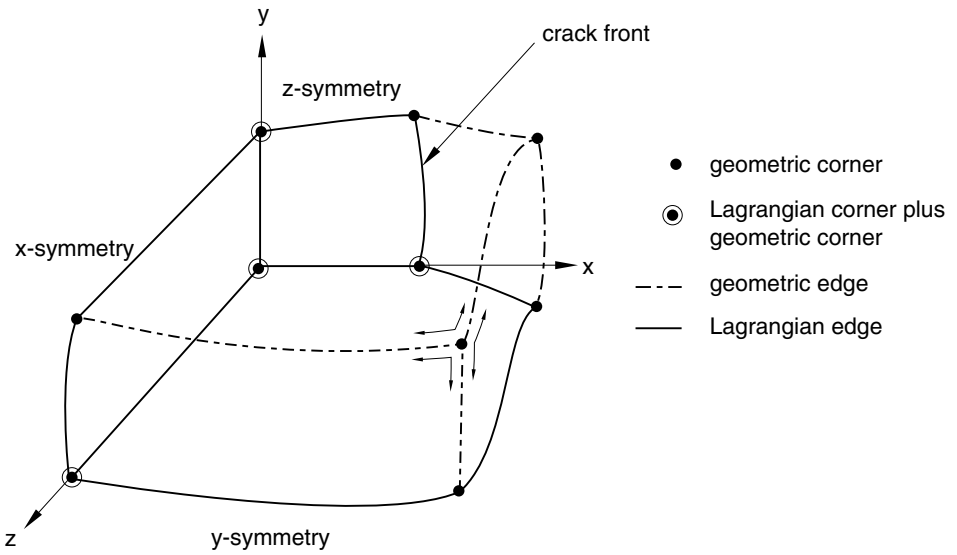
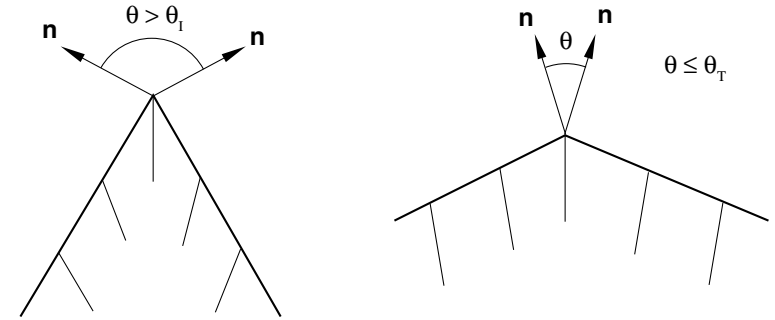


Figure 12.2.2–2 Geometric features formed on a solid block with a crack.



Initial mesh with a geometric feature: no mesh flow is permitted past the corner.

The geometric feature is deactivated during simulation.

Figure 12.2.2–3 Detection and deactivation of geometric features.

You can change the value of the angle that will be used to recognize geometric features. Setting $\theta_I = 180°$ will ensure that no geometric edges or corners are formed on the boundary of the adaptive mesh domain.

Input File Usage: ∗ADAPTIVE MESH CONTROLS, NAME=*name*,
INITIAL FEATURE ANGLE=θ_I

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Initial feature angle:** θ_I

Controlling the deactivation of geometric edges and corners

Geometric features affect only Lagrangian and sliding boundary regions, where they act as temporary Lagrangian edges. During each mesh sweep in an adaptive mesh increment, nodes along a geometric edge are positioned by applying the basic smoothing methods (see "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3). The nodes are constrained to lie along the discrete geometric edge unless the angle forming the geometric edge becomes less than the transition geometric feature angle, θ_T ($0^\circ \leq \theta_T \leq 180^\circ$). The default value for the transition feature angle is $\theta_T = 30^\circ$. If the angle across the geometric edge becomes less than θ_T, the boundary surface is considered to be flattened sufficiently for the feature to be deactivated, and the mesh is allowed to slide freely over the material (unconstrained by the deactivated geometric edge). Geometric corners are allowed to flatten in a similar fashion. This logic allows great flexibility in mesh adaptation while preserving geometric features in the model.

You can change the transition feature angle. Setting $\theta_T = 0^\circ$ will ensure that no geometric edges or corners are ever deactivated.

Input File Usage: *ADAPTIVE MESH CONTROLS, NAME=*name*,
TRANSITION FEATURE ANGLE=θ_T

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Transition feature angle:** θ_T

Mesh constraints

In most adaptive mesh problems the motion of nodes in the mesh is determined by the meshing algorithm, with constraints imposed by the domain boundary and the boundary region edges. However, there are cases when you must explicitly define the motion of the nodes. As explained earlier, Eulerian and sliding boundary regions generally require mesh constraints for the regions to be physically meaningful. In some problems you may wish to keep certain nodes fixed, to move nodes in a particular direction, or to force certain nodes to move with the material. In other problems you may desire a node or particular set of nodes to follow the material motion. Adaptive mesh constraints allow full control over the mesh movement and act independently of any boundary conditions or loads applied to the underlying material.

Applying spatial mesh constraints

Use a spatial mesh constraint (the default) to prescribe spatial mesh motion that is independent of the material motion. You specify the nodes to which the constraint is applied, the directions of the prescribed motion, and the amplitude of the prescribed motion. You can prescribe either a displacement or a velocity for the spatial mesh motion.

Input File Usage: Use the following option to define the mesh constraints explicitly:

*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=SPATIAL,
TYPE=DISPLACEMENT or VELOCITY

Abaqus/CAE Usage: To define the mesh constraints explicitly:

Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**

Rules for applying spatial mesh constraints

Spatial mesh constraints can be applied without restriction to nodes on an Eulerian boundary region or in the interior of an adaptive mesh domain.

In both two and three dimensions nodes at Lagrangian and active geometric corners are fully constrained to move with the underlying material. No mesh constraints can be applied at such corners.

Adaptive mesh constraints must not conflict with other mesh constraints inherent to Lagrangian and sliding boundary regions; therefore, adaptive mesh constraints can be applied only tangentially to a Lagrangian or sliding boundary region. This restriction implies that adaptive mesh constraints can be applied only in two directions in a three-dimensional boundary region, in one direction in a two-dimensional boundary region, or in one direction on a Lagrangian or active geometric edge. It may not always be feasible to adhere to this rule, particularly if the boundary experiences finite rotation. Therefore, if the normal to a boundary region is not perpendicular to a prescribed mesh constraint at a node, it is always moved along the current surface of the boundary region so that the projection of the mesh motion in the direction of the prescribed constraint is correct (see Figure 12.2.2–4).

If the normal to the boundary region approaches the direction of the applied mesh constraint, large mesh motions will be required to satisfy the constraint. By default, an analysis is terminated if the angle between the normal to the boundary region and the direction of the prescribed constraint becomes less than θ_C. This cutoff angle is referred to as the mesh constraint angle, and its default value is 60°.

The mesh constraint angle, θ_C, is also used when adaptive mesh constraints are applied to nodes along a Lagrangian or active geometric edge. Since independent mesh motion cannot be prescribed perpendicular to these edges, an analysis is terminated if the angle between the prescribed constraint and the plane perpendicular to the edge falls below the specified mesh constraint angle.

You can change the value of the mesh constraint angle ($5° \leq \theta_C \leq 85°$). Setting $\theta_C < 45°$ is not recommended because it may cause errors in determining the free surface geometry, especially for curved surfaces.

Input File Usage: *ADAPTIVE MESH CONTROLS, MESH CONSTRAINT ANGLE=θ_C

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Mesh constraint angle:** θ_C

Defining mesh constraints that vary with time

The prescribed magnitude of a nonzero mesh constraint can vary with time during a step according to an amplitude definition (see "Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

*AMPLITUDE, NAME=*name*
*ADAPTIVE MESH CONSTRAINT, AMPLITUDE=*name*

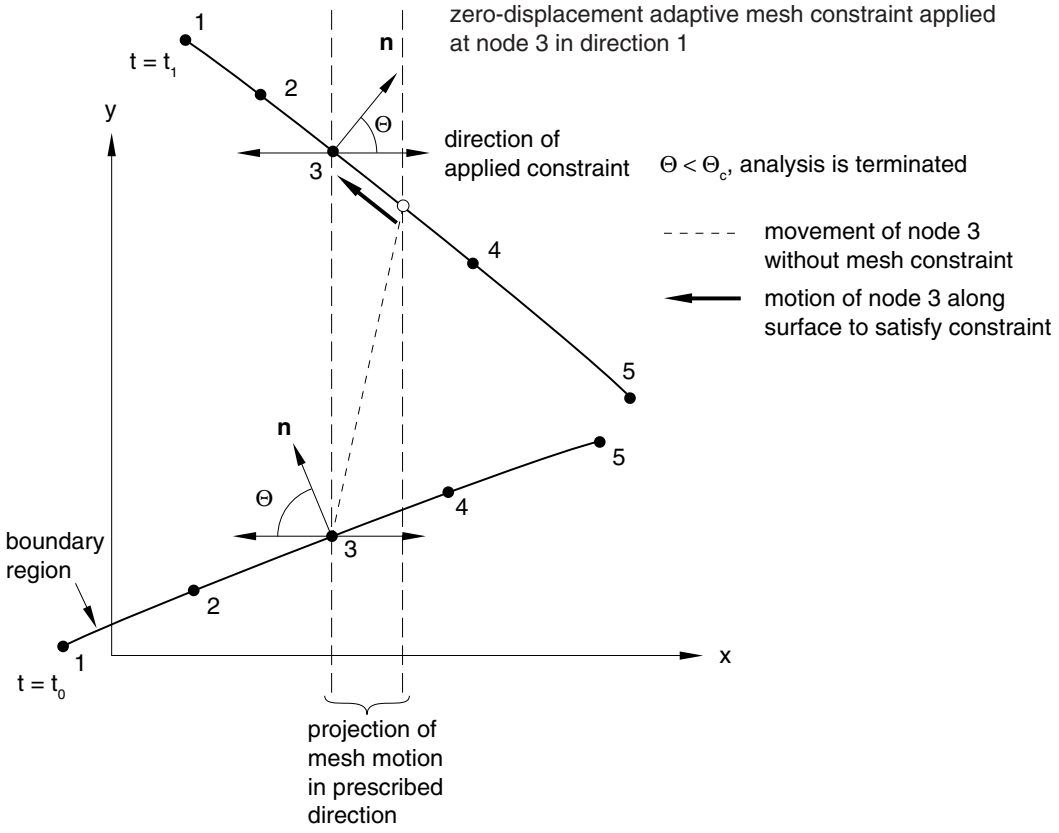


Figure 12.2.2–4 Enforcing a spatial mesh constraint.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**: **Amplitude:** *amplitude*

Applying spatial mesh constraints in local directions

Spatial mesh constraints are applied in local directions if a local coordinate system is defined at a node (see "Transformed coordinate systems," Section 2.1.5); otherwise, they are applied in global directions.

Applying Lagrangian mesh constraints

Lagrangian mesh constraints on a node are used to indicate that mesh smoothing should not be applied; i.e., the node must follow the material.

Input File Usage: *ADAPTIVE MESH CONSTRAINT,
CONSTRAINT TYPE=LAGRANGIAN

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Follow underlying material**

Modifying ALE adaptive mesh constraints

By default, all adaptive mesh constraints defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh constraints in effect for a given step relative to the preexisting adaptive mesh constraints. At each new step the existing adaptive mesh constraints can be modified and additional adaptive mesh constraints can be specified.

Input File Usage: Use either of the following options to modify an existing adaptive mesh constraint or to specify an additional adaptive mesh constraint:

*ADAPTIVE MESH CONSTRAINT,
*ADAPTIVE MESH CONSTRAINT, OP=MOD

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Manager**: select the desired step and mesh constraint: **Edit**

Removing ALE adaptive mesh constraints

If you choose to remove any adaptive mesh constraint in a step, no adaptive mesh constraints will be propagated from the previous step. Therefore, all adaptive mesh constraints that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh constraints and to specify new adaptive mesh constraints:

*ADAPTIVE MESH CONSTRAINT, OP=NEW

If the OP=NEW parameter is used on any *ADAPTIVE MESH CONSTRAINT option within a step, it must be used on all *ADAPTIVE MESH CONSTRAINT options in the step.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Manager**: select the desired step and mesh constraint: **Deactivate**

Initial conditions

There are no initial conditions specific to adaptive meshing; initial conditions can be defined in the same way as in nonadaptive problems. If initial mesh sweeps are performed to smooth the mesh at the beginning of a step (see "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3), all initial conditions (except temperatures and field variables, which are discussed in "Predefined fields," presented later in this section) are remapped to the new mesh. Initial temperatures are remapped during adaptive meshing in an adiabatic analysis.

Initial conditions prescribed near an inflow Eulerian boundary region will affect the state of the material flowing into the domain throughout the analysis. See "Modeling techniques for Eulerian adaptive mesh domains in Abaqus/Explicit," Section 12.2.4, for a discussion of the proper treatment of inflow boundaries.

Defining surfaces on ALE adaptive mesh boundaries

When you define a surface on the boundary of an adaptive mesh domain (see "Surfaces: overview," Section 2.3.1), Abaqus creates a boundary region coinciding with the surface. By default, a sliding boundary region is created. You can choose to create a Lagrangian or Eulerian boundary region instead.

A surface defined in the interior of an adaptive mesh domain will move independently of the material (unless constrained by mesh constraints).

Defining a sliding boundary region using a surface

By default, the boundary region created by a surface definition will be sliding (the surface edge will slide freely over the material).

Input File Usage: *SURFACE, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Defining a Lagrangian boundary region using a surface

To force the surface edge to follow the material, create a Lagrangian boundary region.

Input File Usage: *SURFACE, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Defining an Eulerian boundary region using a surface

To decouple the surface from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the surface will behave like a sliding boundary region (no material will flow through the surface).

As an example, it is often assumed that there is no normal or shear stress in the material at the outflow boundary of an Eulerian domain. This condition can be modeled by defining an Eulerian boundary region using a surface and applying spatial mesh constraints perpendicular to the surface, as shown in Figure 12.2.2–5.

Input File Usage: *SURFACE, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using surfaces are not supported in Abaqus/CAE.

Contact

Lagrangian and sliding boundary regions created using surfaces can be used in contact pairs; they have the same meaning as surfaces defined on nonadaptive regions. Since contact generally involves relative sliding between bodies, sliding boundary regions are typically appropriate for contact surfaces.

Surfaces defined on Eulerian boundary regions cannot be used in contact pairs.

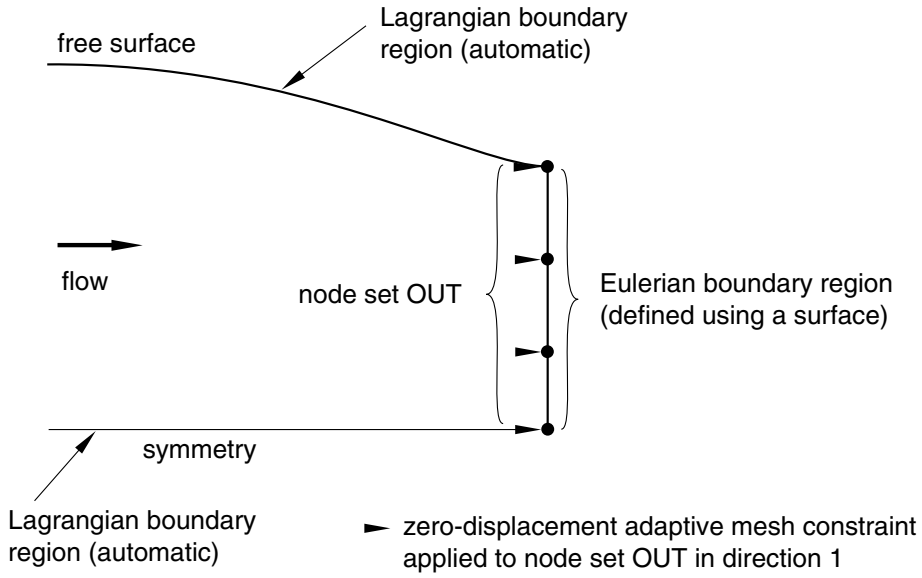


Figure 12.2.2–5 Modeling the outflow boundary of an Eulerian adaptive mesh domain.

If the small-sliding formulation is used for a contact pair, all the nodes on both surfaces are nonadaptive (see "Defining contact pairs in Abaqus/Explicit," Section 36.5.1, and "Contact formulations for contact pairs in Abaqus/Explicit," Section 38.2.2). The nodes of an element-based surface in a no-separation contact pair are nonadaptive (see "Contact pressure-overclosure relationships," Section 37.1.2). All nodes in a general contact domain are nonadaptive (see "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1). Similarly, the nodes at which spot welds are defined are nonadaptive (see "Breakable bonds," Section 37.1.9.)

Distributed loads

When a distributed pressure load is applied to the boundary of an adaptive mesh domain, Abaqus/Explicit creates a boundary region that coincides with the area of the load application. The characteristics of boundary regions created in this way are identical to those of boundary regions created by defining surfaces. If a pressure load is applied to a surface in the interior of an adaptive mesh domain, the nodes on the surface will move with the material in all directions (i.e., they will be nonadaptive).

Boundary regions created by different pressure loads may overlap. If pressure loads with the same magnitude and amplitude definition are applied to adjacent regions, the regions will be merged into a single boundary region to minimize the number of Lagrangian edges and corners formed (see Figure 12.2.2–6).

If a nonuniform pressure is applied (for example, a pressure that varies linearly over a surface) or if a pressure load is defined in user subroutine **VDLOAD**, each element face or edge becomes a separate Lagrangian boundary region. Since Lagrangian corners are formed where Lagrangian edges meet, all

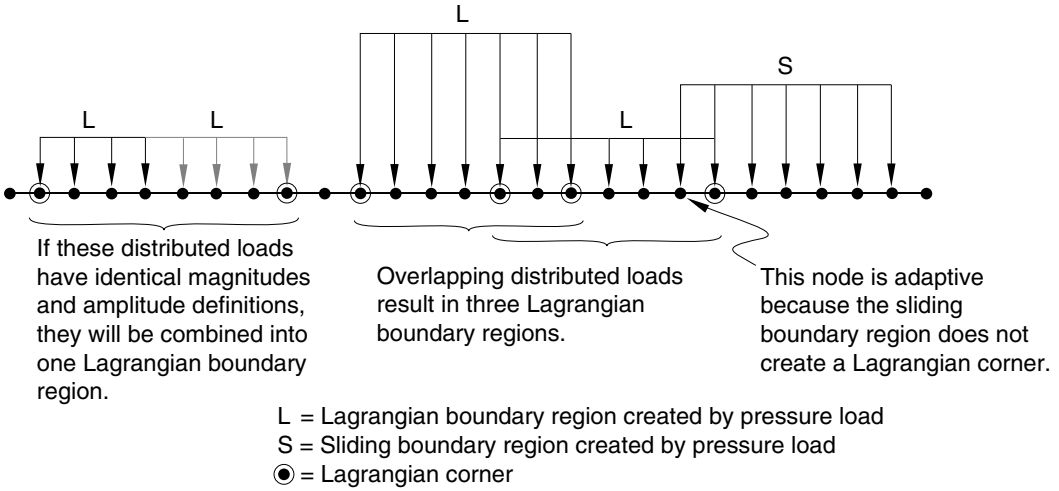


Figure 12.2.2–6 Applying distributed pressure loads to an adaptive mesh domain.

nodes will follow the material in every direction, and each region becomes nonadaptive. Likewise, if a nonuniform body force is applied to an adaptive mesh domain, the domain is split into multiple domains, each with a uniform body force. If this splitting results in one-element domains, the region becomes nonadaptive.

Defining a Lagrangian boundary region with a pressure load

By default, the boundary region created to coincide with a pressure load will be Lagrangian. Pressure loads applied to Lagrangian regions are identical to pressure loads applied to nonadaptive regions, except that the mesh can move inside the boundary region.

Input File Usage: *DLOAD, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

Defining a sliding boundary region with a pressure load

A pressure load can be applied to a sliding boundary region to simulate a load that is fixed in space while material moves past it (see Figure 12.2.2–7). A sliding edge is unconstrained in the direction tangential to the boundary region; therefore, unless adaptive mesh constraints are applied, the area of the load application will move according to the adaptive meshing algorithm, which is probably not physically meaningful.

To allow a pressure load to slide over the material, create a sliding boundary region.

Input File Usage: *DLOAD, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

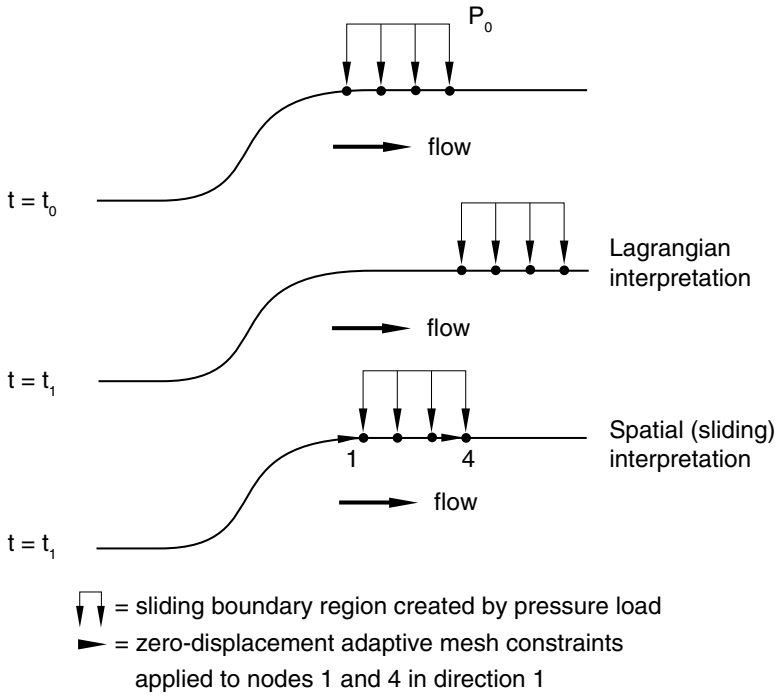


Figure 12.2.2–7 Applying a sliding distributed pressure load to an adaptive mesh domain.

Defining an Eulerian boundary region with a pressure load

To decouple the area of pressure application from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the mesh will behave like a sliding boundary region (no material will flow through the surface).

As an example, it is often assumed that a uniform ambient pressure exists at the outflow boundary of an Eulerian domain. This condition can be modeled by defining the pressure at an Eulerian boundary region using a distributed load and applying spatial mesh constraints perpendicular to the surface, as shown in Figure 12.2.2–8.

Input File Usage: *DLOAD, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using pressure loads are not supported in Abaqus/CAE.

Distributed surface fluxes and thermal conditions

In coupled thermal-stress analysis Abaqus/Explicit also creates boundary regions for distributed surface fluxes, convective film conditions, and radiation conditions. The rules governing boundary regions for

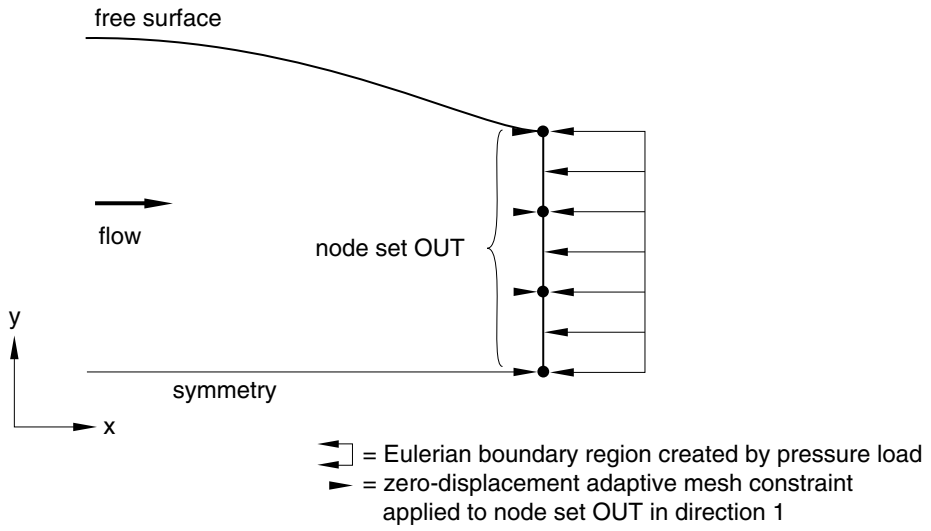


Figure 12.2.2–8 Modeling an ambient pressure at the outflow boundary of an Eulerian adaptive mesh domain.

these loads are identical to those discussed for distributed loads. The ability to specify the boundary region type is also the same.

Concentrated loads

When a concentrated load is applied to the boundary of an adaptive mesh domain, Abaqus/Explicit creates a boundary region to coincide with the load. Every node to which a concentrated load is applied will be considered its own boundary region because it is not possible to identify a surface area associated with a concentrated load. However, you can control the behavior of each one-node region.

If concentrated loads are applied to nodes in the interior of an adaptive mesh domain, those nodes will move with the material in all directions (i.e., they will be nonadaptive).

Defining a Lagrangian boundary region with a concentrated load

By default, the boundary region created by a concentrated load will be Lagrangian. Each one-node Lagrangian boundary region will follow the material in every direction (the nodes will be nonadaptive).

Input File Usage: *CLOAD, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Defining a sliding boundary region with a concentrated load

A concentrated load can be applied to a sliding boundary region to simulate a load that is fixed in space while material moves past it (see Figure 12.2.2–9). A sliding node is unconstrained in the direction

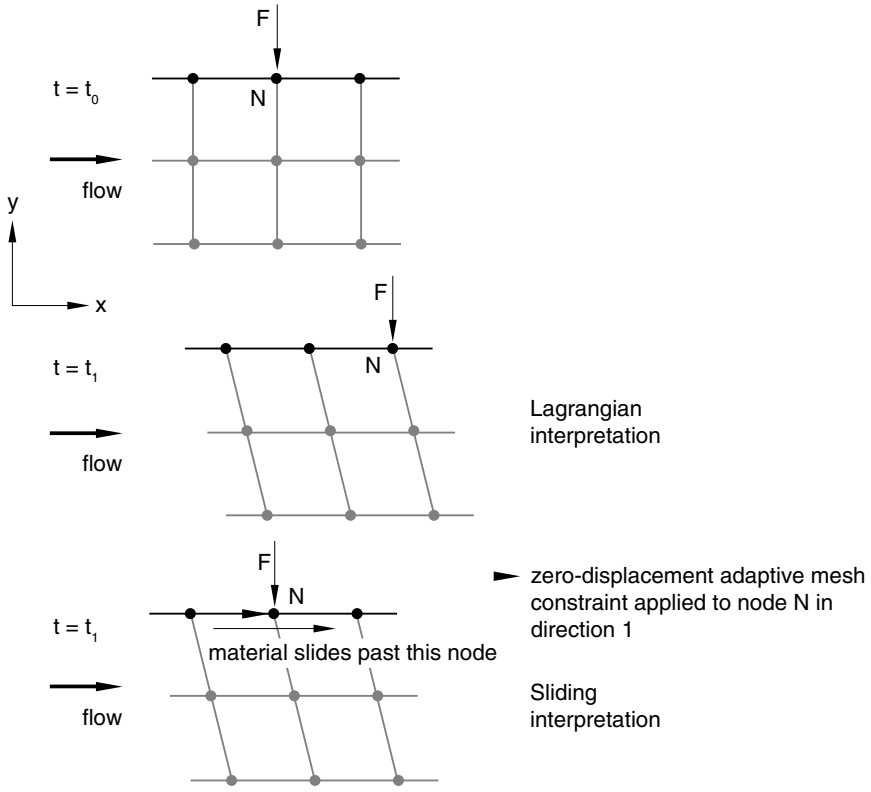


Figure 12.2.2–9 Applying a concentrated sliding load to an adaptive mesh domain.

tangential to the boundary region; therefore, unless adaptive mesh constraints are applied, the point of load application will move according to the adaptive meshing algorithm, which is probably not physically meaningful.

To allow the concentrated load to slide freely over the material, create a sliding boundary region.

Input File Usage: *CLOAD, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Defining an Eulerian boundary region with a concentrated load

To decouple the concentrated load from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, each one-node boundary region will behave like a sliding boundary region.

Input File Usage: *CLOAD, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using concentrated loads are not supported in Abaqus/CAE.

Concentrated fluxes and thermal conditions

In coupled thermal-stress analysis Abaqus/Explicit also creates boundary regions for concentrated heat fluxes, film conditions, and radiation conditions. The rules governing boundary regions for these loads are identical to those discussed for concentrated loads. The ability to specify the boundary region type is also the same.

Boundary conditions

Lagrangian, sliding, and Eulerian boundary regions can be created by applying kinematic constraints to the boundary of an adaptive mesh domain. If kinematic boundary conditions are applied to nodes in the interior of an adaptive mesh domain, those nodes will move with the material in all directions (i.e., they will be nonadaptive), regardless of the specified boundary region type.

Defining a Lagrangian boundary region using a boundary condition

By default, the boundary region created by a kinematic boundary condition will be Lagrangian. Abaqus/Explicit will recognize surface-type and point or edge constraints automatically and will create an appropriate Lagrangian boundary region for each type, as explained in the following subsections.

Input File Usage: *BOUNDARY, REGION TYPE=LAGRANGIAN

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Surface-type constraints applied using boundary conditions

Although boundary conditions are always applied to individual nodes in Abaqus/Explicit, they often represent physical constraints on surfaces. For example, symmetry conditions, where nodes are constrained to move in a plane, are actually surface constraints. A fully clamped (ENCASTRE) condition along a boundary can also be considered a surface constraint. (During adaptive meshing it is meaningful to allow nodes to move along a fully clamped edge.)

Abaqus/Explicit will examine an adaptive mesh boundary and try to create regions that are coincident with the applied boundary conditions. Currently, Abaqus/Explicit can create boundary regions for surface-based constraints on:

- symmetry planes,
- fully clamped planes, and
- planes on which a uniform motion is prescribed.

Figure 12.2.2–2 shows an example in which boundary regions are created by applying surface-type boundary conditions. This figure shows a block of material with a crack and three symmetry planes (therefore, three Lagrangian boundary regions). Material will not flow across any symmetry plane, yet

adaptive meshing can be performed within each Lagrangian boundary region. This flexibility is often helpful in problems that have significant deformation.

Point or edge constraints applied using boundary conditions

Some boundary conditions represent point or edge constraints. For example, a displacement can be prescribed at a node. The boundary regions associated with such nodes are exactly the same as those created by concentrated loads.

Defining a sliding boundary region using a boundary condition

A sliding boundary region associated with a boundary condition can move according to the adaptive meshing algorithm. Since this behavior is probably not physically meaningful, the edges of a sliding boundary region are usually fixed in the direction tangential to the surface using adaptive mesh constraints. This approach can be used, for example, to simulate frictionless contact between a rigid punch and a deformable body, as shown in Figure 12.2.2–10.

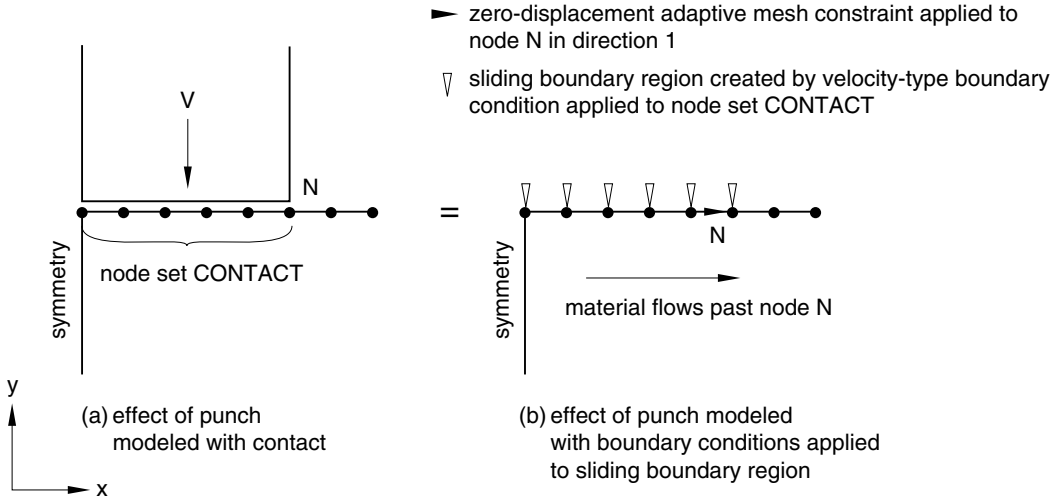


Figure 12.2.2–10 Contact simulation using a sliding boundary region.

In this example the punch is replaced by a sliding boundary region with a constant velocity boundary condition applied in the area of "contact." A tangential mesh constraint is applied to the edge of the boundary region at node N (the other edge is constrained by the Lagrangian boundary region created automatically on the symmetry plane). This problem definition allows material to flow radially underneath the "punch" while retaining the original size and location of the "contact" area.

Abaqus/Explicit makes no distinction between surface-type constraints and point or edge constraints for sliding boundary regions.

To allow the boundary condition to slide freely over the material, create a sliding boundary region.

Input File Usage: *BOUNDARY, REGION TYPE=SLIDING

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Defining an Eulerian boundary region using a boundary condition

To decouple the boundary region from the material motion, create an Eulerian boundary region and apply spatial mesh constraints normal to the surface. If no mesh constraints are applied, the mesh will behave like a sliding boundary region (no material will flow through the surface).

As an example, the mass flow rate at an Eulerian inflow boundary can be prescribed by defining an Eulerian boundary region using a boundary condition.

Abaqus/Explicit makes no distinction between surface-type constraints and point or edge constraints for Eulerian boundary regions.

Input File Usage: *BOUNDARY, REGION TYPE=EULERIAN

Abaqus/CAE Usage: Boundary regions defined using boundary conditions are not supported in Abaqus/CAE.

Overlapping boundary regions

A Lagrangian boundary region can overlap any number of other Lagrangian or sliding boundary regions (see Figure 12.2.2–11). If two boundary regions partially overlap, three regions are formed: the overlapping region and the two initial regions minus the overlapping region. A sliding boundary region is formed when a Lagrangian and a sliding boundary region overlap.

An Eulerian boundary region can never overlap a Lagrangian or sliding boundary region. Furthermore, an Eulerian boundary region can never share a boundary with or overlap a nonadaptive region. Because infinite elements are nonadaptive, this latter restriction implies that infinite elements cannot be used to simulate ambient conditions at an outflow boundary.

Coincident edges

Edges shared by different types of boundary regions are subject to the following rules:

- An edge shared between a Lagrangian and a sliding boundary region will be Lagrangian.
- An edge shared between a Lagrangian and an Eulerian boundary region will be sliding.
- An edge shared between a Lagrangian and a nonadaptive boundary region will be nonadaptive.
- An edge shared between a sliding and a nonadaptive boundary region will be nonadaptive.
- An edge of an Eulerian boundary region can never be coincident with an edge of a nonadaptive region.

Predefined fields

There are no restrictions on applying prescribed temperatures or field variables in an adaptive mesh domain, but these nodal values are not remapped when adaptive meshing is performed. Therefore, predefined fields that are not spatially uniform may not be meaningful within an adaptive mesh domain.

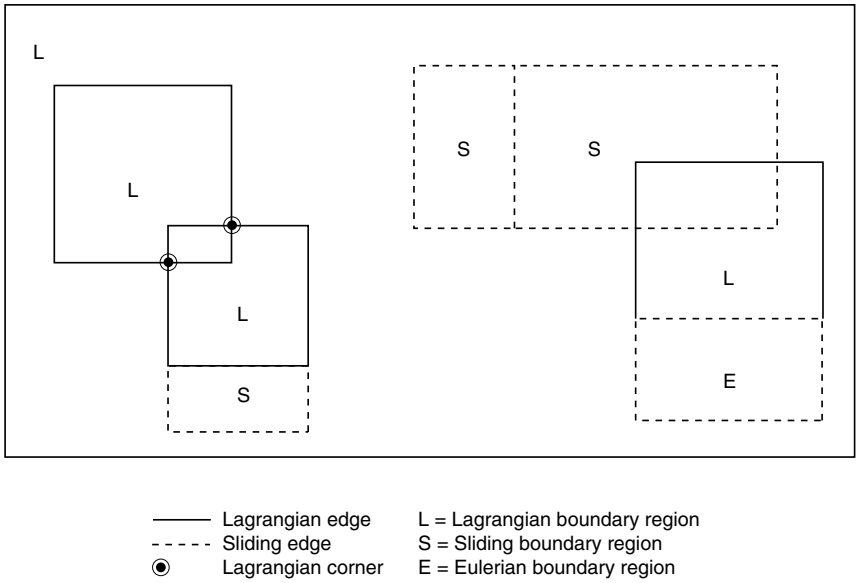


Figure 12.2.2–11 Overlapping boundary regions.

(Time-varying, spatially uniform predefined fields are acceptable, since adaptive meshing is applied at discrete instances in time.) However, if temperature or field variable data are collected from a spatial frame of reference, it may make physical sense to apply a spatially varying field for an Eulerian domain in which the mesh does not move. Abaqus/Explicit does not perform any checks or calculations on predefined fields for adaptive meshing; you must ensure that the predefined fields are meaningful.

Materials

All material models and behaviors, except brittle cracking ("Cracking model for concrete," Section 23.6.2), fabric ("Fabric material behavior," Section 23.4.1), and low-density foam ("Low-density foams," Section 22.9.1) materials, can be used in an adaptive mesh domain.

For domains modeled with hyperelastic or hyperfoam materials the usefulness of adaptive meshing is limited. The recommended enhanced hourglass method ("Section controls," Section 27.1.4), which will generally predict a much better return to the original configuration for these materials when loading is removed, cannot be used in an adaptive mesh domain. Therefore, for hyperelastic or hyperfoam materials it is recommended that the analysis be run without adaptive meshing but with enhanced hourglass control.

If the porous failure model ("Failure criteria in Abaqus/Explicit" in "Porous metal plasticity," Section 23.2.9), shear failure model ("Shear failure model" in "Dynamic failure models," Section 23.2.8), tensile failure model ("Tensile failure model" in "Dynamic failure models," Section 23.2.8), or one of the progressive damage models (Chapter 24, "Progressive Damage and Failure") is specified within an adaptive mesh domain, Abaqus/Explicit will continuously monitor the status of elements while performing adaptive meshing. When elements within the domain fail, the nodes along the interface

between the failed and unfailed elements will become nonadaptive. This has the effect of creating a material boundary between the failed and unfailed zones.

When failure occurs in elements that use the shear failure, the tensile failure, or the progressive damage models without element deletion, elements in the failure zone will not be deleted; they can carry some states of stress. Adaptive meshing will occur within the failure zone but not along the interface with the unfailed material.

Elements

An adaptive mesh domain can contain only first-order, reduced-integration, solid elements. All elements within an adaptive mesh domain must have the same geometry (all two-dimensional, three-dimensional, axisymmetric, or plane strain, etc.). Since adaptive mesh domains are split across element types, degenerate elements should be used for mixed domains that include both triangles and quadrilaterals (or tetrahedron and bricks). All elements other than first-order, reduced-integration, solid elements—including mass, rotary inertia, and infinite elements—are nonadaptive. These elements can be connected to an adaptive mesh domain, but their nodes are nonadaptive. All nodes and elements on a rigid body are nonadaptive. Rebar are not supported within an adaptive mesh domain.

Multi-point constraints and equations

As with boundary conditions, multi-point constraints (“General multi-point constraints,” Section 35.2.2) and equations (“Linear constraint equations,” Section 35.2.1) are always applied to nodes but sometimes represent constraints on surfaces. Abaqus/Explicit will recognize surface-type constraints when the following conditions are satisfied:

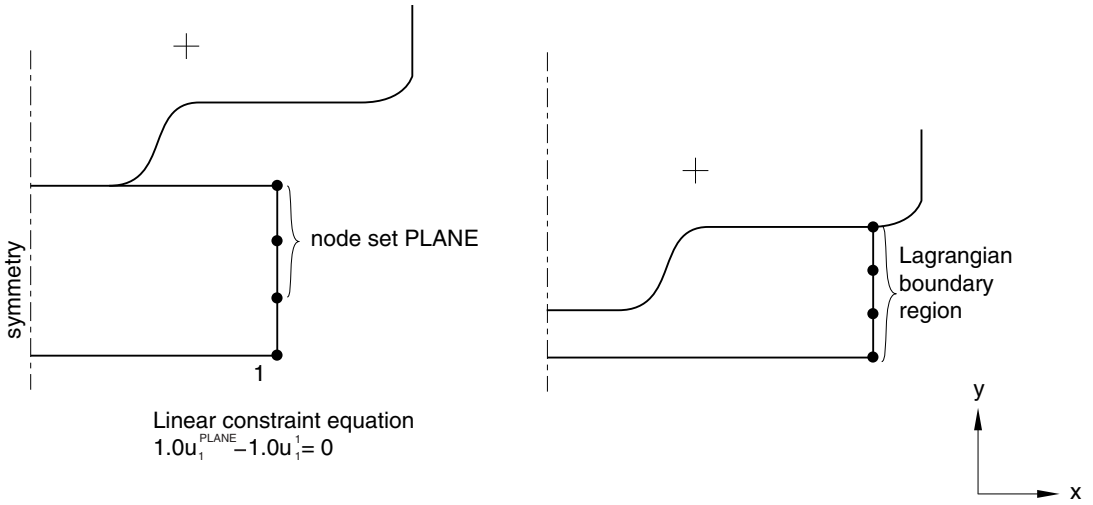
- an equation, PIN MPC, or TIE MPC ties a node set to a single node; and
- all the nodes involved in the MPC or equation are coplanar and lie within the boundary region.

If these conditions are satisfied, a boundary region will be associated with the node set in the MPC or equation. If the MPC is applied within a Lagrangian or sliding boundary region, a Lagrangian edge will be created. If the MPC is applied within an Eulerian boundary region, no edge will be created. If the above conditions are not satisfied, all nodes connected to the MPC or equation will be nonadaptive.

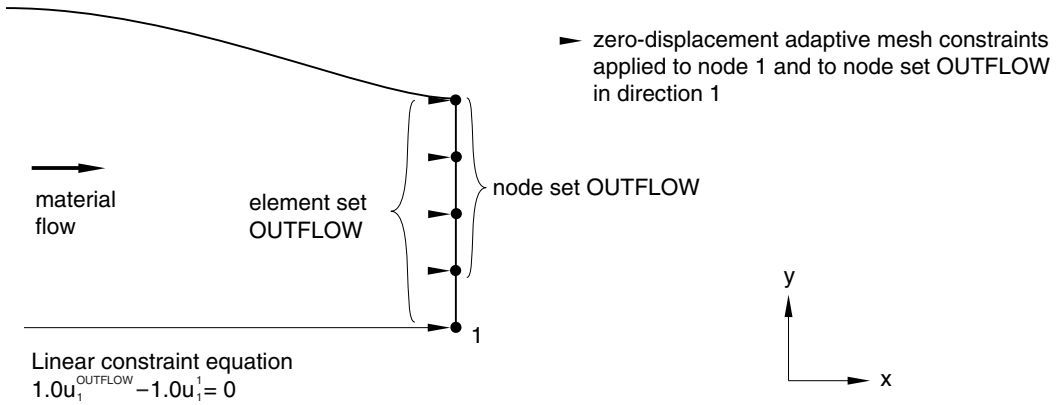
As an example, a constraint can be applied to force a plane section to remain plane in a Lagrangian adaptive mesh domain, as shown in Figure 12.2.2–12(a). In this case all nodes are constrained by an equation to lie in the same plane throughout the analysis, but adaptive meshing is allowed within the plane.

As another example, consider the outflow boundary of an Eulerian domain, as shown in Figure 12.2.2–12(b). The outflow boundary of an Eulerian domain is often assumed to be far enough downstream that the velocity is uniform but unknown. To model this condition, an Eulerian boundary region is created at the outflow boundary using a surface. An adaptive mesh constraint is used to fix the mesh perpendicular to the boundary, and all nodes on the plane are constrained by an equation to have the same velocity orthogonal to the plane.

For surface-based tie constraints (see “Mesh tie constraints,” Section 35.3.1), all nodes on the tied surfaces will be nonadaptive.



(a) Using an equation to force a plane section to remain a plane.



(b) Using an equation to prescribe a uniform velocity outflow condition.

Figure 12.2.2–12 Using equations with adaptive meshing.

Procedures

During an adiabatic analysis temperatures will be remapped properly in adaptive mesh domains. Adaptive meshing is not used during annealing procedures or during geometrically linear analyses.

The definitions of adaptive mesh domains, boundary regions, mesh constraints, and controls (as explained in "ALE adaptive meshing and remapping in Abaqus/Explicit," Section 12.2.3) will propagate from step to step.

User subroutines

Solution-dependent state variables defined in user subroutine **VUMAT** will be remapped to the new mesh when adaptive meshing is performed.

Solution-dependent state variables that are defined on a slave surface in user subroutines **VFRIC**, **VUINTER**, **VFRICTION**, and **VUINTERACTION** will not be remapped to the new mesh when adaptive meshing is performed. Therefore, to ensure physically meaningful results, a Lagrangian adaptive mesh constraint should be used for nodes on the contact slave surfaces with solution-dependent state variables where the contact constraint is defined using these user subroutines.

Output

Since the mesh is no longer constrained to the material when adaptive meshing is performed, output at elements and nodes must be interpreted differently than in a pure Lagrangian problem. See "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5, for details.

Input file template

To create a Lagrangian adaptive mesh domain:

```
*HEADING
 …
*ELSET, ELSET=ADAPT
*************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
*ADAPTIVE MESH, ELSET=ADAPT
...
*END STEP
```

To create an Eulerian adaptive mesh domain with a prescribed velocity inflow condition and a prescribed pressure outflow condition (both in the global x-direction):

```
*HEADING...
*ELSET, ELSET=ADAPT
...
*ELSET, ELSET=OUT
...
*NSET, NSET=INFLOW
...
```

```
*NSET, NSET=OUTFLOW
...
*SURFACE, NAME=INSURF, REGION TYPE=EULERIAN
Data lines to define the surface
*SURFACE, NAME=OUTSURF, REGION TYPE=EULERIAN
Data lines to define the surface
...
*EQUATION
Data lines specifying uniform velocity at the inflow
************************
*STEP
*DYNAMIC, EXPLICIT
Data line to specify the time period of the step
*ADAPTIVE MESH, ELSET=ADAPT
*ADAPTIVE MESH CONSTRAINT
 INFLOW,  1, 1, 0
 OUTFLOW, 1, 1, 0
*BOUNDARY, TYPE=VELOCITY, REGION TYPE=EULERIAN
 INFLOW, 1, 1, 10.0
*DLOAD, REGION TYPE=EULERIAN
 OUT, P2, 15.0
...
*END STEP
```

12.2.3 ALE ADAPTIVE MESHING AND REMAPPING IN Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References

- "ALE adaptive meshing: overview," Section 12.2.1
- "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2
- "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5
- *ADAPTIVE MESH
- *ADAPTIVE MESH CONSTRAINT
- *ADAPTIVE MESH CONTROLS
- "Customizing ALE adaptive meshing," Section 14.14 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

ALE adaptive meshing consists of two fundamental tasks:

- creating a new mesh, and
- remapping solution variables from the old mesh to the new mesh with a process called advection.

The success of the adaptive meshing technique depends on the choice of the methods used for each of these tasks. The default methods for creating a new mesh and for remapping solution variables have been chosen carefully to work for a wide variety of problems. However, you may wish to override the default choices to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing to more difficult or unusual applications.

Meshing

A new mesh:

- is created at a specified frequency for each adaptive domain;
- is found by sweeping iteratively over the adaptive mesh domain and moving nodes to smooth the mesh; and
- can retain the initial gradation of the original mesh.

Remapping

The methods used for advecting solution variables to the new mesh:

- are consistent, monotonic, and (by default) accurate to the second order; and
- conserve mass, momentum, and energy.

Controlling the frequency of ALE adaptive meshing

In most cases the frequency of adaptive meshing is the parameter that most affects the mesh quality and the computational efficiency of adaptive meshing. A typical adaptive mesh application without Eulerian boundaries will require adaptive meshing every 5–100 increments. In contrast, adaptive meshing should generally be performed much more frequently in a steady-state process simulation using Eulerian boundaries. Thus, if a spatial adaptive mesh constraint or an Eulerian boundary region is defined on an adaptive mesh domain, the default frequency is 1; otherwise, the default frequency is 10.

Input File Usage: Use the following option to change the frequency of adaptive meshing:

∗ADAPTIVE MESH, FREQUENCY=*number of increments*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Frequency:** *number of increments*

Controlling the intensity of ALE adaptive meshing

During each adaptive meshing increment, the new mesh is created by performing one or more mesh sweeps and then advecting the solution variables to the new mesh.

Mesh sweeps

In an adaptive meshing increment, a new, smoother mesh is created by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated—based on the current positions of neighboring nodes and elements—to reduce element distortion. In a typical sweep a node is moved a fraction of the characteristic length of any element surrounding the node. Increasing the number of sweeps increases the intensity of adaptive meshing in each adaptive meshing increment. The default number of mesh sweeps is one.

Input File Usage: Use the following option to change the number of mesh sweeps to be performed in each adaptive mesh increment:

∗ADAPTIVE MESH, MESH SWEEPS=*number of sweeps*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Remeshing sweeps per increment:** *number of sweeps*

Advection sweeps

The process of mapping solution variables from an old mesh to a new mesh is referred to as an advection sweep. At least one advection sweep is performed in every adaptive mesh increment. Ideally, an advection sweep will be performed only once, after all mesh sweeps for the increment are complete. However, numerical stability of the advection sweep is maintained only if the difference between the old mesh and the new mesh is small. Therefore, if after a mesh sweep the total accumulated movement of any node in the domain is greater than 50% of the characteristic length of any adjacent element, an advection sweep is performed to remap the solution variables from the old mesh to the intermediate

mesh. Mesh sweeps will continue until the specified number is reached or until the movement of any node again exceeds the 50% threshold. At this time an advection sweep is performed again to map variables from the last intermediate mesh to the new intermediate mesh. The cycle will continue until the number of mesh sweeps reaches the specified number.

The number of advection sweeps per adaptive mesh increment required for each adaptive mesh domain is determined automatically by Abaqus/Explicit; you cannot override this automatic calculation. The number of advection sweeps is printed by default to the message (`.msg`) file (see "Output and diagnostics for ALE adaptive meshing in Abaqus/Explicit," Section 12.2.5).

The computational cost of ALE adaptive meshing

The cost of adaptive meshing depends on the frequency of remeshing, the number of mesh and advection sweeps performed, and the size of the adaptive mesh domains. When compared to a purely Lagrangian analysis, additional computational cost is incurred only within adaptive mesh increments.

Generally, the cost of one advection sweep is several times greater than the cost of one mesh sweep. Multiple advection sweeps are triggered when adaptive meshing is performed too infrequently and/or a high number of mesh sweeps is specified. Performing adaptive meshing more frequently and doing 1–5 mesh sweeps in each adaptive mesh increment will usually generate only one advection sweep, minimizing the computational cost.

The relatively smooth mesh and improved element aspect ratios that result from adaptive meshing may increase the stable time increment compared to a similar pure Lagrangian analysis. In some cases this increase can offset the cost of adaptive meshing completely.

Although computational cost can vary greatly with the type of application, performing adaptive meshing on the entire problem domain in every increment will typically increase the cost of the analysis by 3–5 times that of a similar Lagrangian analysis. Defining adaptive mesh domains that cover only a fraction of the entire problem domain will reduce the cost proportionally. Changing the frequency to every 10–25 increments will result in CPU times that are only moderately higher than those for a pure Lagrangian analysis.

Guidelines for controlling ALE adaptive meshing frequency and intensity

Although the default values work well for many problems, difficult analyses may require a more frequent adaptive meshing frequency or meshing with a higher intensity.

Guidelines for transient analysis

For problems without spatial adaptive mesh constraints or Eulerian boundary regions, the default frequency for adaptive meshing is 10, and the default number of mesh sweeps is 1. The default values are usually adequate for low- to moderate-rate dynamic problems and for quasi-static process simulations undergoing moderate deformation. If the frequency or number of mesh sweeps is too low, excess element distortion may cause the analysis to terminate before the mesh is adapted; or, if a solution can be obtained, it may not be as accurate as the solution that could be obtained with a higher quality mesh. In virtually all cases, however, performing adaptive meshing at any frequency will

reduce the distortion of elements (and, thus, improve the quality of the solution) compared to a pure Lagrangian analysis.

For high-rate impact problems undergoing large amounts of deformation, it may be necessary to increase the frequency of adaptive meshing or the number of mesh sweeps. It is generally less expensive to increase the number of mesh sweeps slightly before increasing the frequency, as long as the number of advection sweeps remains small.

For problems involving explosions taking place over just a few hundred increments, adaptive meshing is usually required at every increment. It may also be necessary to increase the frequency of adaptive meshing for quasi-static process simulations that involve large amounts of flow per increment.

For problems in which the deformation per increment is small, a high-quality mesh can be maintained by performing adaptive meshing only every 25–100 increments. For these problems the additional cost of adaptive meshing is negligible.

Guidelines for steady-state analysis

When an adaptive mesh domain contains Eulerian boundary regions or has spatial adaptive mesh constraints, the default frequency of adaptive meshing is 1. This default frequency is conservative and is chosen primarily because spatial mesh constraints are applied only during adaptive mesh increments. Thus, between adaptive mesh increments the mesh may drift from its prescribed location, which may affect the solution. However, drift from adaptive mesh constraints will always be eliminated in the next adaptive mesh increment: it will not accumulate.

For problems in which the speed of deformation or the speed of material flow from element to element is much less than the material wave speed, the frequency typically can be increased to 5 or higher. This class of problems includes most steady-state process simulations, where the drift of the mesh from the prescribed location is negligible over a few increments. By performing adaptive meshing less often, steady-state simulations become competitive with their corresponding transient simulations. For Eulerian domains in which the speed of the deformation or material flow is high, such as in dynamic shock problems, the default frequency of 1 should be used.

Mesh smoothing methods

The determination of the new mesh in Abaqus/Explicit is based on four aspects. You can control each of these aspects by defining adaptive mesh controls. Defaults have been chosen so that the overall algorithm works well for most problems.

First, the calculation of the new mesh in Abaqus/Explicit is based on some combination of three basic smoothing methods: volume smoothing, Laplacian smoothing, and equipotential smoothing. The smoothing methods are applied at each node in the adaptive mesh domain to determine the new location of the node based on the locations of surrounding nodes or elements. Although all the smoothing methods tend to smooth the mesh and reduce element distortion, the resulting meshes will differ depending on the methods used.

Second, initial element gradation can be maintained at the expense of element distortion if desired. Third, optimal positioning of the nodes before the basic smoothing methods are applied can improve mesh quality and minimize the frequency of adaptive meshing required.

Finally, solution-dependent meshing is used to concentrate mesh refinement near areas of evolving boundary curvature. This counteracts the tendency of the basic smoothing methods to reduce the mesh refinement near concave boundaries where solution accuracy is important.

Volume smoothing

Volume smoothing relocates a node by computing a volume-weighted average of the element centers in the elements surrounding the node. In Figure 12.2.3–1 the new position of node M is determined by a volume-weighted average of the positions of the element centers, C, of the four surrounding elements. The volume weighting will tend to push the node away from element center C1 and toward element center C3, thus reducing element distortion.

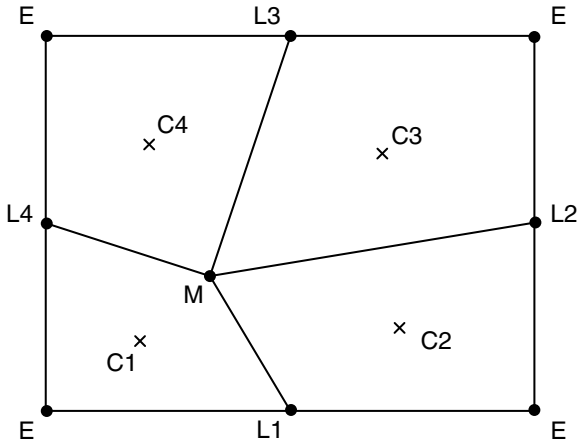


Figure 12.2.3–1 Relocation of a node during a mesh sweep.

Volume smoothing is very robust and is the default method in Abaqus/Explicit. It works well for both structured and highly unstructured domains. (A structured domain is one that contains no degenerate elements and where every node is surrounded by four elements in two dimensions or eight elements in three dimensions.)

Laplacian smoothing

Laplacian smoothing relocates a node by calculating the average of the positions of each of the adjacent nodes connected by an element edge to the node in question. In Figure 12.2.3–1 the new position of node M is determined by averaging the positions of the four nodes, L, connected to M by element edges. The locations of nodes L2 and L3 will pull node M up and to the right to reduce element distortion.

Laplacian smoothing is the least expensive smoothing algorithm and is commonly used in mesh preprocessors. For low to moderately distorted mesh domains, the results of Laplacian smoothing are similar to volume smoothing. For domains with boundaries of complex curvature, volume smoothing generally results in a more balanced mesh.

Equipotential smoothing

Equipotential smoothing is a higher-order method that relocates a node by calculating a higher-order, weighted average of the positions of the node's eight nearest neighbor nodes in two dimensions (or its eighteen nearest neighbor nodes in three dimensions). In Figure 12.2.3–1 the new position of node M is based on the position of all the surrounding nodes, L and E.

The weighted averaging for the equipotential smoothing method is fairly complex and is based on the solution of the Laplace equation. Equipotential smoothing tends to minimize the local curvature of lines running across a mesh over several elements. Although this tendency can be desirable for gently curving domains, it can inhibit the ability of equipotential smoothing to reduce element distortion in highly deformed and locally curved domains.

Equipotential smoothing can be performed only for nodes that are surrounded by a locally structured mesh. Nodes that are surrounded by an unstructured mesh are moved with an equivalent amount of volume smoothing when equipotential smoothing is chosen.

Combining smoothing methods

The default smoothing method in Abaqus/Explicit is volume smoothing. To choose an alternate smoothing method or to combine smoothing methods, you specify the weighting factor for each method. When more than one smoothing method is used, a node is relocated by computing a weighted average of the locations predicted by each chosen method. All weights must be positive, and their sum should typically be 1.0. If the sum of the chosen weights is less than 1.0, the mesh smoothing algorithm will be less aggressive at each adaptive mesh increment. If the sum of the chosen weights is greater than 1.0, their values are normalized so that their sum is 1.0.

Input File Usage: *ADAPTIVE MESH CONTROLS, NAME=*name*
volume smoothing weight, Laplacian smoothing weight, equipotential smoothing weight

For example, the following option could be used to define an equal blend of volume and equipotential smoothing, with no Laplacian smoothing:

```
*ADAPTIVE MESH CONTROLS, NAME=name
0.5, 0.0, 0.5
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Volumetric:** *volume smoothing weight*, **Laplacian:** *Laplacian smoothing weight*, **Equipotential:** *equipotential smoothing weight*

Geometric enhancements to the basic smoothing methods

The conventional forms of the basic smoothing methods do not perform well in highly distorted domains. To ensure the robustness of adaptive meshing, Abaqus/Explicit uses geometrically enhanced forms of the basic smoothing algorithms by default. The enhanced forms are recommended for all adaptive mesh applications. However, since the basic smoothing algorithms are used by many finite element preprocessors, their conventional forms are provided as an option.

Input File Usage: Use the following option to use the conventional forms of the volume, Laplacian, or equipotential smoothing algorithms:

*ADAPTIVE MESH CONTROLS, NAME=*name*,
GEOMETRIC ENHANCEMENT=NO

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, toggle off **Use enhanced algorithm based on evolving element geometry**

Specifying a uniform mesh smoothing objective

For adaptive mesh domains without any Eulerian boundary regions, the default objective of the mesh smoothing methods is to minimize mesh distortion while improving element aspect ratios, at the expense of diffusing initial mesh gradation. The uniform mesh smoothing objective is recommended for problems with moderate to large overall deformation.

Input File Usage: Use the following option to specify the uniform mesh smoothing objective:

*ADAPTIVE MESH CONTROLS, NAME=*name*,
SMOOTHING OBJECTIVE=UNIFORM

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Priority: Improve aspect ratio**

Specifying a graded mesh smoothing objective

Alternatively, the smoothing methods can attempt to preserve initial mesh gradation while reducing element distortion as the analysis evolves. This objective is the default for adaptive mesh domains with one or more Eulerian boundary regions. The graded mesh smoothing objective is recommended only for adaptive mesh domains with reasonably structured graded meshes undergoing low to moderate overall deformation. Element distortion will be minimized, but the aspect ratios of adjacent elements will be maintained approximately. Mesh gradation is particularly useful in steady-state problems where overall deformations are small and a focused mesh is used in a specific area to capture high solution gradients.

Input File Usage: Use the following option to specify the graded mesh smoothing objective:

*ADAPTIVE MESH CONTROLS, NAME=*name*,
SMOOTHING OBJECTIVE=GRADED

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Priority: Preserve initial mesh grading**

Positioning nodes in Lagrangian domains

If an adaptive mesh domain has no Eulerian boundary regions, then, by default, the mesh sweeps are based on current nodal locations, which account for material motion accumulated since the last adaptive mesh increment. This approach is generally the best for Lagrangian problems that undergo large overall deformation.

Input File Usage: Use the following option to request that the current deformed positions of nodes be used as the starting locations for mesh smoothing:

∗ADAPTIVE MESH CONTROLS, NAME=*name*,
MESHING PREDICTOR=CURRENT

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Meshing predictor: Current deformed position**

Positioning nodes in Eulerian domains

Mesh sweeps can be based on the locations of nodes at the end of the previous adaptive mesh increment. This technique is recommended for problems that are Eulerian in nature, where material flow is significant compared to overall deformation. Therefore, it is the default for adaptive mesh domains with one or more Eulerian boundary regions. This approach will result in a virtually stationary mesh.

Input File Usage: Use the following option to use the position of the nodes at the end of the previous adaptive mesh increment as a starting location for mesh smoothing:

∗ADAPTIVE MESH CONTROLS, NAME=*name*,
MESHING PREDICTOR=PREVIOUS

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Meshing predictor: Position from previous adaptive mesh increment**

Solution-dependent meshing based on concave boundary curvature

Mesh smoothing algorithms based only on minimizing element distortion tend to reduce the mesh refinement in areas of concave curvature, especially as the curvature evolves. Having sufficient mesh refinement near highly curved boundaries is often important to model both the shape and volume of the domain. To prevent the natural reduction in mesh refinement of areas near evolving concave curvature, Abaqus/Explicit uses solution-dependent meshing to focus mesh gradation toward these areas automatically.

Although solution-dependent meshing may "pull" more elements into areas of high curvature, its primary purpose is to retain the nominal refinement in these zones. Therefore, a fine mesh should always be used when and where highly curved boundaries are expected and solution-dependent meshing should generally not be used as a direct substitute for more elements.

The aggressiveness of the solution-dependent meshing is governed by the curvature refinement weight, α_C. By default, $\alpha_C = 1$, which correponds to an aggressivity level that has been chosen to work well on a wide variety of problems. You can change the curvature refinement weight. A value of zero indicates no solution dependence due to evolving boundary curvature, and a value greater than one increases the aggressivity from the default.

Input File Usage: ∗ADAPTIVE MESH CONTROLS, NAME=*name*,
CURVATURE REFINEMENT=α_C

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Curvature refinement:** α_C

Smoothing a distorted mesh at the beginning of a step

When an adaptive mesh domain contains a structured mesh of uniform density, the mesh will move independently from the material only when the domain deforms. If the mesh is initially nonuniform, the meshing algorithms in Abaqus/Explicit will smooth the mesh even in the absence of deformation or material transport.

When the initial mesh contains highly distorted elements, it is often useful to smooth the mesh before the step begins so that the best possible mesh is used throughout the step. When a uniform smoothing objective is used, five mesh sweeps are performed by default at the beginning of the step in which the adaptive mesh domain is defined. For a graded smoothing objective, two mesh sweeps are performed by default at the beginning of the step without acccounting for gradation. The aspect ratios used for gradation in all subsequent mesh sweeps are based on this locally smoothed mesh.

Initial conditions are advected to the new mesh when initial mesh sweeps are performed.

Input File Usage: Use the following option to change the number of mesh sweeps that will be performed at the beginning of the first step in which the adaptive mesh definition is active:

∗ADAPTIVE MESH, INITIAL MESH SWEEPS=*number of initial sweeps*

For example, the following option would smooth a badly distorted mesh with 15 mesh sweeps at the beginning of the step, before performing adaptive meshing with three mesh sweeps every 20 increments throughout the step:

```
*ADAPTIVE MESH, FREQUENCY=20, MESH SWEEPS=3,
INITIAL MESH SWEEPS=15
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, **Initial remeshing sweeps: Value:** *number of initial sweeps*

Meshing on boundary regions

Adaptive meshing on Lagrangian and sliding boundary regions is subject to the constraint that the mesh and material must move together in the direction normal to the boundary. Nodes on the interior of such a boundary are allowed to slide freely over the material within the boundary region, which maximizes the amount of mesh smoothing that can be performed. Nodes are positioned in each mesh sweep by applying the basic smoothing methods while constraining the nodes to lie on the boundary region. In three dimensions some nodes on Lagrangian boundary regions will be on a Lagrangian edge. In each mesh sweep these nodes are positioned by applying the basic smoothing methods while constraining the node to lie along the discrete Lagrangian edge.

For problems in which the flow of material from element to element along the boundary is significant compared to the deformation, oscillations in the boundary mesh can result if these constraints are applied symmetrically with respect to the upstream and downstream directions of material flow. Abaqus/Explicit

uses a Petrov-Galerkin weighting of the free boundary constraint to suppress any oscillations. The algorithm is volume preserving, and the degree of upwinding is chosen automatically.

Advecting solution variables to the new mesh

The framework for adaptive meshing in Abaqus/Explicit is the Arbitrary Lagrangian-Eulerian method, which introduces advective terms into the momentum balance and mass conservation equations to account for independent mesh and material motion. There are two basic ways to solve these modified equations: solve the nonsymmetric system of equations directly, or decouple the Lagrangian (material) motion from the additional mesh motion using an operator split. The operator split method is used in Abaqus/Explicit because of its computational efficiency. Furthermore, this technique is appropriate in an explicit setting because small time increments limit the amount of motion within a single increment.

In an adaptive meshing increment the element formulations, boundary conditions, external loads, contact conditions, etc. are handled first in a manner consistent with a pure Lagrangian analysis. Once the Lagrangian motion is updated and mesh sweeps have been performed to find the new mesh, the solution variables are remapped by performing an advection sweep. The advection sweep accounts for the advective terms in the momentum balance and continuity equations.

Advection methods for element variables

Element and material state variables must be transferred from the old mesh to the new mesh in each advection sweep. The number of variables to be advected depends on the material model and element formulation; however, stress, history variables, density, and internal energy are always solution variables. Two methods are available for the advection of element variables: the default second-order method based on the work of Van Leer (Van Leer, 1977) and a first-order method based on donor cell differencing.

Both advection methods incorporate the concept of upwinding. They also conserve the element variables in an integral sense when mapping from the old mesh to the new mesh (that is, the value of any solution variable integrated over the domain is unchanged by adaptive meshing). Using a conservative algorithm to advect the element density and the internal energy automatically ensures conservation of mass and energy for an adaptive mesh domain without Eulerian boundary regions.

Both advection methods are also monotonic and consistent. A method is monotonic if an element quantity with a monotonic, increasing spatial distribution over a portion of the old mesh remains as such in the new mesh. A method is consistent if, when solution variables are advected to a new mesh that is identical to the old mesh, all element quantities remain unchanged.

Second-order advection

Second-order advection is used by default for all adaptive mesh domains. It is recommended for all problems, ranging from quasi-static to transient dynamic shock. An element variable, ϕ, is remapped from the old mesh to the new mesh by first determining a linear distribution of the variable in each old element, as illustrated in Figure 12.2.3–2 for a simple one-dimensional mesh. The linear distribution of ϕ in the middle element depends on the values of ϕ in the two adjacent elements. To construct the linear distribution:

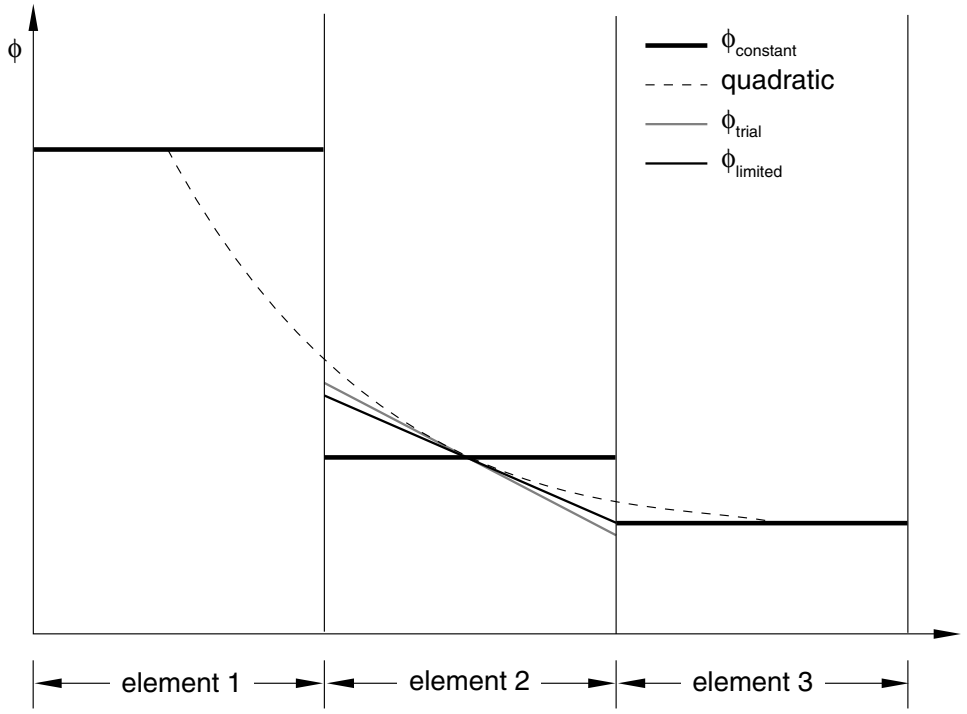


Figure 12.2.3–2 Second-order advection.

1. A quadratic interpolation is constructed from the constant values of ϕ at the integration points of the middle element and in its adjacent elements.
2. A trial linear distribution, ϕ_{trial}, is found by differentiating the quadratic function to find the slope at the integration point of the middle element.
3. The trial linear distribution in the middle element is limited by reducing its slope until its minimum and maximum values are within the range of the original constant values in the adjacent elements. This process is referred to as flux-limiting and is essential to ensure that the advection is monotonic.

Once the flux-limited linear distributions are determined for all elements in the old mesh, these distributions are integrated over each new element. The new value of the variable is found by dividing the value of each integral by the new element volume. (See Figure 12.2.3–3 for a first-order example of this calculation.)

Input File Usage: Use the following option to specify that the second-order advection method should be used to remap element variables:

∗ADAPTIVE MESH CONTROLS, NAME=*name*,
ADVECTION=SECOND ORDER

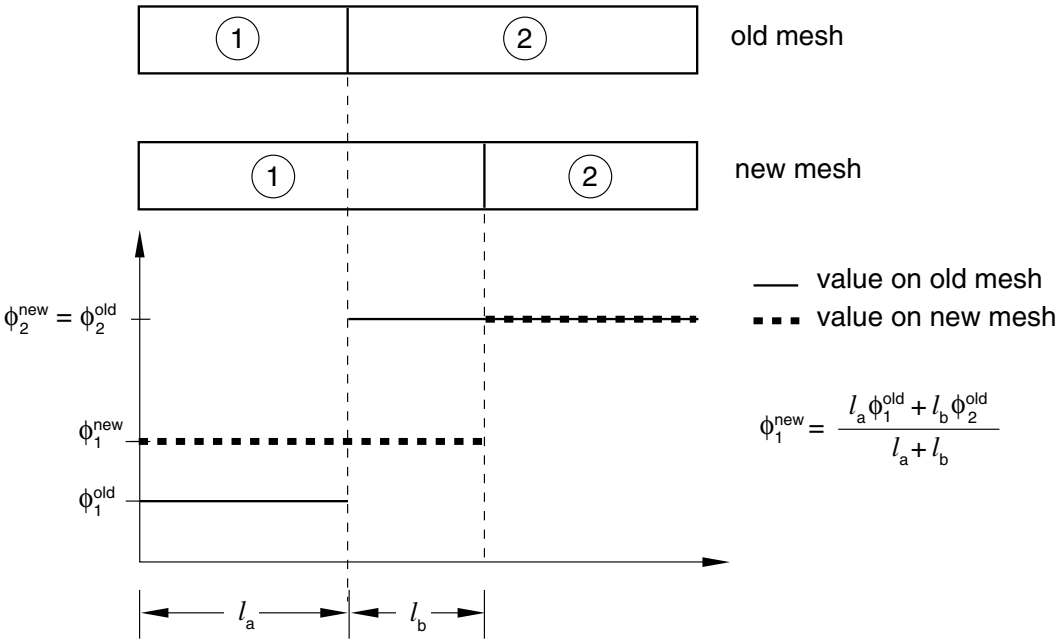


Figure 12.2.3–3 First-order advection.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, toggle on **Second order**

First-order advection

First-order advection is simple and computationally efficient; however, it tends to diffuse sharp gradients over time, especially in transient dynamic analyses or other problems that require fairly frequent adaptive meshing. Therefore, this technique should be used only as a computationally efficient alternative for quasi-static simulations that do not require frequent adaptive meshing.

Figure 12.2.3–3 illustrates the first-order method for a portion of a one-dimensional mesh. An element variable, ϕ, is remapped from the old mesh to the new mesh by first assuming a constant value of the variable for each old element. These values are then integrated over each new element. The new value of the variable is found by dividing the value of each integral by the new element volume.

Input File Usage: Use the following option to specify that the first-order advection method should be used to remap element variables:

```
*ADAPTIVE MESH CONTROLS, NAME=name,
ADVECTION=FIRST ORDER
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, toggle on **First order**

Momentum advection

Nodal velocities are computed on a new mesh by first advecting momentum, then using the mass distribution on the new mesh to calculate the velocity field. Advecting momentum directly ensures that momentum is conserved properly in the adaptive mesh domain during remapping. Two methods are available for advecting momentum: the default element center projection method and the half-index shift method (Benson, 1992). Both methods are applicable for all adaptive mesh applications.

Element center projection method

The element center projection method is the default method used to advect momentum and requires the fewest numerical operations. The element momentum is calculated first for the old mesh based on the mass and velocity of the element nodes. The element momentum is then advected from the old mesh to the new mesh by the same first- or second-order algorithms used for advecting element variables. Finally, the element momentum on the new mesh is assembled at the nodes using a projection. The element center projection method requires the advection of only two or three extra variables in two dimensions or three dimensions, respectively.

Input File Usage: Use the following option to request the most computationally efficient momentum advection method:

```
*ADAPTIVE MESH CONTROLS, NAME=name,
MOMENTUM ADVECTION=ELEMENT CENTER PROJECTION
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Momentum advection: Element center projection**

Half-index shift method

The half-index shift method is computationally more intensive than the element center projection method, but it may result in less wave dispersion for some problems. This method first shifts each of the nodal momentum variables from the nodes surrounding an element to the element center. The shifted momentum variables are then advected from the old mesh to the new mesh by the same first- or second-order algorithms used for advecting element variables, providing momentum variables at the center of the new elements. Finally, the momentum variables at the element centers in the new mesh are shifted back to the nodes. The half-index shift method requires the advection of 8 or 24 extra variables in two or three dimensions, respectively, which can increase the cost of each advection sweep significantly.

Input File Usage: Use the following option to specify that the half-index shift method should be used for momentum advection:

```
*ADAPTIVE MESH CONTROLS, NAME=name,
MOMENTUM ADVECTION=HALF INDEX SHIFT
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Momentum advection: Half-index shift**

Additional references

- Benson, D. J., "Momentum Advection on a Staggered Mesh," Journal of Computational Physics, vol. 100, pp. 143–162, 1992.
- Van Leer, B., "Towards the Ultimate Conservative Difference Scheme III. Upstream-centered Finite-Difference Schemes for Ideal Compressible Flow," Journal of Computational Physics, vol. 23, pp. 263–275, 1977.
- Van Leer, B., "Towards the Ultimate Conservative Difference Scheme IV. A New Approach to Numerical Convection," Journal of Computational Physics, vol. 23, pp. 276–299, 1977.

12.2.4 MODELING TECHNIQUES FOR EULERIAN ADAPTIVE MESH DOMAINS IN Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References

- "ALE adaptive meshing: overview," Section 12.2.1
- "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2
- *ADAPTIVE MESH CONSTRAINT
- "Understanding ALE adaptive meshing," Section 14.6 of the Abaqus/CAE User's Guide

Overview

An Eulerian adaptive mesh domain:

- is used to model material flowing through the mesh; and
- typically has two Eulerian boundary regions, one inflow and one outflow, connected by Lagrangian and/or sliding boundary regions.

The correct combination of mesh constraints and material boundary conditions applied to an Eulerian boundary region depends on whether the region acts as an inflow or an outflow boundary. The region types and mesh constraints assigned to the boundary regions that are connected to the Eulerian boundary regions must be chosen to simulate the correct physical behavior as well.

The adaptive meshing technique in Abaqus/Explicit is robust if the mesh is underconstrained: the modeling techniques presented in this section are intended to provide guidance in properly defining Eulerian models.

ALE adaptive mesh constraints on Eulerian boundaries

ALE adaptive mesh constraints should be applied normal to an Eulerian boundary region; otherwise, the motion of the mesh on the boundary is ambiguous. If no mesh constraints are applied normal to the boundary, Abaqus/Explicit will treat the region as if it were sliding, and the mesh will follow the material normal to the boundary.

Although there are no restrictions on specifying adaptive mesh constraints at nodes on an Eulerian boundary region, the following guidelines should be followed in most cases:

- Mesh constraints should be applied to every node on the Eulerian boundary region, including the corners and edges.
- Mesh constraints should be applied either only normal to the Eulerian boundary region or in every direction. Mesh constraints should not be specified in only the direction tangential to an Eulerian boundary region; such constraints are ambiguous and may result in undesirable motion of the mesh at the boundary.

Loads and boundary conditions on Eulerian boundaries act on the material that instantaneously coincides with the mesh at the surface. When used in combination with spatial adaptive mesh constraints, physically meaningful Eulerian flow conditions can be defined.

Defining inflow Eulerian boundaries

The material flowing into an adaptive mesh domain through an Eulerian boundary will have the same stress and material state as the material in the elements immediately adjacent to the boundary. Therefore, it is important to maintain the stress and material state in those elements at the desired values (which in many cases will be zero, to simulate stress-free material entering the Eulerian domain). To accomplish this goal:

- position the inflow boundary far enough upstream from high solution gradients to ensure that the response at the inflow boundary is smooth, and
- apply sufficient mesh and material constraints at the boundary (as described later in this section).

To be physically meaningful, the size and shape of the inflow boundary region must be maintained. For example, applying sufficient constraints is crucial for steady-state process simulations where the cross-section of the workpiece entering the adaptive mesh domain is known and affects the response downstream. The types of constraints appropriate for an inflow boundary depend on whether the precise location of the inflow boundary region is known or whether it is part of the solution.

Known inflow boundary location

In many problems the area, shape, and position of the inflow boundary are known a priori. For example, in the steady-state analysis of a forward extrusion process, an inflow Eulerian boundary can be used to model the flow of material into the adaptive mesh domain. The size of the inflow boundary is based on the known billet cross-section, and the location of the inflow boundary is fixed because of the confined conditions on the material.

When the area, shape, and location of the inflow boundary are known, both material and mesh constraints should be applied. Figure 12.2.4–1 shows a typical model setup for a two-dimensional forward extrusion problem where either a prescribed mass flow rate or a prescribed uniform pressure is applied to a known inflow boundary. Apply boundary conditions at all nodes on the inflow boundary region to prescribe material constraints in the directions tangential to the boundary surface. Preventing motion of the material tangential to the inflow boundary helps to maintain the stress and material state of the elements adjacent to the Eulerian boundary.

Apply adaptive mesh constraints in the normal direction at all nodes on the inflow boundary. In addition, apply mesh constraints in all tangential directions at the edges and corners surrounding the Eulerian boundary region. These constraints fix the location and size of the cross-sectional area at the inflow boundary.

If a nonuniform boundary condition or load is applied to the material at the inflow boundary or if the initial material state in the elements adjacent to the boundary is nonuniform in the tangential direction, apply tangential mesh constraints to the nodes strictly in the interior of the Eulerian boundary region.

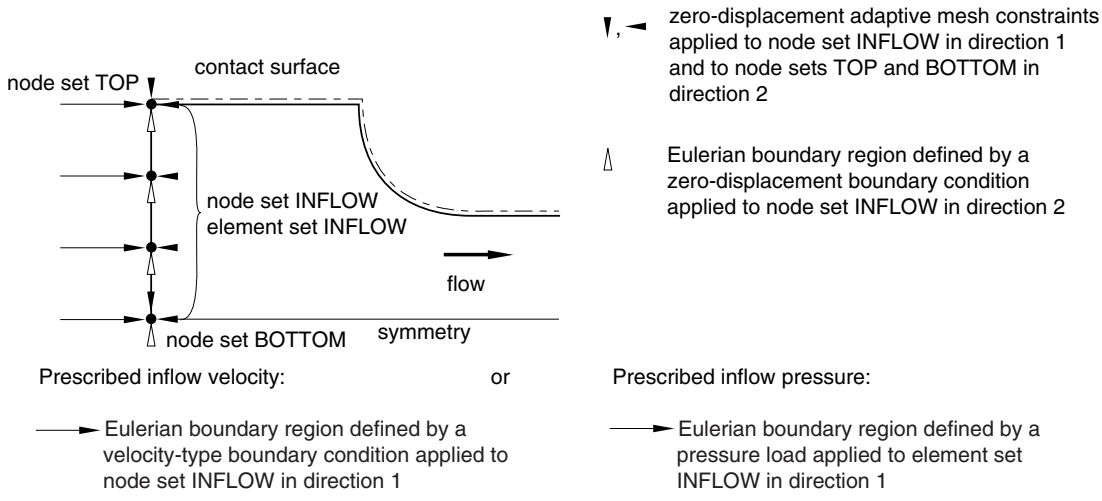


Figure 12.2.4–1 Known inflow boundary.

Although the application of mesh and material constraints tangential to and along the edges and corners of an inflow Eulerian boundary may appear to be redundant, they are actually independent. For example, consider a long billet with a variable cross-section, as shown in Figure 12.2.4–2.

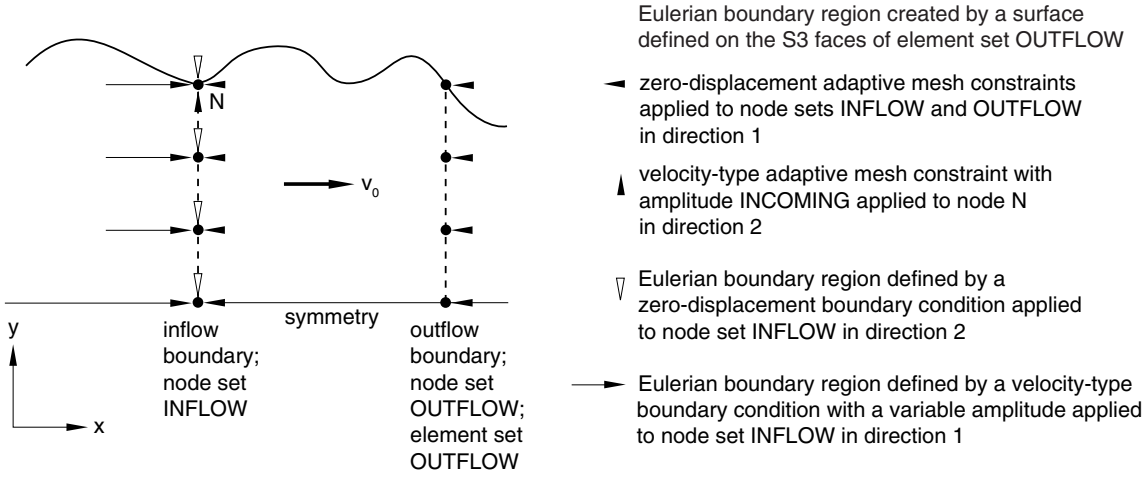


Figure 12.2.4–2 Modeling a billet with a variable cross-section.

The adaptive mesh domain, with its inflow and outflow Eulerian boundary regions, is assumed to represent a portion of the billet along its length. The entire billet moves with a rigid body velocity along

its length (x-direction) so that material flows into one Eulerian boundary and out the other. Boundary conditions are applied to the material at the inflow boundary to maintain this velocity. Mesh constraints are applied normal to the inflow and outflow boundary regions. The mesh constraint applied in the y-direction at node N is used to prescribe the known variable incoming cross-section of the material. The motion of this node does not affect the velocity field of the material entering the domain.

Unknown inflow boundary location

Sometimes, the location of the inflow boundary region is known only approximately; its precise location will be determined from the solution. For these problems, apply adaptive mesh constraints only in the direction normal to the Eulerian boundary region. In the absence of tangential mesh constraints at the edges and corners of the Eulerian boundary region, Abaqus/Explicit will move these edges and corners with the material in the tangential direction but with the mesh constraints in the normal direction. Therefore, material constraints should be applied using multi-point constraints (see "General multi-point constraints," Section 35.2.2) or linear constraint equations (see "Linear constraint equations," Section 35.2.1) to preserve the cross-sectional area of the inflow boundary.

For example, consider a steady-state rolling simulation with multiple rollers in an asymmetric configuration, as shown in Figure 12.2.4–3.

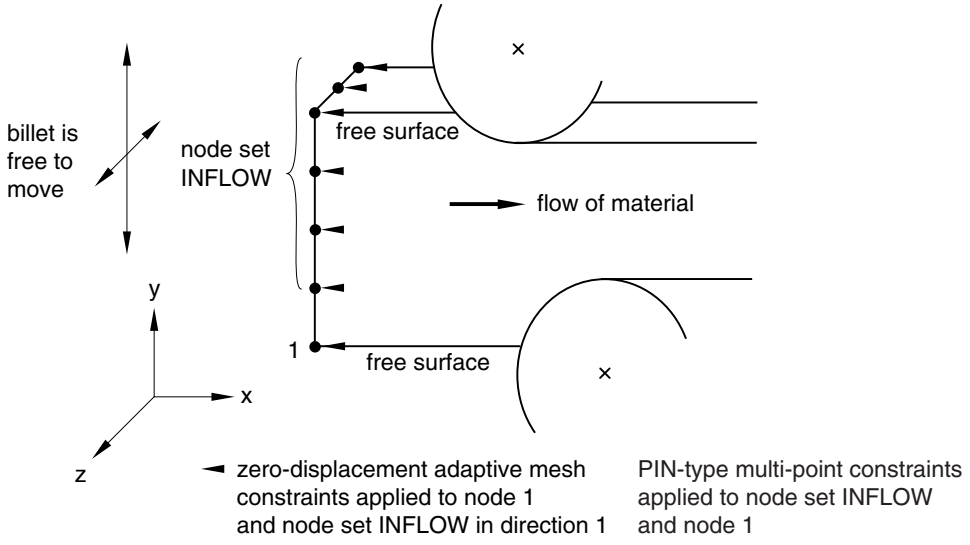


Figure 12.2.4–3 Unknown inflow boundary location.

It may be impractical to extend the analysis domain as far as the guides on the upstream side, but spatially fixing the inflow boundary at an arbitrary position in the y- and z-directions may cause unrealistic stress on the workpiece as it finds an equilibrium position between the rollers. Mesh constraints are applied normal to the Eulerian boundary region to fix the position of the inflow boundary relative to the rollers in the x-direction. Material constraints (applied with a PIN MPC) are used to ensure that material enters

the domain at a uniform velocity and that the cross-section does not rotate. The material constraints will maintain the cross-sectional shape of the section while allowing it to move laterally to the correct equilibrium position. Since tangential mesh constraints are not used, the mesh will follow the material in the directions tangential to the Eulerian boundary region.

Defining outflow Eulerian boundaries

Typically, adaptive mesh constraints should be applied only in the direction normal to the surface on an Eulerian boundary region that acts as an outflow boundary. No tangential mesh constraints should be applied to the edges or corners of an outflow boundary adjacent to a Lagrangian (or sliding) boundary region acting as a free surface. In contrast to inflow boundaries, the cross-section of an outflow boundary adjacent to a free surface is determined by the solution in the domain. At the edge or corner where an Eulerian boundary region meets a Lagrangian or sliding boundary region, Abaqus/Explicit will satisfy the applied mesh constraint normal to the Eulerian boundary region and the inherent mesh constraint normal to the Lagrangian or sliding boundary region simultaneously, thus correctly handling the evolution of the free surface adjacent to the outflow boundary. Figure 12.2.4–4 shows the evolution of an outflow boundary from t_0 to t_1, where material continues to flow through the outflow boundary.

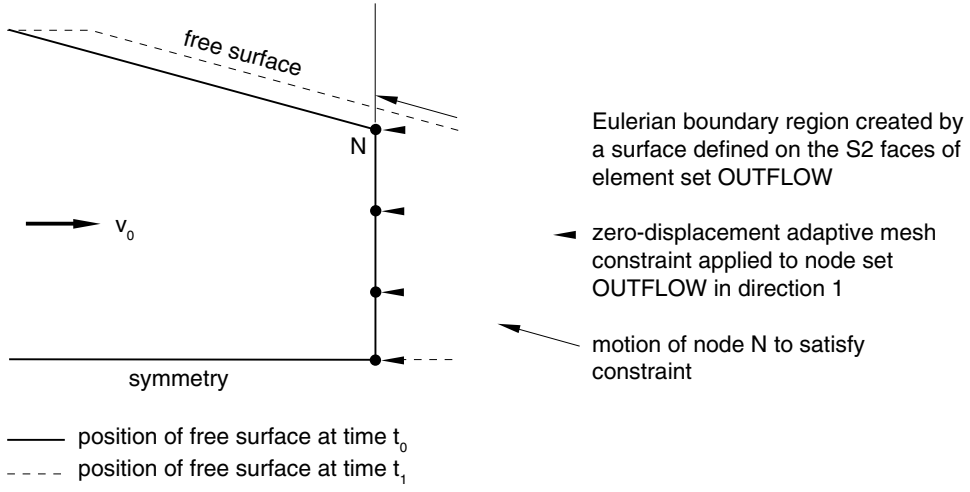


Figure 12.2.4–4 Abaqus/Explicit will respect the free surface at an Eulerian outflow boundary.

The mesh constraint normal to the Eulerian outflow boundary is applied by moving node N along the free surface of the material, so that the outflow boundary "expands" with the material arriving from upstream. Although not shown in the figure, mesh smoothing causes all other nodes on the outflow boundary, with the exception of the node on the symmetry plane, to move up toward node N as the boundary expands.

No special material boundary conditions are required at outflow Eulerian boundaries. Boundary conditions tangential to the outflow boundary are recommended only if they are the same as those defined upstream (e.g., a symmetry plane running along the length of an Eulerian domain). However, to improve

convergence to the steady-state solution in steady-state process simulations, it is often useful to constrain the material velocity to be uniform normal to the outflow boundary using multi-point constraints or linear constraint equations.

Defining Eulerian boundary regions that act as both inflow and outflow boundaries

Although it is rarely appropriate, an Eulerian boundary region can act as both an inflow and an outflow boundary at different times during the same analysis step. Adaptive mesh constraints and material boundary conditions at such a boundary should be chosen to be physically meaningful for both inflow and outflow situations.

For each node on the edges and corners of an Eulerian boundary region that does not have mesh constraints tangential to the boundary surface, Abaqus/Explicit will determine in each adaptive mesh increment whether the boundary at the node is acting as an inflow or an outflow boundary. If an inflow condition is detected, the node will move with the material in the tangential direction but with the mesh constraints in the normal direction. If an outflow condition is detected, the movement of the node will both follow the adjacent Lagrangian boundary region and satisfy the mesh constraint normal to the Eulerian boundary region.

Lagrangian versus sliding boundary regions on Eulerian domains

Many applications using Eulerian adaptive mesh domains, including the simulation of steady-state processes, have a primary direction of material flow and use a control volume approach to model the process zone. These problems usually include two Eulerian boundary regions, representing an inflow boundary and outflow boundary. The remaining surfaces between the Eulerian boundaries can be either Lagrangian or sliding boundary regions. Determining which type of boundary region to use between the two Eulerian boundary regions depends on the type of load or boundary condition that is required:

- Use a sliding boundary region to define boundary conditions or loads that act at a spatial location on a portion of the surface along the length of the control volume. Apply adaptive mesh constraints to fix the mesh spatially in the flow direction (and possibly in the direction transverse to the flow). For example, a distributed pressure can be applied around the circumference of the control volume, as shown in Figure 12.2.4–5. The distributed pressure load is defined using a sliding boundary region. Mesh constraints are applied to fix the boundary region spatially in the flow direction. Similarly, a concentrated load could be applied to a specific spatial location to model the effect of a very sharp body pressing into the workpiece at a known location with a known force.
- Use a sliding boundary region to define boundary conditions or loads that act along the entire length of the Eulerian control volume between the inflow and outflow boundaries and act in a spatial manner transversely to the flow. If the load acts on only a portion of the surface in the transverse direction, it may be necessary to apply mesh constraints in the direction transverse to the flow. For example, a boundary condition that acts as a knife edge along the length of the domain is shown in Figure 12.2.4–6. Mesh constraints are applied in the transverse direction (and, if the line of application is curved, along the line) to keep the boundary condition fixed spatially.
- Use a Lagrangian boundary region (default) to define boundary conditions or loads that act along the entire length of the surface of the Eulerian control volume between the inflow and

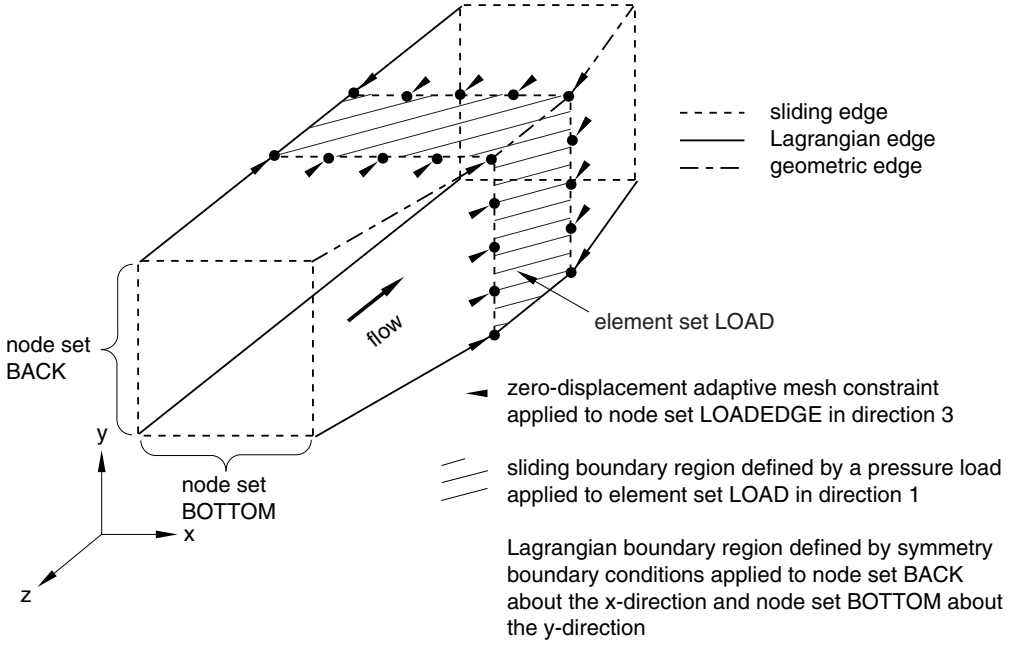


Figure 12.2.4–5 Applying a spatial pressure load to a portion of the surface along the length of an Eulerian control volume.

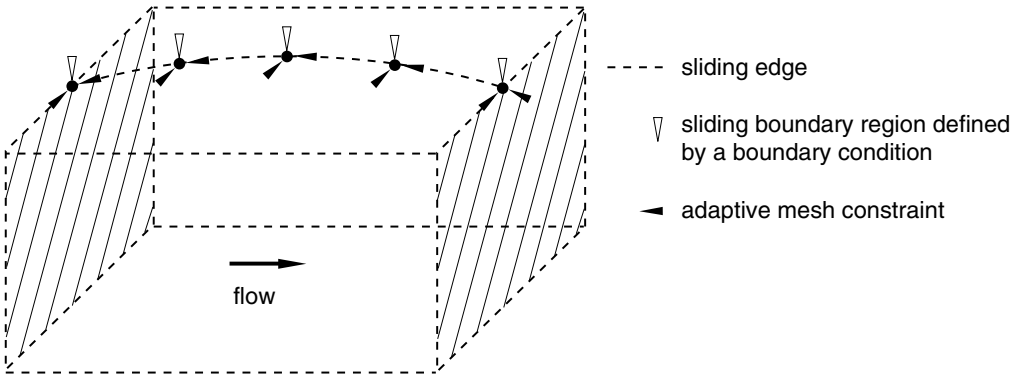


Figure 12.2.4–6 Applying a boundary condition along the entire length of the Eulerian control volume.

outflow boundary and act in a Lagrangian manner transversely to the flow. In three dimensions, symmetry conditions should typically act in a Lagrangian manner transverse to the flow direction. In many cases geometric edges will prevent material from flowing off the symmetry plane and

onto the free surface. However, since geometric edges can be deactivated as surfaces flatten, Lagrangian boundary regions should be used to define the symmetry planes for these types of problems. In Figure 12.2.4–5 quarter-symmetry is assumed, and the symmetry planes are defined using Lagrangian boundary regions. The resulting Lagrangian edges that run from one Eulerian boundary to the other separate the symmetry planes from the free surface.

- Boundary conditions or loads that act on only a specific portion of the material between the inflow and outflow boundaries cannot usually be modeled for problems utilizing Eulerian control volumes. Since the mesh underneath the load or boundary condition must follow the material, it will eventually be restricted by the Eulerian boundary. This treatment of loads and boundary conditions is not usually consistent with a steady-state model and should not arise in practical simulations using Eulerian adaptive mesh domains.

12.2.5 OUTPUT AND DIAGNOSTICS FOR ALE ADAPTIVE MESHING IN Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References

- "Output to the output database," Section 4.1.3
- "ALE adaptive meshing: overview," Section 12.2.1
- "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2
- *ADAPTIVE MESH
- *ADAPTIVE MESH CONTROLS
- *DIAGNOSTICS
- *TRACER PARTICLE
- Chapter 78, "Using display groups to display subsets of your model," of the Abaqus/CAE User's Guide

Overview

Output for ALE adaptive meshing:

- can be used to verify the automatic splitting of user-defined domains, the formation of Lagrangian edges and corners, the formation of geometric edges and corners, and the determination of nonadaptive nodes;
- must be interpreted carefully, since the values of output variables at specific locations in the mesh are no longer linked to values at particular material points;
- can include the definition of tracer particles, which follow material points and allow you to examine the trajectory of those points and plot material time histories of all element and nodal variables at those points; and
- can include diagnostic information on the efficiency of adaptive meshing and the accuracy of advection.

Verifying the model

Output that can be used to verify adaptive meshing models is available in the data (`.dat`) file and in the output database (`.odb`) (see "Output," Section 4.1.1, for details on these files).

Element sets

When user-defined adaptive mesh domains are split by Abaqus/Explicit, the elements that compose the new subdivided domains are printed to the data (`.dat`) file.

New element sets are created and written to the output database (`.odb`) for all adaptive mesh domains. The name of the element set created for each domain is the user-defined name, plus the

number of the subdivision (1 if no subdivisions were created), plus the step number. For example, if the user-defined adaptive mesh domain specified for the element set *domain_name* spanned three disjoint parts, Abaqus/Explicit would subdivide the user-defined domain into three domains and create three element sets in the output database (**.odb**) for the first step: *domain_name*-1-1, *domain_name*-2-1, and *domain_name*-3-1.

Abaqus/CAE can be used to verify the creation of the subdivided domains.

Edges and nonadaptive nodes

Abaqus/Explicit automatically forms Lagrangian edges and corners and identifies nonadaptive nodes based on the topology of the adaptive mesh domains, connections to nonadaptive domains, and user-specified boundary regions. Furthermore, geometric edges and corners are formed automatically based on the initial geometry and the value of the initial feature angle. See "Defining ALE adaptive mesh domains in Abaqus/Explicit," Section 12.2.2. Lagrangian edges, geometric edges and corners, and nonadaptive nodes (including Lagrangian corners) are output to the data (**.dat**) file for each adaptive mesh domain. This information can be obtained by requesting a history definition summary printout to the data file (see "Model and history definition summaries" in "Output," Section 4.1.1) or by monitoring the progress of the adaptive meshing (see "Monitoring the progress of ALE adaptive meshing" below).

In addition, up to three node sets are created in the output database (**.odb**) for each adaptive mesh domain in each step. The names of the node sets are created by concatenating the following information:

- the domain element set name;
- the number of the subdivision (1 if no subdivisions were created);
- the letters LE for Lagrangian edge, GE for geometric edge or corner, or NA for nonadaptive nodes (including Lagrangian corners); and
- the step number.

For example, if a user-defined three-dimensional adaptive mesh domain specified for element set *domain_name* is subdivided automatically into two adaptive mesh domains, Abaqus/Explicit will generate up to six node sets in the output database for the first step: *domain_name*-1-LE-1, *domain_name*-1-GE-1, *domain_name*-1-NA-1, *domain_name*-2-LE-1, *domain_name*-2-GE-1, and *domain_name*-2-NA-1.

Since boundary regions are separated by corners, not edges, in two dimensions, node sets will not be created for Lagrangian edges in two-dimensional adaptive mesh domains. The Lagrangian corners are included in the nonadaptive (NA) node set, as for three-dimensional domains.

Abaqus/CAE can be used to verify the creation of Lagrangian edges and corners, geometric edges and corners, and nonadaptive nodes.

Interpreting results

When adaptive meshing is not performed, the finite element mesh follows the material, which enables a straightforward interpretation of analysis results. You can visualize deformation and material motion by studying the motion of the mesh. Each nodal and element output variable corresponds to a specific material location, because the mesh is fixed to the same material point throughout time.

Once adaptive meshing takes place, the locations of mesh and material points deviate, and analysis results must be interpreted accordingly. The motion of the mesh on the interior of an adaptive mesh domain represents the composite effects of the material motion and adaptive meshing. The motion of the mesh and the motion of the material on Lagrangian and sliding boundary regions is identical in the direction normal to the boundary but not in the direction tangential to it.

Nodal variables

When adaptive meshing is performed, a material point that is coincident with a node at the beginning of the step may not remain coincident with that node throughout the step. Values of displacement and current coordinates represent the motion of the node, not necessarily the motion of the material. All other nodal variables—including velocity, acceleration, and reaction forces—represent the value of the variable for the material particle at the current location of the node. Contour or vector plots of these variables will show their correct spatial distribution and are, therefore, meaningful. However, time histories of nodal variables for nodes that undergo adaptive meshing are generally not meaningful. In steady-state problems, though, a velocity or acceleration time history based at a fixed spatial location rather than at a specific material point may be useful.

Element variables

Similarly, when adaptive meshing is performed, a material particle that is coincident with an element integration point at the beginning of a step may not remain so throughout the step. Therefore, element integration point variables do not necessarily represent values at the same material point throughout the step. Contour or vector plots of element integration point variables are meaningful for the same reasons described for nodal variables. However, time histories are based at the spatial location of the element integration point and not at a specific material point.

Whole element variables have a similar interpretation.

Tracking nodal or element variables at material points

Tracer particles can be defined to track material points in an adaptive mesh domain. These particles can also be used to obtain time histories of nodal or element integration point variables that correspond to the time variation of the variable at a specific material point. Tracer particles are defined as described below (see “Output to the output database,” Section 4.1.3, for more information). Node and element variable output can be requested for tracer particle sets to examine the trajectory of material particles or to obtain material time histories. Output for tracer particles can be written only to the output database (**`.odb`**).

Using tracer particles in Lagrangian domains

In most adaptive meshing simulations using Lagrangian domains, the nodes and elements in the domain correspond neither to a specific spatial location nor to a specific material point or volume. Thus, time histories of variables at nodes and at element integration points are often physically meaningless in a Lagrangian adaptive mesh domain. Tracer particles should be defined to view time history information. Tracer particles can also be used to visualize the motion of the material.

The initial location of a tracer particle is defined to be coincident with a node, termed the parent node. Tracer particles are defined in sets by defining multiple parent nodes or node sets. You indicate the nodes whose current locations correspond to the initial location of the tracer particles and assign a name to the tracer particle set to identify it for use in output requests. Tracer particles are released from their parent nodes repeatedly at specified intervals during the step in which they are defined. The particles follow material points for the remainder of that step and in all subsequent steps.

Tracer particles are typically defined only on adaptive mesh domains, although they can be defined on nodes connected to any low-order solid element in the model. For analyses in which adaptive meshing is not performed until later steps, tracer particles can be defined on nonadaptive domains at the beginning of an analysis and will be tracked continuously as the domain becomes adaptive. Similarly, tracer particles will be tracked from domain to domain if adaptive mesh domain topologies change from step to step.

Input File Usage: Use the following option to define a tracer particle set:

∗TRACER PARTICLE, TRACER SET=*tracer_set_name*
list of tracer particle parent nodes

Abaqus/CAE Usage: Tracer particles are not supported in Abaqus/CAE.

Using tracer particles in Eulerian domains

Time histories at nodes and element integration points in an Eulerian domain may have physical meaning at points where spatial adaptive mesh constraints are applied. For example, the time variation of equivalent plastic strain in elements along an outflow Eulerian boundary acts as a spatial time history of that variable and can be used to evaluate whether the process has reached a steady-state solution.

Tracer particles can be defined to evaluate the material time history of variables at a material point as it flows through the Eulerian domain. Tracer particles can also be used to evaluate the trajectory and path of material points as they pass through the domain.

Tracer particles can be assigned to any parent node in an Eulerian adaptive mesh domain. If a tracer particle reaches an outflow boundary and material continues to flow out, the tracer particle will no longer be tracked and all output history variables associated with the tracer particle will be zero after deactivation.

When material flow through the mesh domain is significant, sets of tracer particles can be released from the current locations of the parent nodes at multiple times during the step. Each release of tracer particles is referred to as particle birth. After particle birth the tracer particles follow the motion of the material regardless of the motion of the mesh. You can indicate the number of particle birth stages in a step. These stages will be evenly spaced throughout the time period of the step.

For example, a tracer particle set can be defined such that all nodes along an inflow Eulerian boundary are parent nodes. Multiple birth stages can be specified so that a set of tracer particles is released from the mesh at the inflow boundary periodically during the step. If enough birth stages are defined, the domain will eventually be spanned with tracer particles as material flows from the inflow boundary to the outflow boundary.

Input File Usage: Use the following option to define a tracer particle set with multiple birth stages:

∗TRACER PARTICLE, TRACER SET=*tracer_set_name*,
PARTICLE BIRTH STAGES=*n*
list of tracer particle parent nodes

Abaqus/CAE Usage: Tracer particles are not supported in Abaqus/CAE.

Monitoring the progress of ALE adaptive meshing

Diagnostic information can be written to the message (**.msg**) file to track the efficiency and accuracy of adaptive meshing. You can select the level of diagnostic output that is written.

Obtaining a summary at the end of a step

By default, a summary of adaptive meshing information for each adaptive mesh domain will be written to the message (**.msg**) file at the end of each step. This summary information includes:

- the average percentage of nodes moved,
- the maximum percentage of nodes moved,
- the minimum percentage of nodes moved, and
- the average number of advection sweeps.

Each value is calculated for a single adaptive mesh domain over all adaptive mesh increments. The cost of advection is approximately proportional to the percentage of nodes moved, since variables are not advected for elements that have not been relocated during adaptive meshing.

Input File Usage: Use the following option to request a summary for each adaptive mesh domain at the end of each step:

∗DIAGNOSTICS, ADAPTIVE MESH=STEP SUMMARY

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Obtaining a summary for every ALE adaptive mesh increment

In addition to the step summary information, the following diagnostics can be obtained for each adaptive mesh domain at every adaptive mesh increment:

- the percentage of nodes moved, and
- the number of advection sweeps.

Input File Usage: Use the following option to obtain summary information at the end of the step and at every adaptive mesh increment:

∗DIAGNOSTICS, ADAPTIVE MESH=SUMMARY

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Obtaining details of advection accuracy for every ALE adaptive mesh increment

The following detailed diagnostic information can also be written to the message (**.msg**) file to track the accuracy of the advection:

- mass and momentum before and after advection, and
- percentage volume change.

Input File Usage: Use the following option to request the most detailed diagnostics, which include advection accuracy measures and summary information for each adaptive mesh domain, reported at every adaptive mesh increment:

*DIAGNOSTICS, ADAPTIVE MESH=DETAIL

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

Suppressing ALE adaptive mesh diagnostics

You can suppress output of all adaptive mesh diagnostic information.

Input File Usage: *DIAGNOSTICS, ADAPTIVE MESH=OFF

Abaqus/CAE Usage: Adaptive mesh diagnostics are not supported in Abaqus/CAE.

12.2.6 DEFINING ALE ADAPTIVE MESH DOMAINS IN Abaqus/Standard

Products: Abaqus/Standard Abaqus/CAE

References

- "ALE adaptive meshing: overview," Section 12.2.1
- "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7
- *ADAPTIVE MESH
- *ADAPTIVE MESH CONSTRAINT
- *ADAPTIVE MESH CONTROLS
- "Customizing ALE adaptive meshing," Section 14.14 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

ALE adaptive meshing in Abaqus/Standard:

- maintains a topologically similar mesh;
- can be used to solve Lagrangian problems (in which no material leaves the mesh) and to model effects of ablation, or wear (in which material is eroded at the boundary);
- can be used in static stress/displacement analysis, steady-state transport analysis, coupled pore fluid flow and stress analysis, and coupled temperature-displacement analysis;
- can be used only in geometrically nonlinear general analysis steps; and
- is available only for acoustic elements and a subset of the solid elements.

Defining an ALE adaptive mesh domain

You can apply ALE adaptive mesh smoothing to an entire model or to individual parts of the model as a step-dependent feature. Adaptive meshing for solid elements in Abaqus/Standard uses a subset of the adaptive meshing functionality available in Abaqus/Explicit.

You must specify the portion of the original mesh that will be subject to adaptive meshing.

Input File Usage: *ADAPTIVE MESH, ELSET=*name*

Multiple adaptive mesh domains can be defined in a step by reusing the *ADAPTIVE MESH option, but each element set must refer to a unique set of elements.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the ALE adaptive mesh domain below**, and click **Edit** to select the region

Only one adaptive mesh domain can be defined in Abaqus/CAE for any particular step.

Modifying an ALE adaptive mesh domain

By default, all adaptive mesh domains defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh domains in effect for a given step relative to the preexisting adaptive mesh domains. At each new step the existing adaptive mesh domains can be modified and additional adaptive mesh domains can be specified (except in Abaqus/CAE, where only one adaptive mesh domain can be in effect for a given step).

Input File Usage: Use either of the following options to modify an existing adaptive mesh domain or to specify an additional adaptive mesh domain:

∗ADAPTIVE MESH, ELSET=*name*
∗ADAPTIVE MESH, ELSET=*name*, OP=MOD

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**

Removing an ALE adaptive mesh domain

If you choose to remove any adaptive mesh domain in a step, no adaptive mesh domains will be propagated from the previous step. Therefore, all adaptive mesh domains that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh domains and to specify new adaptive mesh domains:

∗ADAPTIVE MESH, ELSET=*name*, OP=NEW

If the OP=NEW parameter is used on any ∗ADAPTIVE MESH option within a step, it must be used on all ∗ADAPTIVE MESH options in the step.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **No ALE adaptive mesh domain for this step**

Splitting ALE adaptive mesh domains

Abaqus/Standard may subdivide each adaptive mesh domain that you specify such that

- all elements in an adaptive domain refer to one element property definition; and
- all elements in an adaptive domain are of similar type (such as hybrid elements with linear pressure).

If Abaqus/Standard subdivides the adaptive mesh domains that you specified, each of the adaptive mesh domain subdivisions will have a new name, which will be used for output and diagnostic purposes. The new names will be formed by concatenating the name of the user-specified element set, a number identifying the subdivision, and the step number. Each of the subdivisions will be further examined to ensure that all the elements in a subdivision are subjected to the same body forces. You may be asked to modify the definition of the adaptive mesh domain to satisfy this requirement.

ALE adaptive mesh regions

Each adaptive mesh domain has an interior region and a boundary region. The boundary region may include distinct kinks that take the form of geometric edges or corners. The nodes on the boundary region

are, therefore, further separated into free surface nodes, edge nodes, and constrained nodes. Different updating rules are applied to nodes in these different regions. These regions are created automatically by Abaqus/Standard. You can control the detection of the geometric features. In addition, mesh constraints can be applied to any node in the adaptive mesh domain.

Since acoustic elements do not have displacement degrees of freedom, their treatment for adaptive meshing includes some additional considerations. The acoustic adaptive domain must be connected to the structural domain using a surface-based tie constraint with the slave surface defined on the acoustic domain. Thus, an acoustic adaptive domain has an additional boundary region that is connected to the structural domain. These slave surface nodes are updated based on the displaced configuration of the master surface nodes on the structural domain, without permitting relative sliding between the surfaces. The displacements of the master surface defined on the structural domain, together with nonzero adaptive mesh constraints, serve as the forcing function that drives adaptive mesh smoothing of an acoustic adaptive domain. The mesh smoothing algorithm will produce no changes in the acoustic adaptive domain if these displacements are zero.

Options for controlling the mesh smoothing algorithm are described in "ALE adaptive meshing and remapping in Abaqus/Standard," Section 12.2.7.

ALE adaptive mesh interior regions

Nodes in the interior region are defined as nodes that are surrounded entirely by elements in the adaptive mesh domain. By default, the new position of an interior node is computed from the positions of the adjacent nodes that are connected through element edges to the node in question. These nodes can move in any direction.

To control the displacement of these nodes, you can apply an adaptive mesh constraint in any direction.

ALE adaptive mesh boundary regions

The boundary region is that part of the surface of the adaptive mesh domain that is not constrained to other elements in the mesh. The nodes on the boundary region are further separated into surface nodes, edge nodes, corner nodes, and constrained nodes.

Surface, edge, and corner nodes

Surface nodes are defined as nodes at which the surrounding surface facets have the same normal vector within a user-defined angle. These nodes are constrained against movement in the normal direction, but sliding in any tangential direction is permitted. The new position of a surface node is computed from the positions of the adjacent nodes that are connected through the edges of the surface facets to the node in question.

Edge nodes are nodes in a three-dimensional model at which the surrounding surface facets have two different normals and where the vectors along two of the surface edges are colinear. Nodes on an edge can slide only along the edge. The new position of an edge node is computed from the positions of the two adjacent nodes along the edge.

Corner nodes are nodes at which all the surrounding surface facet normals are different. These nodes are constrained against all mesh smoothing movement.

You can control the displacement of these node types on the boundary region by applying an adaptive mesh constraint in any direction.

Constrained nodes in an acoustic adaptive domain

A surface-based tie constraint can be used to connect two acoustic surfaces together. When both the master and slave nodes of the tie constraint belong to the same adaptive mesh domain, the master surface nodes are updated according to the rules for surface, edge, and corner nodes. An adaptive mesh constraint can be applied at master surface nodes. Slave nodes are updated by applying a tie constraint. Adaptive mesh constraints cannot be applied at slave surface nodes.

Mesh smoothing is not applied to these nodes when the master and slave nodes belong to different acoustic adaptive mesh domains.

Constrained nodes in a solid adaptive domain

Mesh smoothing is not applied to nodes that are involved in multi-point constraints (see "General multi-point constraints," Section 35.2.2), equations (see "Linear constraint equations," Section 35.2.1), or kinematic coupling constraints ("Coupling constraints," Section 35.3.2).

Geometric features

The classification of boundary region nodes as surface, edge, and corner nodes is performed based on the identification of geometric features in the mesh's configuration at the start of a step where adaptive mesh domains are defined and is updated as the analysis proceeds and the configuration changes. You can define the criteria that Abaqus/Standard uses in classifying geometric features through adaptive mesh controls.

Controlling the detection of geometric edges and corners

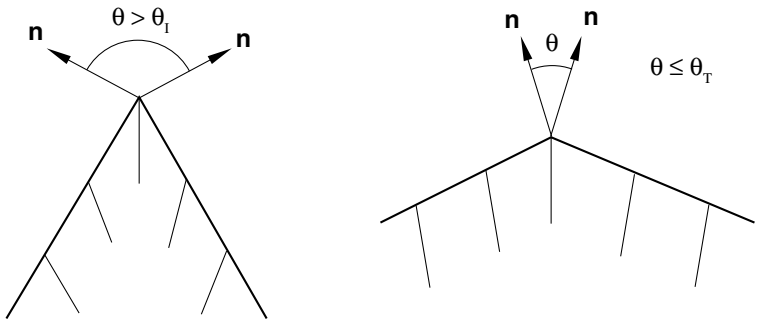
Geometric features are identified initially as edges on boundary regions where the angle between the normals on adjacent element faces is greater than the initial geometric feature angle, θ_I ($0^\circ \leq \theta_I \leq 180^\circ$), as shown in Figure 12.2.6–1. The default value for the initial geometric feature angle is $\theta_I = 30^\circ$. Setting $\theta_I = 180^\circ$ will ensure that no geometric edges or corners are formed on the boundary of the adaptive mesh domain. You can define adaptive mesh controls to change the value of the angle that will be used to recognize geometric features.

Input File Usage: *ADAPTIVE MESH CONTROLS, NAME=*name*,
INITIAL FEATURE ANGLE=θ_I

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**:
Name: *name*, **Initial feature angle:** θ_I

Controlling the activation and deactivation of geometric edges and corners

Abaqus/Standard allows geometric features, and consequently the updating rules applied at a node, to change during the analysis. For example, nodes are constrained to lie along a discrete geometric edge unless the angle forming the geometric edge becomes less than the transition geometric feature angle, θ_T



Initial mesh with a geometric feature: no mesh flow is permitted past the corner.

The geometric feature is deactivated during simulation.

Figure 12.2.6–1 Detection and deactivation of geometric features.

($0^\circ \leq \theta_T \leq 180^\circ$). The default value for the transition feature angle is $\theta_T = 30^\circ$. If the angle across the geometric edge becomes less than θ_T, the boundary surface is considered to be flattened sufficiently for the feature to be deactivated, and the mesh is allowed to slide freely on the surface. Geometric corners are allowed to flatten in a similar fashion. In addition, surfaces that are initially flat may develop edges or corners during the simulation. This logic allows great flexibility in mesh adaptation while preserving geometric features in the model.

Setting $\theta_T = 0^\circ$ will ensure that no geometric edges or corners are ever deactivated. You can change the transition feature angle using adaptive mesh controls.

Abaqus/Standard will issue a warning message when geometric features are activated or deactivated.

Input File Usage: *ADAPTIVE MESH CONTROLS, NAME=*name*,
TRANSITION FEATURE ANGLE=θ_T

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**:
Name: *name*, **Transition feature angle:** θ_T

Mesh constraints

In most adaptive mesh problems the motion of nodes in the mesh is determined by the mesh smoothing algorithm, with constraints imposed by the domain boundary and the boundary region edges. However, there may be cases when you will want to define the motion of the nodes explicitly. You may also wish to keep certain nodes fixed, to move nodes in a particular direction, or to force certain nodes to move with the material.

Adaptive mesh constraints give you the flexibility to define the motion of the node explicitly.

Input File Usage: *ADAPTIVE MESH CONSTRAINT

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Create**

Applying spatial mesh constraints

Spatial mesh constraints are applied to define the motion of the nodes explicitly. Spatial mesh constraints allow full control over the mesh movement and can be applied to any node except those that have Lagrangian mesh constraints applied to them.

You can also prescribe the spatial mesh constraints via user subroutine **UMESHMOTION**. The user subroutine allows you to let the spatial mesh constraints depend on available nodal or material point information.

Input File Usage: Use the following option to define the mesh constraints explicitly:

*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=SPATIAL, TYPE=DISPLACEMENT or VELOCITY

Use the following option to define the mesh constraints in user subroutine **UMESHMOTION**:

*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=SPATIAL, TYPE=DISPLACEMENT or VELOCITY, USER

Abaqus/CAE Usage: To define the mesh constraints explicitly:

Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**

To define the mesh motion in user subroutine **UMESHMOTION**:

Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: User-defined**

Defining mesh constraints that vary with time

The prescribed magnitude of a nonzero mesh constraint can vary with time during a step according to an amplitude definition (see "Amplitude curves," Section 34.1.2).

Input File Usage: Use both of the following options:

*AMPLITUDE, NAME=*name*
*ADAPTIVE MESH CONSTRAINT, AMPLITUDE=*name*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Independent of underlying material**: **Amplitude:** *amplitude*

Applying spatial mesh constraints in local directions

Mesh constraints are applied in local directions if a transformed coordinate system is used at a node ("Transformed coordinate systems," Section 2.1.5); otherwise, they are applied in global directions.

Applying Lagrangian mesh constraints

Lagrangian mesh constraints on a node are used to indicate that mesh smoothing should not be applied; i.e., the node must follow the material.

Input File Usage: *ADAPTIVE MESH CONSTRAINT,
CONSTRAINT TYPE=LAGRANGIAN

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Create**: **Types for selected step: Displacement/Rotation** or **Velocity/Angular velocity**: select region: **Motion: Follow underlying material**

Spatial mesh constraint considerations

When you decide on the type of spatial adaptive mesh constraint, (displacement, velocity, or specified with a user subroutine), you should consider the guidelines below.

Choosing between displacement and velocity adaptive mesh constraints

Displacement and velocity mesh constraints differ in their application. Displacement constraints define a node's displacement relative to its original coordinates, while velocity constraints define a node's velocity relative to the motion of the material. You will use a displacement constraint to control a node's motion to a specific coordinate location, while you will use a velocity constraint to control a node's motion relative to the Lagrangian motion. Therefore, a constant velocity adaptive mesh constraint does not in general lead to a constant velocity of the node relative to it's original coordinates.

Applying spatial adaptive mesh constraints to model material ablation

Your spatial mesh constraint is applied without regard to the current material displacement at the node. This behavior allows you to prescribe mesh motion that differs from the current material displacement at the free surface of the adaptive mesh domain, effectively eroding, or adding, material at the boundary. Using adaptive mesh constraints this way is an effective technique for modeling wear or ablation processes. As described above, in common ablation modeling cases you will use the velocity form of the constraint. In addition, for general boundary shapes the most effective interface for ablation is user subroutine **`UMESHMOTION`**, where you can apply spatial mesh constraints to the nodes on the free surface in general ways according to solution-dependent variables, if needed. The user subroutine interface provides a local coordinate system that is normal to the free surface at the surface node, enabling you to describe mesh motions in this local system.

Modifying ALE adaptive mesh constraints

By default, all adaptive mesh constraints defined in the previous analysis step remain unchanged in the subsequent step. You define the adaptive mesh constraints in effect for a given step relative to the preexisting adaptive mesh constraints. At each new step the existing adaptive mesh constraints can be modified and additional adaptive mesh constraints can be specified.

Input File Usage: Use either of the following options to modify an existing adaptive mesh constraint or to specify an additional adaptive mesh constraint:

*ADAPTIVE MESH CONSTRAINT,
*ADAPTIVE MESH CONSTRAINT, OP=MOD

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Manager**: select the desired step and mesh constraint: **Edit**

Removing ALE adaptive mesh constraints

If you choose to remove any adaptive mesh constraint in a step, no adaptive mesh constraints will be propagated from the previous step. Therefore, all adaptive mesh constraints that are in effect during this step must be respecified.

Input File Usage: Use the following option to remove all previously defined adaptive mesh constraints and to specify new adaptive mesh constraints:

*ADAPTIVE MESH CONSTRAINT, OP=NEW

If the OP=NEW parameter is used on any *ADAPTIVE MESH CONSTRAINT option within a step, it must be used on all *ADAPTIVE MESH CONSTRAINT options in the step.

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Constraint→Manager**: select the desired step and mesh constraint: **Deactivate**

Contact

When surfaces are defined for large-sliding contact, adaptive meshing may relocate the nodes on the surfaces. If the bodies in contact are sliding or deforming considerably, you may want to use Lagrangian mesh constraints on the boundary of the surfaces to prevent the surfaces from sliding from their intended place.

For small-sliding contact Abaqus/Standard assumes that the reference configuration does not change significantly. If the reference configuration does not change significantly, the amount of adaptive meshing on these surfaces should be small and the contact quantities computed based on the reference configuration should continue to remain valid (Abaqus/Standard updates the tangent planes if nodes change positions). Hence, Abaqus/Standard will allow the nodes on the contact surface to move as needed by the mesh smoothing. You should apply Lagrangian mesh constraints in cases where nodes are intended to remain nonadaptive.

Initial conditions

Initial temperatures and field variables can be defined on any region subjected to adaptive mesh smoothing. However, these variables will not be remapped from the original to the updated configuration.

Loads

For elements with displacement degrees of freedom, no restrictions are made to loads applied to adaptive mesh domains. In cases where loads are intended to follow the material motion, Lagrangian mesh constraints must be applied to the nodes on the boundary of the surface on which distributed loads are applied to prevent the surface from sliding. This will allow adaptive meshing to occur inside the surface while maintaining the location of the distributed load.

All the nodes on which concentrated loads are applied become nonadaptive.

The loads that can be applied to an acoustic domain are described in "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1. These loads cannot be applied in procedures in which mesh smoothing can be performed.

Boundary conditions

Special consideration is given to nodes on which boundary conditions are applied. No adaptive meshing is done in the direction in which the boundary condition is applied, but adaptive meshing is carried out in other directions. When a boundary condition is removed (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) in a step, the same restriction applies since Abaqus/Standard will ramp off the contribution of the boundary condition over the duration of the step.

The boundary conditions that can be applied to an acoustic domain are described in "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1. These boundary conditions cannot be applied in any analysis procedure in which mesh smoothing can be performed.

Predefined fields

There are no restrictions on applying prescribed temperatures or field variables in an adaptive mesh domain, but these nodal values are not remapped when adaptive meshing is performed. Therefore, predefined fields that are not constant may not be meaningful in an adaptive mesh domain.

Material options

For elements with displacement degrees of freedom all material models that are isotropic and homogeneous can be used in an adaptive domain. Material options that have anisotropic behavior such as anisotropic materials (see "Defining fully anisotropic elasticity" in "Linear elastic behavior," Section 22.2.1), jointed material models (see "Jointed material model," Section 23.5.1), and concrete material models (see "Concrete smeared cracking," Section 23.6.1) cannot be used in an adaptive mesh domain.

For acoustic elements the relevant material models are described in "Acoustic, shock, and coupled acoustic-structural analysis," Section 6.10.1. Mesh smoothing assumes that the geometric changes in the acoustic domain do not lead to changes in material properties, such as fluid density.

Elements

Adaptive mesh domains can be defined for all acoustic first-order and second-order planar, axisymmetric, and three-dimensional elements in Abaqus/Standard and for a limited number of other elements. Table 12.2.6–1 provides a list of supported elements.

Table 12.2.6–1 Elements supported for adaptive meshing.

AC1D2, AC1D3, AC2D3, AC2D4, AC2D6, AC2D8, AC3D4, AC3D6, AC3D8, AC3D10, AC3D15, AC3D20, ACAX3, ACAX4, ACAX6, ACAX8
CPS4, CPS4T, CPS3
CPE4, CPE4H, CPE4T, CPE4HT, CPE4P, CPE4PH, CPE3, CPE3H
CAX4, CAX4H, CAX4T, CAX4HT, CAX4P, CAX4PH, CAX3, CAX3H
C3D8, C3D8R, C3D8H, C3D8RH, C3D8T, C3D8HT, C3D8RT, C3D8RHT, C3D8P, C3D8PH, C3D8RP, C3D8RPH

Procedures

Adaptive meshing can be used only in geometrically nonlinear general steps that invoke one of the following procedures:

- Static stress/displacement analysis (“Static stress analysis procedures: overview,” Section 6.2.1)
- Steady-state transport analysis (“Steady-state transport analysis,” Section 6.4.1)
- Coupled temperature-displacement analysis (“Fully coupled thermal-stress analysis,” Section 6.5.3)
- Coupled pore fluid flow and stress analysis (“Coupled pore fluid diffusion and stress analysis,” Section 6.8.1)

Acoustic elements will typically undergo adaptive meshing during static procedures and then participate in subsequent acoustic procedures in their updated configuration.

Limitations

- Elements within the adaptive domain cannot be removed or added (“Element and contact pair removal and reactivation,” Section 11.2.1).
- Deformable elements that are declared rigid cannot be part of adaptive mesh domains.
- Elements in the adaptive domain cannot contain embedded elements or rebars.
- Symmetric results transfer cannot be done from an axisymmetric model that had solid elements in an adaptive domain.
- Import cannot be done from a model that had solid elements in the adaptive domain.
- It is not meaningful to drive a submodel using the nodes from a global model that were part of an adaptive mesh domain.

- Only enhanced hourglass control can be used with reduced-integration elements.
- When used with acoustic elements, adaptive mesh smoothing must be applied in steps prior to a coupled structural-acoustic analysis. It cannot be applied during a large-displacement dynamic analysis.
- Mesh smoothing assumes that the geometric changes in the acoustic domain do not lead to changes in material properties, such as fluid density.
- The coupling between the fluid and structure must be defined using a surface-based tie constraint with the slave surface defined on the acoustic domain.
- Nodes in the adaptive domain that are involved in constraints such as multi-point constraints ("General multi-point constraints," Section 35.2.2) and equations ("Linear constraint equations," Section 35.2.1) should be made non-adaptive by applying Lagrangian constraints.

Input file template

Applying ALE adaptive meshing for acoustic analysis

```
*HEADING
…
*ELEMENT, TYPE=…, ELSET=ACOUSTIC
Data lines to define acoustic elements
*ELEMENT, TYPE=…, ELSET=SOLID
Data lines to define structural elements
*SURFACE, NAME=TIE_ACOUSTIC
Data lines to define the acoustic surface interface with the structural mesh
*SURFACE, NAME=TIE_SOLID
Data lines to define the solid surface interface with the acoustic mesh
*TIE, NAME=COUPLING
TIE_ACOUSTIC, TIE_SOLID
…
*STEP
*STATIC
*ADAPTIVE MESH, ELSET=ACOUSTIC, MESH SWEEPS=10
…
*END STEP
**
*STEP
*STEADY STATE DYNAMICS, DIRECT
…
*END STEP
```

Applying ALE adaptive meshing in other uses

```
*HEADING
…
*ELEMENT, TYPE=C3D8, ELSET=..
```
Data lines to define solid elements
```
*NSET, NSET=LAG
```
Data lines to define nodes that should be nonadaptive
```
*NSET, NSET=SPATIAL
```
Data lines to define nodes that will have spatial adaptive mesh constraints applied
```
*ELEMENT, TYPE=…, ELSET=SOLID
```
Data lines to define structural elements
```
*STEP, NLGEOM=YES
*STATIC
*ADAPTIVE MESH, ELSET=SOLID, MESH SWEEPS=10
*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=LAGRANGIAN
LAG
*ADAPTIVE MESH CONSTRAINT, CONSTRAINT TYPE=SPATIAL, USER
SPATIAL
*END STEP
```

12.2.7 ALE ADAPTIVE MESHING AND REMAPPING IN Abaqus/Standard

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining ALE adaptive mesh domains in Abaqus/Standard," Section 12.2.6
- *ADAPTIVE MESH
- *ADAPTIVE MESH CONSTRAINT
- *ADAPTIVE MESH CONTROLS
- "Customizing ALE adaptive meshing," Section 14.14 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

ALE adaptive meshing consists of two fundamental tasks:

- creating a new mesh through a process called sweeping, and
- remapping solution variables from the old mesh to the new mesh with a process called advection.

You can control the process of mesh sweeping after which, if necessary, Abaqus/Standard will automatically perform advection. The default methods for creating a new mesh have been chosen carefully to work for acoustic analysis and for modeling the effects of ablation, or wear, of material. However, you may need to override the default choices to balance the robustness and efficiency of adaptive meshing or to extend the use of adaptive meshing for other types of applications.

Adaptive mesh smoothing is defined as part of a step definition. The adaptive meshing in Abaqus/Standard uses an operator split method wherein each analysis increment consists of a Lagrangian phase followed by an Eulerian phase. The Lagrangian phase is the typical Abaqus/Standard solution increment where neither mesh sweeps nor advection occur. Once the equilibrium equations have converged, mesh smoothing is applied. Following the adjustment of nodes through the mesh sweeping process, material point quantities are advected in an Eulerian phase to account for the revised meshing of the model in its current configuration. This operator split method is chosen to avoid unsymmetric Jacobian terms that would result when the advection and material straining occur simultaneously. Advection is not required for, and is not applied to, acoustic elements.

The ALE adaptive mesh sweeping algorithm

Adaptive mesh smoothing is performed after the structural equilibrium equations have converged. The mesh smoothing equations are solved explicitly by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated—based on the positions of neighboring nodes obtained during the previous mesh sweep—to reduce element distortion. The new position, $\mathbf{x}_{i+1}$, of a node is obtained as

$$\mathbf{x}_{i+1} = \mathbf{X} + \mathbf{u}_{i+1} = \mathbf{N}^N \mathbf{x}_i^N,$$

where $\mathbf{X}$ is the original position of the node, $\mathbf{u}_{i+1}$ is the nodal displacement, $\mathbf{x}_i^N$ are the neighboring nodal positions obtained during the previous mesh sweep, and $\mathbf{N}^N$ are weight functions obtained from one or a weighted mixture of the following methods. The displacements applied during sweeps are not associated with mechanical behavior.

Original configuration projection

Original configuration projection is the default in Abaqus/Standard and determines the weight function from a least squares minimization procedure that minimizes node displacement in a projection of the mesh back to the original configuration. This method of smoothing affects only deformations of the mesh and not the original mesh.

Volume smoothing

Volume smoothing determines the weight function by computing a volume-weighted average of the element centers in the elements surrounding the node. In Figure 12.2.7–1 the new position of node M is determined by a volume-weighted average of the positions of the element centers, C, of the four surrounding elements. The volume weighting will tend to push the node away from element center C1 and toward element center C3, thus reducing element distortion.

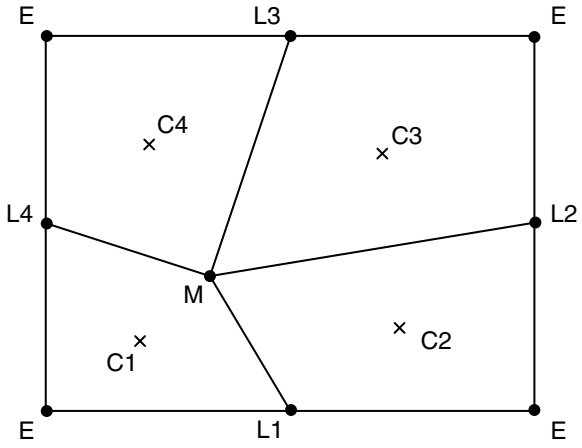


Figure 12.2.7–1 Relocation of a node during a mesh sweep.

Volume smoothing is supported in structured domains, where every node is surrounded by four elements in two dimensions or eight elements in three dimensions.

Combining smoothing methods

The default smoothing method in Abaqus/Standard is original configuration projection. To choose an alternate smoothing method or to combine the smoothing methods, you specify the weighting factor for each method. When more than one smoothing method is used, a node is relocated by computing a weighted average of the locations predicted by each chosen method. All weights must be zero or positive, and their sum must be nonzero. The weights are significant only in a relative sense; their values are normalized so that their sum is 1.0.

Input File Usage: ∗ADAPTIVE MESH CONTROLS, NAME=*name*
original configuration projection weight, volume smoothing weight

For example, the following option could be used to define an equal blend of original configuration projection and volume smoothing:

```
*ADAPTIVE MESH CONTROLS, NAME=name
0.5, 0.5
```

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, **Original configuration projection:** *original configuration projection weight*, **Volumetric:** *volume smoothing weight*

Geometric enhancements to the basic smoothing methods

The conventional forms of the basic smoothing methods may not perform well in highly distorted domains. You can use geometrically enhanced forms of the basic smoothing algorithms as a technique to mitigate distortion. These forms are heuristic and based on nodal locations only. Due to their heuristic nature, geometric enhancements may not always improve the mesh smoothing.

Input File Usage: Use the following option to apply geometric enhancements to the smoothing algorithm:

∗ADAPTIVE MESH CONTROLS, NAME=*name*,
GEOMETRIC ENHANCEMENT=YES

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Controls→Create**: **Name**: *name*, toggle on **Use enhanced algorithm based on evolving element geometry**

Application of the sweeping algorithm

The mesh smoothing process begins with the mesh in its current displaced equilibrium configuration. Nodes that have no displacement degrees of freedom, such as those connected to acoustic elements, are maintained at their most recent configuration. Mesh smoothing is then driven by distortions in the current configuration and by boundary constraints. These boundary constraints can be described directly through adaptive mesh constraints. In the case of structural-acoustic boundaries the structural mesh boundary provides a constraint that controls the smoothing of adjacent acoustic element regions.

When these boundary constraints are much larger than the characteristic element length in the adaptive mesh domain, significant geometric changes, such as the development of corners, can occur.

To prevent such changes, the constraints are applied gradually over a series of "sub-increments" onto the domain boundary. The number of sub-increments used is determined on the basis of the magnitude of the maximum surface displacement and the characteristic element dimension.

The remaining nodes (nodes not driven by constraints) are identified as interior nodes, free surface nodes, edge nodes, or corner nodes. These nodes are treated as described in "Defining ALE adaptive mesh domains in Abaqus/Standard," Section 12.2.6.

At the end of mesh sweeping the new geometry is checked to ensure that elements did not become severely distorted during mesh smoothing. Abaqus/Standard responds to severe distortion in different ways, depending on the elements and procedures used. When adaptive meshing is used with acoustic elements, the current analysis increment is repeated with a reduced time increment, followed by another adaptive mesh smoothing attempt. When adaptive meshing is used with other elements, severe distortion results in abandonment of mesh smoothing for that increment. In cases where adaptive mesh constraints are also defined, Abaqus/Standard aborts since the constraints cannot be satisfied.

Controlling the frequency of ALE adaptive mesh smoothing

In most cases the frequency of adaptive meshing is the parameter that most affects the mesh quality. By default, mesh smoothing will be performed after each converged structural analysis increment. You can change the frequency of adaptive meshing, except when adaptive mesh constraints are defined.

Input File Usage: *ADAPTIVE MESH, FREQUENCY=*number of increments*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the adaptive mesh domain below**, **Frequency:** *number of increments*

Controlling convergence of ALE adaptive mesh smoothing

The adaptive mesh smoothing equations are solved explicitly by sweeping iteratively over the adaptive mesh domain. During each mesh sweep, nodes in the domain are relocated based on the current positions of neighboring nodes to reduce element distortion.

Mesh smoothing is performed following the end of a converged increment. You can control the intensity of the mesh smoothing by defining the number of mesh sweeps required. When the displacements are large, more iterations are usually required. When used in acoustic analyses, more iterations are usually required when the volume of the elements in the acoustic domain decreases compared to the case when the volume increases during structural loading.

You can specify the number of mesh sweeps to be performed in each adaptive mesh increment. The default number of mesh sweeps is one.

By applying the mesh sweeping algorithm repeatedly, the mesh will converge; in other words, nodal positions are obtained that do not change with further mesh sweeping. However, it is usually not necessary to apply mesh smoothing until a converged mesh is obtained; the main objective is to reduce element distortion.

Input File Usage: *ADAPTIVE MESH, MESH SWEEPS=*number of sweeps*

Abaqus/CAE Usage: Step module: **Other→ALE Adaptive Mesh Domain→Edit**: toggle on **Use the adaptive mesh domain below**, **Remeshing sweeps per increment:** *number of sweeps*

The ALE adaptive mesh advection algorithm

Abaqus/Standard applies an explicit method, based on the Lax-Wendroff method, to integrate the advection equation. The key principle of the Lax-Wendroff method is replacement of the time derivatives of the material point quantities with the spatial derivatives using the classical relationship between the material time derivative, the referential derivative, and the spatial derivative. The update scheme is second-order accurate and provides some upwinding. Nodal quantities are advected by first converting them to the material point quantities.

Advection of the material quantities will generally result in loss of equilibrium, for two main reasons. The first reason is the errors in the advection process itself. To minimize the errors in advection, Abaqus/Standard imposes restrictions on the magnitude of the advection velocity by requiring that the Courant number for every element in the adaptive domain be less than one. In cases where the Courant number is greater than one you will be informed and Abaqus/Standard will generate multiple advection passes per increment. The second reason for the loss of equilibrium is changes in the representation of the underlying material quantities by the changed mesh. For example, consider a region of the structure having some stress gradients spanned initially by two elements. After mesh smoothing, the same region might have more than two elements. This will lead to slightly different volume integration while computing the internal force even when there are no errors in advection.

These sources of error in equilibrium are significant only when the mesh is too coarse to provide a good solution and mesh smoothing is carried out with such small frequency that the mesh motion is larger than the average element size. In practical applications these errors are typically insignificant, the resulting loss of equilibrium is generally small, and the residuals generated by the loss of the equilibrium fall within the limits of the Abaqus/Standard convergence criterion. Any loss of equilibrium is not propagated since equilibrium will again be satisfied at the end of the Lagrangian phase of the next increment.

Impact of advection on subsequent steps

To ensure that the results are output only for the configuration that satisfies equilibrium, Abaqus/Standard always outputs the results at the end of the Lagrangian phase. The Eulerian phase that follows the Lagrangian phase will leave the structure out of equilibrium for the next increment. This sequence has a consequence that after the last Eulerian phase is carried out at the end of the step, equilibrium will not be satisfied exactly at the beginning of the next step and the solution at the end of the step will differ slightly from the solution at the zero increment of the following step. Equilibrium can again be established by following the step that had adaptive meshing by a step that removes all the adaptive mesh domains and allows the structure to equilibrate. A one-increment step will usually suffice. This is particularly important when the following step is a perturbation procedure that uses the solution from the previous step as the base state.

Frequency steps that follow adaptive mesh steps will also be impacted, because element mass is not advected during mesh smoothing. This impact on the element mass can be significant, depending on the extent of adaptive mesh motion and change in element size due to mesh smoothing. Abaqus will provide a warning message in cases where adaptive meshing precedes a frequency step; you should evaluate

the impact of your updated mesh configuration when interpreting results from a frequency step in these cases.

Output

In adaptive meshing the integration point of an element will generally not refer to the same material point throughout the analysis. Contour plots of material variables will show correct spatial distribution, but history plots are not meaningful. The displacement of the nodes contains the material displacement as well as the displacement due to mesh motion. You can obtain measures of the volume lost due to adaptive mesh constraints with the partial model variable VOLC, which is useful when using adaptive mesh constraints to model ablation.

A summary of the adaptive meshing in each adaptive mesh domain is written to the message (`.msg`) file. This summary includes the total number of load increments over which the structural displacement is transferred to the fluid, the total number of mesh sweeps performed, the magnitude of the maximum displacement increment, and the node and degree of freedom at which the maximum displacement increment is measured. Warning messages are issued when geometric features change during mesh smoothing.

More detailed diagnostic output for adaptive mesh smoothing can be requested; see "The Abaqus/Standard message file" in "Output," Section 4.1.1. This output provides the magnitude of the maximum displacement and the node and degree of freedom where the maximum displacement increment occurs during each mesh sweep. In addition, the nodes experiencing changes in geometric features are listed.

Additional references

- Lax, P. D., and B. Wendroff, "Difference Schemes for Hyperbolic Equations with High-Order Accuracy," Communications on Pure and Applied Mathematics, vol. 17, p. 381, 1964.
- Lax, P. D., and B. Wendroff, "Systems of Conservations Laws," Communications on Pure and Applied Mathematics, vol. 13, pp. 217–237, 1960.

12.3 Adaptive remeshing

- "Adaptive remeshing: overview," Section 12.3.1
- "Selection of error indicators influencing adaptive remeshing," Section 12.3.2
- "Solution-based mesh sizing," Section 12.3.3

12.3.1 ADAPTIVE REMESHING: OVERVIEW

Abaqus/CAE provides an automated process to remesh your model adaptively. The goal of the adaptive remeshing process is to approach or reach targets on selected error indicators for a specified model and its accompanying load history. See "Adaptivity techniques," Section 12.1.1, for a comparison of this process to other Abaqus adaptivity methods.

Overview

The following steps are required to incorporate adaptive remeshing into your Abaqus/CAE model:

- You identify regions of the model where you wish to apply one or more adaptive remeshing rules. A remeshing rule defines the step during which it will be applied, the error indicator output variables and targets for those error indicators, the sizing method, and any constraints on element size. See "What are remeshing rules?," Section 17.13.1 of the Abaqus/CAE User's Guide.
- You define a succession of analysis jobs, an "adaptivity process," that will be run as Abaqus/CAE attempts to meet your remeshing rule targets. See "What is an adaptivity process?," Section 19.3.1 of the Abaqus/CAE User's Guide.

Based on these remeshing rules and your adaptivity process definition, Abaqus/CAE iteratively:

- executes an Abaqus/Standard analysis, which will write selected error indicator output variables based on your remeshing rule settings (See "Selection of error indicators influencing adaptive remeshing," Section 12.3.2),
- uses the error indicator variables in a sizing function to compute element sizes for a new mesh, respecting any size constraints you might specify (See "Solution-based mesh sizing," Section 12.3.3), and
- generates a new mesh in the regions specified, based on the computed element sizes. The neighboring regions will also be remeshed.

These iterations continue until either:

- all remeshing rule targets are satisfied, or
- a maximum number of remesh iterations is reached.

See "When will my mesh adaptivity stop iterating?," Section 19.3.2 of the Abaqus/CAE User's Guide, for more details. Figure 12.3.1–1 shows the interaction of Abaqus products and files in this process.

Typical applications

Adaptive remeshing can improve the quality of your simulation results. Adaptive remeshing can be helpful when:

- you are unsure how refined a mesh needs to be to reach a particular level of accuracy or how coarse the mesh can be without unacceptably impacting solution accuracy;
- it is difficult to design an adequately refined mesh near a region of interest, such as near a stress riser; or

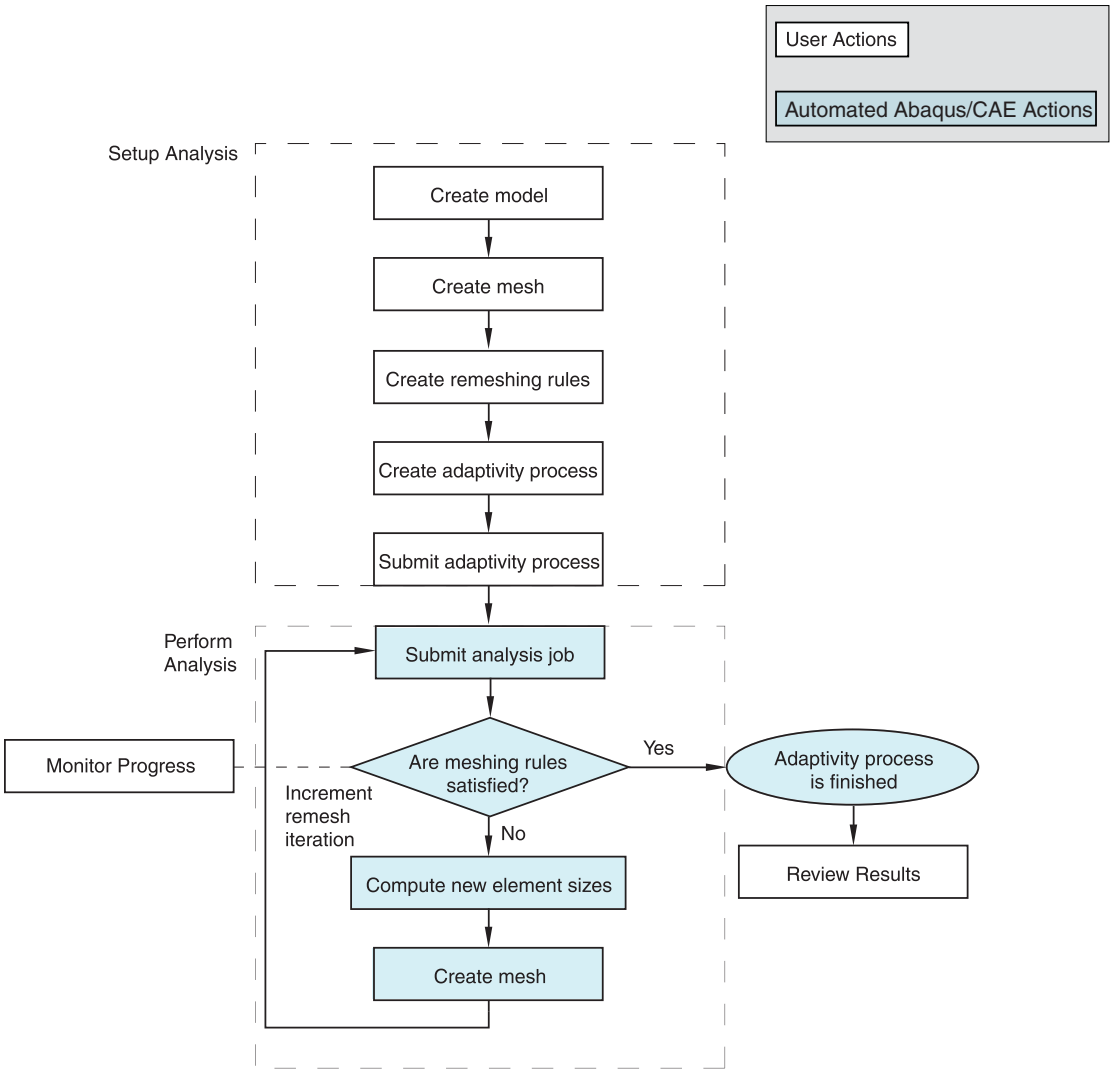


Figure 12.3.1–1 User actions and automated Abaqus/CAE actions in the adaptive remeshing process.

- you do not know a location of interest, such as with formation of a plastic zone, a priori.

An example of using adaptive remeshing to study the thermal and stress behavior of a bolted vessel is provided in "Thermal-stress analysis of a reactor pressure vessel bolted closure," Section 5.1.6 of the Abaqus Example Problems Guide. The example includes a Python script that you can run from

Abaqus/CAE to create the model and the remeshing rules. A second script allows you to submit the adaptivity process and to view the changing mesh as Abaqus/CAE computes new element sizes.

Example: stress riser

Figure 12.3.1–2 shows how adaptive remeshing generates a high-quality mesh for a typical notched specimen subjected to axial loading.

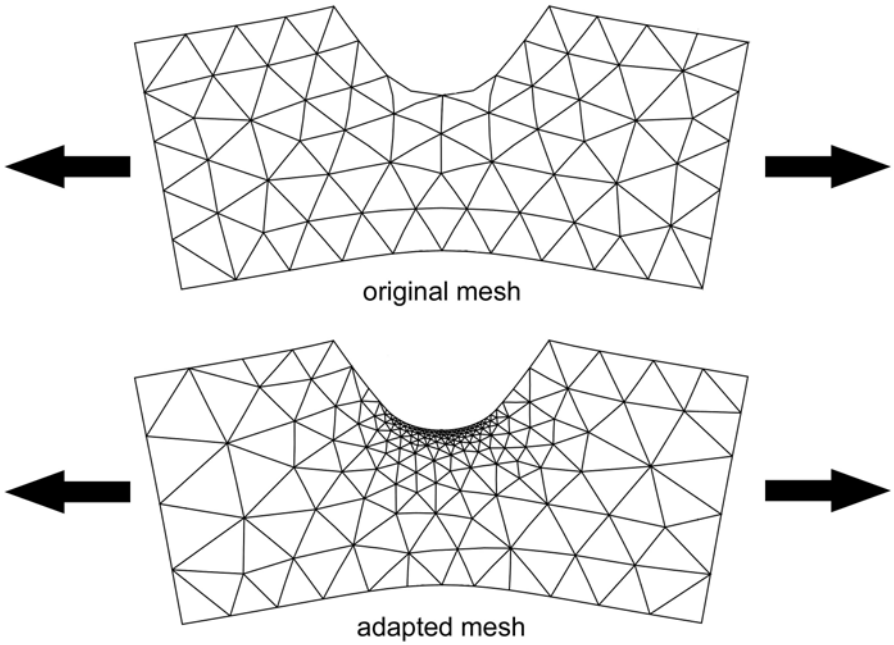


Figure 12.3.1–2 Stress riser mesh before and after refinement.

Figure 12.3.1–3 shows the effect of these mesh changes on solution accuracy in comparison to the effect of uniform mesh refinement on solution accuracy. Adaptive mesh refinement is much more efficient than uniform mesh refinement at reducing solution error.

Example: plastic hinge

This example, a doubly-notched specimen axially strained until a plastic hinge or band forms, is used to demonstrate how adaptive remeshing will focus a mesh on a plastic hinge. It illustrates the value of adaptive remeshing in cases where the region of interest may not be known a priori. Figure 12.3.1–4 shows the specimen and the region of active yielding. Figure 12.3.1–5 shows the original mesh and the adapted mesh after three adaptive remeshing iterations.

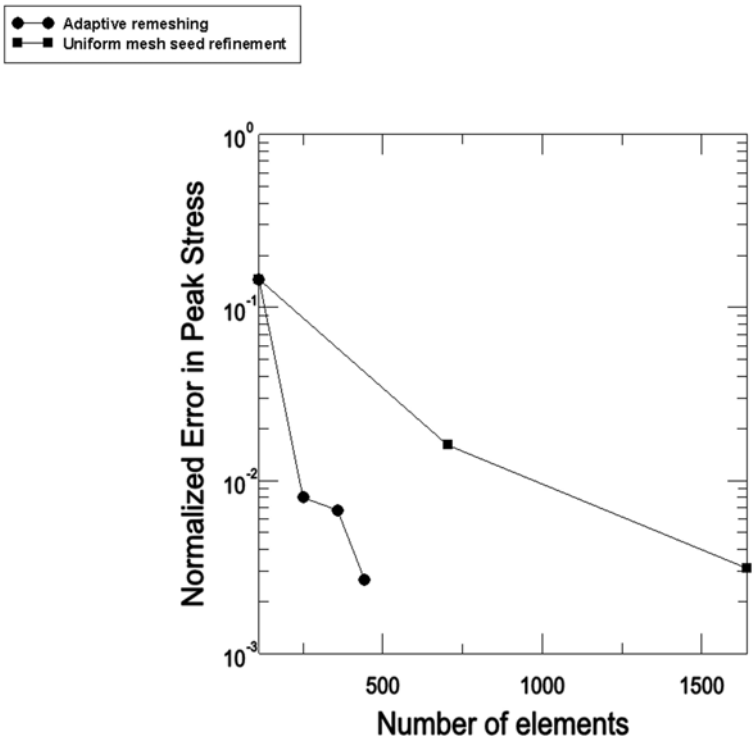


Figure 12.3.1–3 Comparison of adaptive remeshing to uniform mesh refinement based on boundary seeding.

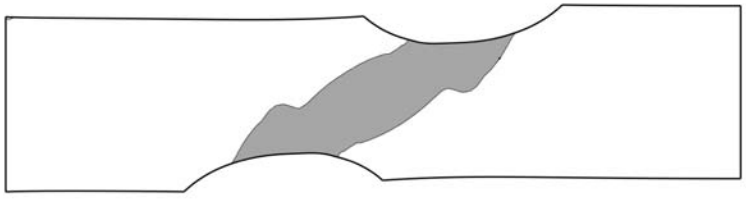


Figure 12.3.1–4 Region of active yielding in a doubly-notched specimen.

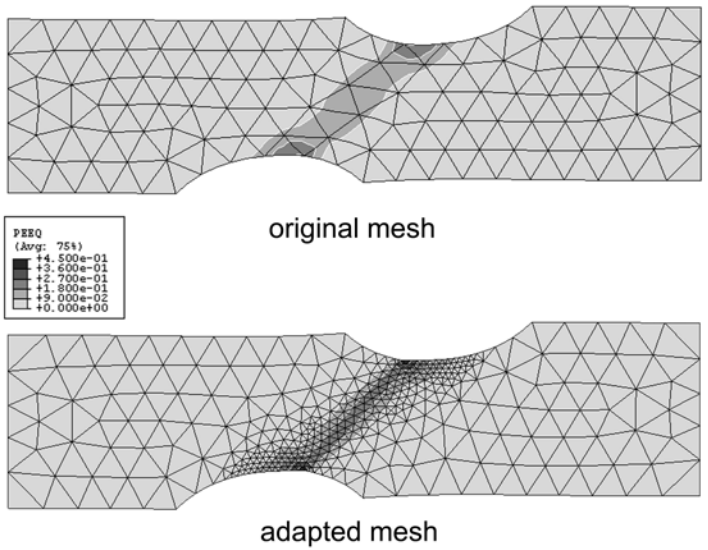


Figure 12.3.1–5 Mesh of doubly-notched specimen before and after adaptive remeshing.

Preparing your model for adaptive remeshing

You use Abaqus/CAE to do the following when performing adaptive remeshing:

- create the model and specify the boundary conditions and loading history,
- create remeshing rules,
- create an adaptivity process, and
- start and monitor the progress of the adaptivity process.

Creating the model

You do not have to consider adaptive remeshing when you create the model and specify the boundary conditions and loading history; however, before using adaptive remeshing you must do the following:

- create the geometry of the model—you cannot use an orphan mesh part—and
- provide an initial, nominal, mesh. This mesh can be fairly coarse. Providing an extremely coarse mesh, however, can result in more adaptive remesh iterations due to the poor quality of early remesh iteration error indicator calculations. You can, in typical cases, define a reasonable initial mesh by using the default part instance mesh seeding in Abaqus/CAE.

Creating a remeshing rule

You create and configure a remeshing rule using the Mesh module in Abaqus/CAE. See "Creating a remeshing rule," Section 17.21.1 of the Abaqus/CAE User's Guide, for details on defining remesh rules. Refer to "Selection of error indicators influencing adaptive remeshing," Section 12.3.2, and "Solution-based mesh sizing," Section 12.3.3, for details on the methods used to determine revised mesh size distributions.

Abaqus/CAE Usage: Mesh module: **Adaptivity→Remeshing Rule→Create**

Creating an adaptivity process

You create and configure an adaptivity process using the Job module in Abaqus/CAE. When you create an adaptivity process, you can specify the maximum number of remesh iterations to be performed and set various system resource parameters. See "Creating, editing, and manipulating jobs," Section 19.7 of the Abaqus/CAE User's Guide, for details.

Abaqus/CAE Usage: Job module: **Adaptivity→Create**

Performing adaptive remeshing with a provisional analysis

In some cases you will want to determine an adequate mesh for your model prior to conducting a fully detailed analysis, which might include many steps and complex behavior. A "provisional" analysis can often be used, along with adaptive remeshing, to efficiently determine a good mesh for a model. The provisional analysis may include various simplifications of your fully detailed analyis, such as

- replacing your steps with a single linear perturbation step with loading that adequately reflects your more general loading cases,
- removing plasticity and other material nonlinearities, and
- disabling geometric nonlinearity.

The provisional analysis approach may result in a mesh that is not ideally suited to your ultimate choice of loading. However, the cost for obtaining a mesh from a provisional model may be significantly lower than the case where your adaptivity process considers all of the complexity in the fully detailed analysis, and you may find the refined mesh adequate for use in a variety of analysis situations.

Special considerations

In general, the Abaqus adaptive remeshing process iterates automatically toward a better quality mesh; however, you should be aware of certain considerations.

Singularities

Stress singularities frequently result from geometric abstractions, such as reentrant corners and contact of a sharp edge in elastic materials, and from point loads or abruptly ended distributed load regions. In these situations the stress field near the singularity is unbounded, and no amount of mesh refinement will enable resolution of the correct solution. If you apply the adaptive remeshing process to regions of your

model that include singularities, the process will drive elements near the singularity to very small sizes. The end result may be unacceptably expensive analyses.

You can prevent excessively expensive analyses of models with singularities using the following techniques:

- Exclude the region of the singularity from consideration in the remeshing process. You exclude a region by partitioning the model and assigning remeshing rules only to regions away from the singularity.
- Apply a minimum element size constraint in the remeshing rule. Abaqus/CAE does assign a minimum element size by default, which is a fraction of the default part instance mesh seed. You can modify this constraint to achieve a quality solution near the singularity while avoiding an excessively refined mesh. You can also use the remeshing rule to control the rate at which Abaqus/CAE refines the size of the elements. Element size constraints may prevent an adaptivity process from achieving specified error indicator targets.
- Specify a maximum number of elements for a remeshing rule region. Abaqus/CAE adjusts the mesh sizing such that the generated total number of elements approximately satisfies this constraint.

Convergence issues

Figure 12.3.1–6 shows a typical history of an error indicator and the computational cost, in Abaqus/Standard, versus remesh iteration.

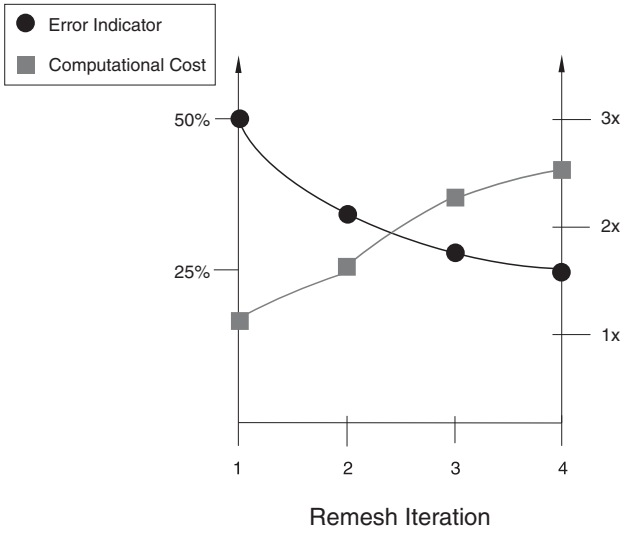


Figure 12.3.1–6 Error indicator and computational cost versus iteration for a model with a 25% error indicator target.

The example in Figure 12.3.1–6 shows a desirable convergence profile. The solution error indicator decreases monotonically and quickly to the desired 25% error indicator target. Accompanying this error indicator decrease is a moderate increase in computational cost, measured either in model degrees of freedom or time in Abaqus/Standard. Certain situations can interfere with this desirable convergence profile, as follows:

- If your initial mesh is too coarse, the error indicator variables may be of insufficient quality to result in a mesh that is sufficiently improved in the next iteration. The adaptive remeshing process typically creates a high-quality mesh eventually even if the initial mesh is quite coarse. However, some mesh iterations can be avoided with a reasonably refined initial mesh.
- Minimum element size constraints and constraints on the maximum number of elements that you specify when creating the remeshing rule can prevent the mesh from achieving sufficient refinement (in the extreme case of singularities this will always be the case) to satisfy your error indicator targets. You may be able to satisfy your targets by relaxing these constraints; for example, by decreasing the minimum element size. For more information, see "What are remeshing rules?," Section 17.13.1 of the Abaqus/CAE User's Guide.
- In addition to producing small mesh sizes resulting in a large number of elements, singularities can cause an adaptivity process to fail in achieving the error target or to require more remeshing iterations. As described in "Singularities," above, you can control the computational cost by specifying a minimum element size constraint or the maximum number of elements. In any case where a singularity exists within a remeshing rule region, you may see poor convergence in the error indicator results.
- Linear elements (C3D4, CPS4, etc.) and modified elements (C3D10M, CPS6M, etc.) converge slowly compared to quadratic elements (C3D10, CPS6, etc.) requiring a relatively large number of elements to achieve a given error target. Hence, you should use quadratic elements whenever possible.

Continuing a stopped adaptive remeshing process

The adaptive remeshing process is designed to be automatic—Abaqus/CAE performs a sequence of analyses as it continues to refine your mesh. However, there are occasions where the process will stop and you will want to continue adaptive remeshing from your most recent mesh:

- when you want to change remeshing rules for later remesh iterations, or
- when the adaptive remesh process fails to complete due to machine resource problems.

You can continue the adaptive remeshing process by resubmitting an existing adaptivity process, creating and submitting a new adaptivity process, or performing manual remeshing. See "Manually resizing and remeshing," Section 17.21.6 of the Abaqus/CAE User's Guide.

Limitations

Adaptive remeshing requires the use of Abaqus/CAE, and only Abaqus/Standard procedures are supported. Other specific limitations also apply.

Element types

Abaqus/CAE can perform adaptive remeshing only with elements of the following shapes (see "Which mesh controls can I use with adaptive remeshing?," Section 17.13.2 of the Abaqus/CAE User's Guide):

- Planar continuum triangles and quadrilaterals
- Shell triangles and quadrilaterals
- Tetrahedrals

Procedures

Abaqus/CAE can perform remeshing with the following Abaqus/Standard procedures:

- "Static stress analysis," Section 6.2.2 (general and linear perturbation).
- "Quasi-static analysis," Section 6.2.5.
- "Uncoupled heat transfer analysis," Section 6.5.2.
- "Fully coupled thermal-stress analysis," Section 6.5.3.
- "Coupled thermal-electrical analysis," Section 6.7.3.
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1.

12.3.2 SELECTION OF ERROR INDICATORS INFLUENCING ADAPTIVE REMESHING

Products: Abaqus/Standard Abaqus/CAE

References

- "Error indicator output," Section 4.1.4
- "Adaptive remeshing: overview," Section 12.3.1
- "Abaqus/Standard output variable identifiers," Section 4.2.1
- *CONTACT OUTPUT
- *ELEMENT OUTPUT
- "Understanding adaptive remeshing," Section 17.13 of the Abaqus/CAE User's Guide
- "Controlling adaptive remeshing," Section 17.21 of the Abaqus/CAE User's Guide

Overview

Your selection of which error indicator variables to use in adaptive remeshing rules for a particular analysis should take into consideration:

- characteristics of the error indicator variables;
- which fields exist and are of interest; and
- the nature of the loading.

Error indicator characteristics

Error indicator output variables provide estimates of solution accuracy (see "Error indicator output," Section 4.1.4). In the context of adaptive remeshing, error indicators help determine where the mesh should be refined or coarsened to achieve the specified accuracy targets (see "Adaptive remeshing: overview," Section 12.3.1 and "Solution-based mesh sizing," Section 12.3.3). This Section discusses additional characteristics of error indicators in the context of how well-suited they are for influencing adaptive remeshing in various analysis types.

Which fields exist and are of interest

Certain variables apply naturally to certain types of analyses. For example, the heat flux indicator (HFLERI) is used in analyses with temperature degrees of freedom. When selecting error indicator variables in the **Remeshing Rule** editor in Abaqus/CAE (see "What are remeshing rules?," Section 17.13.1 of the Abaqus/CAE User's Guide), your choices will be restricted to variables available for the selected procedure type.

The nature of the loading

Some error indicator variables only indicate discretization error at the current analysis time—the particular increment in a step. Other error indicator variables provide a record of the solution history

up to the current analysis time. For example, if your simulation involves non-proportional loading or a significantly nonlinear response, you will typically see better adaptive remeshing results when using error indicator variables that record the solution history. Table 12.3.2–1 lists the error indicator variables applicable to adaptive remeshing and indicates whether they record the solution history.

Table 12.3.2–1 Error indicator variables applicable to adaptive remeshing that record the solution history.

Solution Quantity	Error indicator variable (c_e)	Records the solution history?
Element energy density	ENDENERI	Yes
Mises stress	MISESERI	No
Equivalent plastic strain	PEEQERI	Yes
Plastic strain	PEERI	No
Creep strain	CEERI	No
Heat flux	HFLERI	No
Electric flux	EFLERI	No
Electric potential gradient	EPGERI	No

By default, when you create a remeshing rule, error indicators are specified for the final increment of the final step of your analysis and adaptive remeshing is based on error indicators in this final increment. When you select an error indicator that records the solution history, this default error indicator specification is appropriate for almost all analyses. However, for other error indicator variables that do not record the solution history, you may find it appropriate (for multistep cases with non-proportional loading, for example) to define mutiple remeshing rules for the same region, with each rule applied to a different step.

The examples that follow provide simple illustrations of typical cases and show appropriate choices of error indicator output variables.

Linear response example

Figure 12.3.2–1 illustrates the simplest load case, where the load is proportional to the step time and the model's response is linear. In this case the solution at the final increment would be proportional to any other increment. Therefore, it is appropriate to base the remeshing on the value of the error indicator in the last increment for any choice of error indicator variable.

Monotonic response example

Figure 12.3.2–2 illustrates a more general case, where the model has a nonlinear response—in this case resulting from a geometric nonlinearity—and the loading is monotonic but not generally proportional to the step time. The response of the model is slightly more general because the solution at a particular increment is not proportional to the solution at the final increment. However, the value of the error indicator output in the final increment still reflects the extreme of the model's response to the load history.

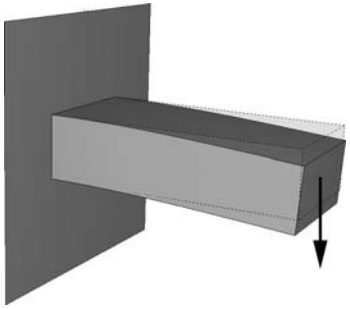


Figure 12.3.2–1 Proportional-loading, linear-response example: small deflection of a cantilever.

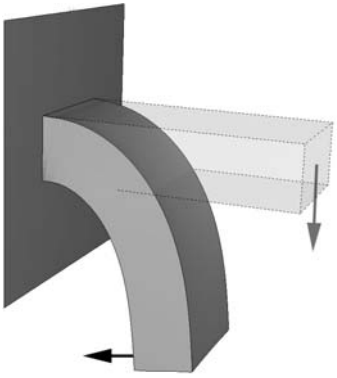


Figure 12.3.2–2 Monotonic response example: large deflection of a cantilever.

Therefore, it is appropriate to base the remeshing on the value of the error indicator in the last increment for any choice of error indicator variable.

General response example

Figure 12.3.2–3 illustrates a case where the loading characteristics change dramatically during the analysis.

Your choice of error indicator in this case will depend on the material model. The element energy density error indicator, ENDENERI, will account for the complexity of load history (and lead to an adapted mesh that provides an accurate solution through the analysis) regardless of the material type. If plastic deformation occurs, you also have the option to use the equivalent plastic strain, PEEQERI, or plastic strain, PEERI, error indicators. Plastic strain and the plastic strain error indicator generally do not capture history effects; for example, they do not account for peak straining in models undergoing symmetric strain reversals. This example, however, involves no strain reversals; therefore, PEERI would be a valid error indicator choice.

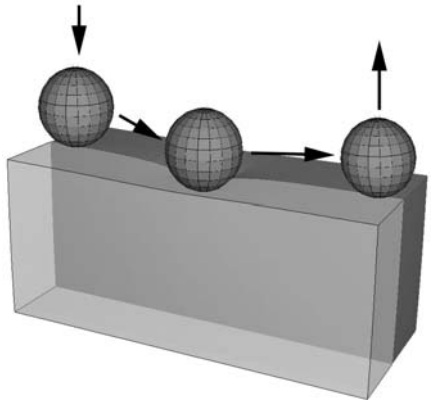


Figure 12.3.2–3 General response example: block subjected to a rigid indenter.

General multistep response example: die forming and springback

Figure 12.3.2–4 illustrates a further generalization of a general response. Here, a forming operation is simulated, and different steps are used for different stages of the operation.

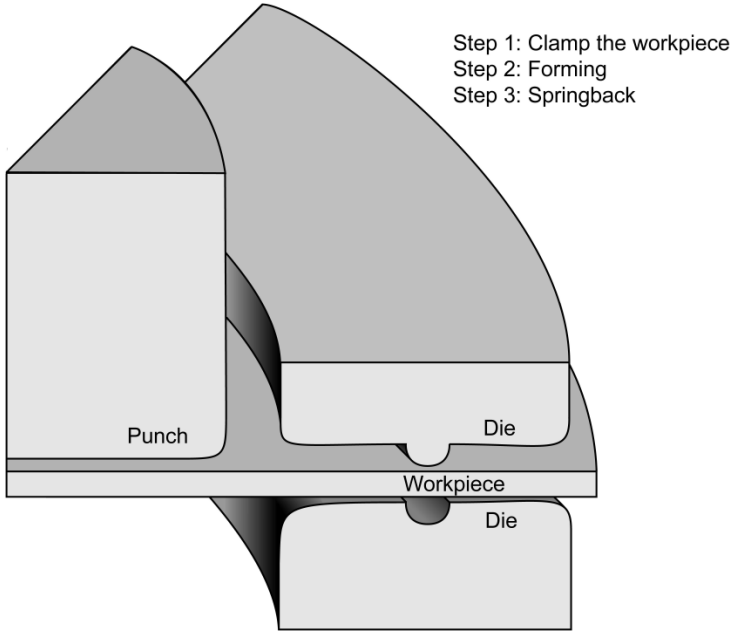


Figure 12.3.2–4 General multistep response example.

In this case the response of the model varies from step to step. You will typically want the error indicator to capture the extreme of the model's response to the load history adequately. However, you do not know if any particular increment captures this extreme. Therefore, you should select an error indicator variable that records the solution history.

12.3.3 SOLUTION-BASED MESH SIZING

Products: Abaqus/Standard Abaqus/CAE

References

- "Adaptive remeshing: overview," Section 12.3.1
- "Selection of error indicators influencing adaptive remeshing," Section 12.3.2
- "Understanding ALE adaptive meshing," Section 14.6 of the Abaqus/CAE User's Guide
- "Advanced meshing techniques," Section 17.14 of the Abaqus/CAE User's Guide

Overview

Solution-based mesh sizing:

- is performed in Abaqus/CAE; and
- operates on error indicator output variables and your remeshing rule parameters (see "Creating a remeshing rule," Section 17.21.1 of the Abaqus/CAE User's Guide) to determine an improved element size distribution for your mesh.

Basic operation of the sizing method

The sizing method calculates new element sizes during the adaptive remeshing process. Abaqus/CAE applies the sizing method to a field of error indicator variables and their corresponding base solution variables over the region defined by the remeshing rule. The output of a sizing method is a set of scalar sizes located at the nodes in the region defined by the remeshing rule. Figure 12.3.3–1 illustrates the sizing operation. Figure 12.3.3–1 shows the base solution and error indicator distributions after the first remesh iteration. The sizing method determines that the element size should be reduced in the region of greatest error indicator and increased in the region of the lowest error indicator. The mesh that is generated from these target element sizes is illustrated.

Characteristics of error indicators

The sizing method and parameter settings that you select have a significant impact on how adaptive remeshing changes the error indicator distribution in your model. You may, for example, choose a sizing method that aggressively reduces error indicators only near a stress riser. In other cases, where the global response of your structure is more important than local effects, you may choose a sizing method that attempts to reduce the error indicators to a uniform level throughout the region. To understand how the sizing methods affect the error indicators, you should first understand typical characteristics of the error indicator variables.

Figure 12.3.3–2 provides an illustration of an error indicator and corresponding base solution distribution on a generalized slice through a model.

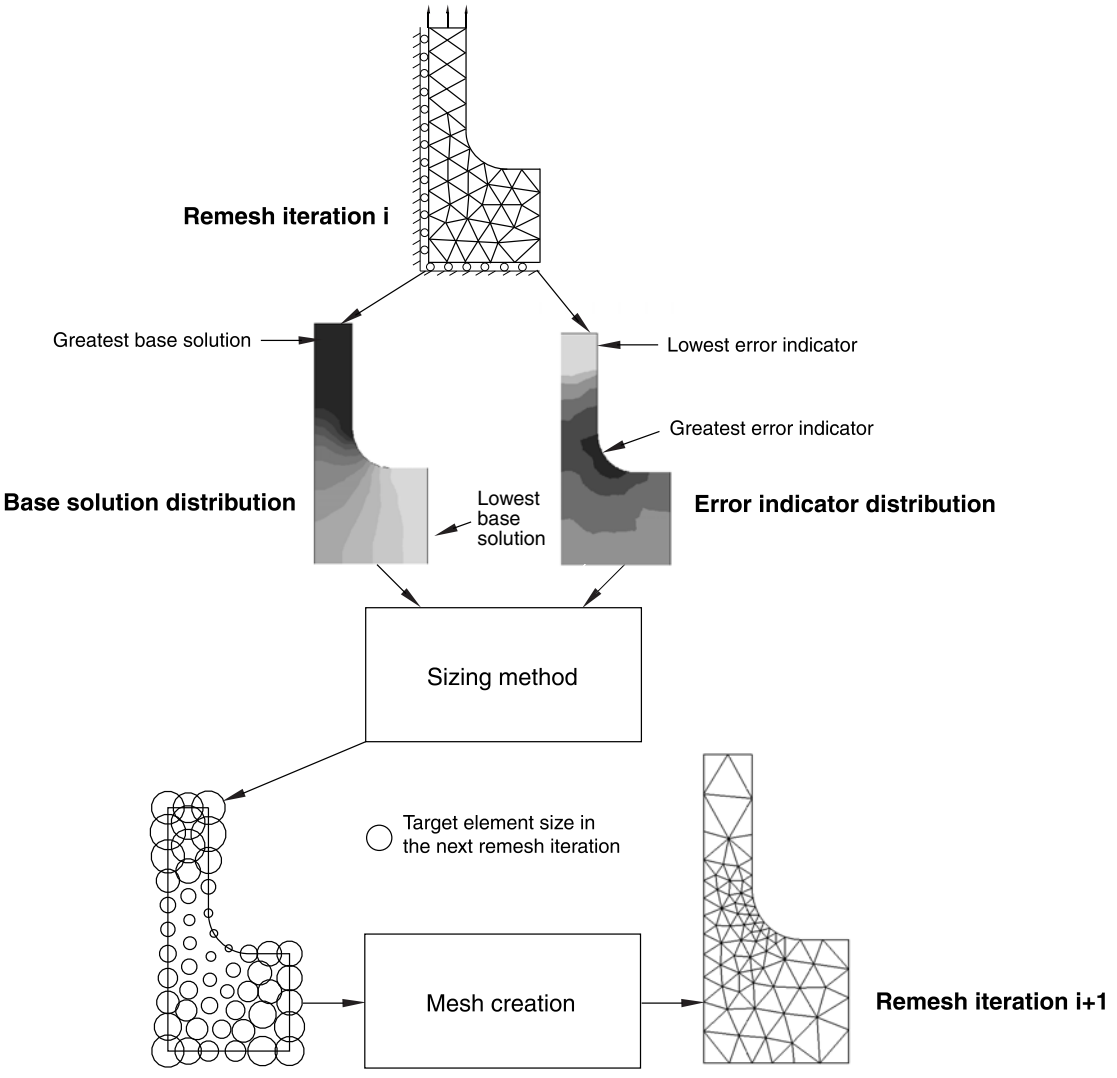


Figure 12.3.3–1 Sizing method operation and interaction with meshing.

Figure 12.3.3–2 illustrates the following error indicator characteristics:

- In regions where the value of the base solution is high, such as for element "i" in Figure 12.3.3–2, error indicator values can be low relative to local values of the base solution. In many cases you may want to use mesh refinement to drive these error indicators even lower.

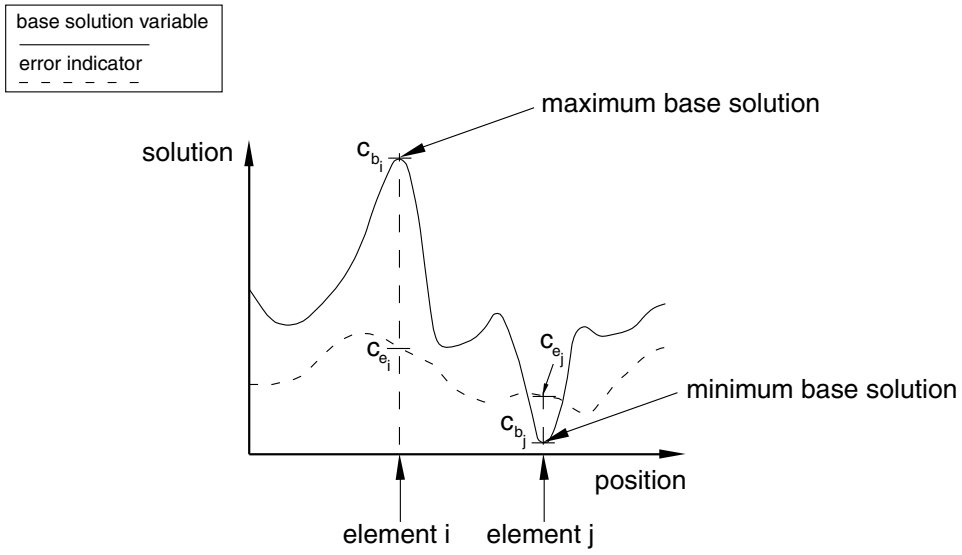


Figure 12.3.3–2 Error indicator and base solution distribution.

- In regions where the base solution is low, such as for element "j" in Figure 12.3.3–2, error indicator values can be high relative to the local values of the base solution. In many cases you may not be interested in obtaining an accurate solution in these regions.

These characteristics can affect your decision on which sizing method to use and what parameters to set in the sizing method.

Sizing methods

Sizing methods employ the concept of an error target, η, which is expressed in a normalized percentage form and which defines a general goal

$$\left(\frac{\mathbf{c_e}}{\mathbf{c_b}}\right) \times 100 \Rightarrow \eta,$$

where $\mathbf{c_e}$ is a measure of the error indicator and $\mathbf{c_b}$ is a measure of the base solution. Based on your definition of the error targets when you created the remeshing rule, Abaqus/CAE creates a size distribution that attempts to meet your error target in the subsequent analysis job using the remeshed model. The specific meaning of an error target depends on your choice of the sizing method.

Abaqus/CAE provides two fundamental sizing methods: **Minimum/maximum control** and **Uniform error distribution**. You can also choose a third method, **Default method and parameters**, which results in Abaqus/CAE choosing one of the fundamental sizing methods for you, based on your choice of error indicator variable.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**

Minimum/maximum control

The minimum/maximum control method provides the most flexibility in the remeshing of your model. This method has the following characteristics:

- Two error indicator targets for controlling the sizing. $\eta_{\max}$ controls the sizing in regions where the base solution (such as stress) is highest, and $\eta_{\min}$ controls the sizing in regions where the base solution is lowest.
- A continuous variation in error targets between regions of high and low base solution values, with a bias factor parameter provided to control the variation.
- To avoid excessive refinement at elements with a small base solution, a global averaged element base is chosen when the element base solution is smaller than the global averaged element base.
- If singularities are present in the remeshing rule region, this method will fail to satisfy the error target because the maximum base solution, which occurs at the location of the singularity, is unbounded.

You can either allow Abaqus to choose the targets automatically, or you can specify the error targets. Similarly, you can accept the default bias factor displayed by Abaqus/CAE, or you can specify a qualitative bias factor.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Minimum/Maximum control**

Allowing Abaqus/CAE to choose the error targets

If you specify the minimum/maximum error control method without setting error targets, Abaqus/CAE automatically chooses the error targets, $\eta_{\max}$ and $\eta_{\min}$. Both targets are computed as a fraction of the error indicator result in the previous remesh iteration analysis. Automatic error target reduction is a good choice for mesh refinement studies, where you have no specific error target goal but want to see the impact of mesh refinement on your results.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Automatic error target reduction**

Specifying the error targets

As an alternative to automatic error target reduction, you can specify the two error targets, $\eta_{\max}$ and $\eta_{\min}$. Figure 12.3.3–2 illustrates these two locations. $\eta_{\max}$ is applied to element i, and $\eta_{\min}$ is applied to element j.

Using the value of the two error targets, Abaqus/CAE applies a sizing method that attempts to meet both $\eta_{\min}$ and $\eta_{\max}$ at their respective locations.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Fixed error targets**; enter the maximum base solution error indicator target, $\eta_{\max}$, and the minimum base solution error indicator target, $\eta_{\min}$.

Bias factor

You can use the bias factor definition in the remeshing rule to further tune the distribution of sizing between maximum and minimum base solution locations. The bias factor defines the gradient of the size distribution between these two extremes in your remesh region, as shown in Figure 12.3.3–3.

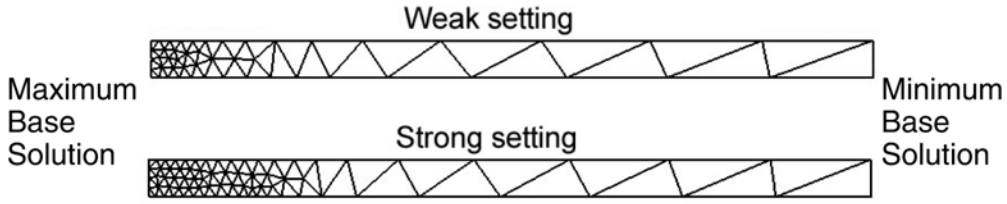


Figure 12.3.3–3 The impact of the bias factor on the element size distribution.

You can set this factor between two qualitative extremes, "weak" and "strong." At the weak extreme, element sizes will increase most quickly at locations moving away from the maximum base solution. At the strong extreme, element sizes will increase most slowly. The default setting is a bias toward the strong extreme.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Mesh Bias**; drag the slider to a setting between **Weak** and **Strong**

Uniform error distribution

The uniform error distribution method provides a single error indicator target, η, for controlling the sizing. Abaqus/CAE applies a sizing method such that the total error in the remeshing rule region is distributed uniformly across all the elements and satisfies the given error indicator target. This method attempts to satisfy the error indicator target collectively for the whole remeshing rule region but not at every element. Therefore, the presence of singularities will not prevent the adaptivity process from achieving the error target.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Uniform error distribution**

Allowing Abaqus/CAE to choose the error target

If you specify the uniform error distribution method without setting an error target, Abaqus/CAE automatically chooses the error target, η. The target is computed as a fraction of the error indicator result in the previous remesh iteration analysis. Automatic error target reduction is a good choice for mesh refinement studies, where you have no specific error target goal but want to see the impact of mesh refinement on your results.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**; choose **Automatic error target reduction**

Specifying the error target

As an alternative to the automatic error target reduction, you can specify the single error target, η. When you use the uniform error distribution method, Abaqus/CAE compares the error target to a global norm of a normalized form of the error indicator. Such an approach ensures a globally converging mesh within the region.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Error Targets**: choose **Fixed error target**; enter the error indicator target, η

Default sizing methods and parameters

This method results in application of the **Automatic error target reduction** form of either the **Minimum/maximum control** or **Uniform error distribution** method, with the method applied based on the error indicator variable according to Table 12.3.3–1.

Table 12.3.3–1 Default sizing method for each error indicator.

Solution Quantity	Error indicator variable	Default sizing method
Element energy density	ENDENERI	Uniform error distribution
Mises stress	MISESERI	Minimum/maximum control
Equivalent plastic strain	PEEQERI	Minimum/maximum control
Plastic strain	PEERI	Minimum/maximum control
Creep strain	CEERI	Minimum/maximum control
Heat flux	HFLERI	Uniform error distribution
Electric flux	EFLERI	Minimum/maximum control
Electric potential gradient	EPGERI	Minimum/maximum control

When your remeshing rule refers to multiple error indicators, sizing methods will be applied independently to each error indicator variable with the resulting local size based on the smallest size calculated from each sizing method.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Sizing Method**: **Method**: choose **Default methods and parameters**

Example: Plate with a circular stress riser

The difference between the basic behavior of the minimum/maximum control and the uniform error distribution methods is illustrated by a simple example. Figure 12.3.3–4 shows the stress result for a simple loading of a plate with a hole.

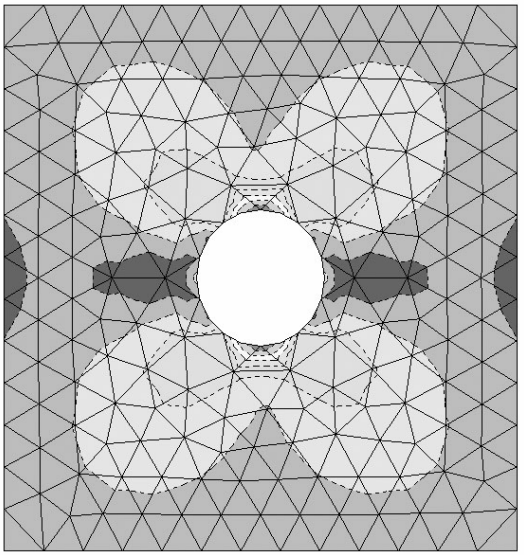


Figure 12.3.3–4 Initial mesh and Mises stress distribution for a plane stress plate with a hole, subjected to a uniform horizontal boundary traction.

Minimum/maximum control

Figure 12.3.3–5 illustrates the adaptive mesh that was generated by Abaqus/CAE when the user selected the minimum/maximum control method and specified the two error targets ($\eta_{\min}$ and $\eta_{\max}$). In this example $\eta_{\min}$=5% and $\eta_{\max}$=1%, and the mesh bias is set to the default setting. These settings result in a mesh that focuses tightly around the hole, the stress riser, while transitioning smoothly to a relatively coarse mesh away from the hole.

Uniform error distribution

Figure 12.3.3–6 illustrates the adaptive mesh that was generated by Abaqus/CAE when the user selected the uniform error distribution method and specified the single uniform error indicator target (η). In this example η=1%. This setting results in a mesh that focuses around the hole, the stress riser, while also refining the mesh in less stressed regions.

Impact of additional remeshing rule settings

You specify the sizing method when you create a remeshing rule, and the sizing method calculates new element sizes during the adaptive remeshing process. However, the following additional settings in the remeshing rule can affect the mesh generated by Abaqus/CAE, regardless of the sizing method that you selected:

- region selection,
- step and frame selection,

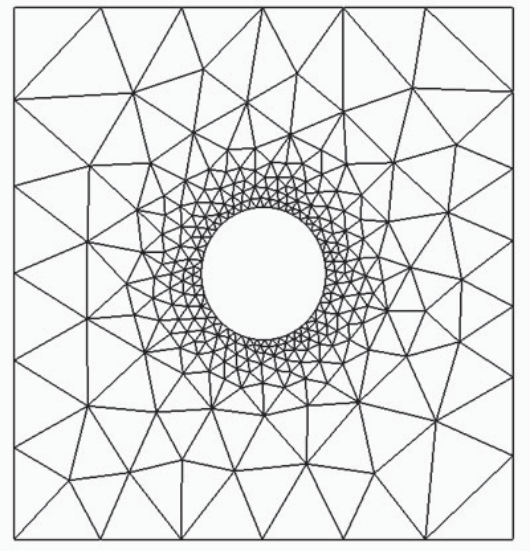


Figure 12.3.3–5 Adaptive remesh resulting from the minimum/maximum control sizing method.

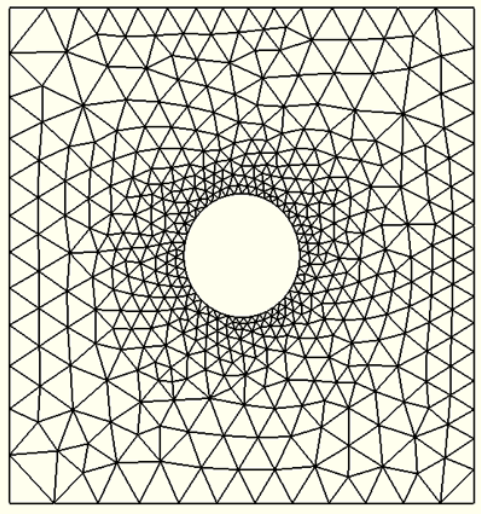


Figure 12.3.3–6 Adaptive remesh resulting from the uniform error distribution sizing method.

- size constraints,
- approximate maximum number of elements, and
- refinement and coarsening rate factors.

Region selection

Sizing methods are defined across sets of elements, corresponding to the regions over which the remeshing rules were applied in Abaqus/CAE. Within each set of elements, Abaqus/CAE applies the sizing operation to the error indicator variables specified in the remeshing rule. The results of the sizing operation are based on the extrapolation of whole element calculations to the nearest nodes, and the results are node based.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Edit Region**

Step and frame selection

Abaqus applies sizing operations to error indicator variables from only the last available frame in a specified step. See "Error indicator characteristics" in "Selection of error indicators influencing adaptive remeshing," Section 12.3.2, for a discussion of how your selection of the step, frame, and error indicator can affect your ability to capture the response in transient analyses.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Step and Indicator**: **Step**; select the step to which the rule is applied

and

Mesh module: **Create Remeshing Rule**: **Step and Indicator**: **Output Frequency**; choose either **Last increment of step** or **All increments of step**

Size constraints

When you create the remeshing rule, you can constrain the sizing operation from specifying elements greater than or less than size constraints that you define for the remesh rule region. Abaqus/CAE provides default settings for these constraints.

- The default minimum element size constraint is 1% of the default boundary seed size for the part instance to which the remeshing rule is applied.
- The default maximum element size constraint is 10 times the default boundary seed size for the part instance to which the remeshing rule is applied.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Element Size**

Approximate maximum number of elements

For a remeshing rule you can specify an approximate limit for the maximum number of elements. By using this constraint, you can control the cost of your analysis and ensure that unreasonably large meshes are not created. If the target error requires more elements than the specified limit when this constraint is defined, Abaqus/CAE will reduce the target error internally so that the generated elements approximately satisfy the specified element count. The use of this constraint may prevent an adaptivity process from achieving the error targets. By default, this constraint is not active.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Approximate maximum number of elements**

Refinement and coarsening rate factors

The refinement and coarsening factors that you specify define a constraint on the mesh size in terms of iteration to iteration changes to the mesh. These factors modulate the aggressivity of the sizing methods. The refinement factor controls the refinement of the mesh or the introduction of smaller elements. The coarsening factor controls the coarsening of the mesh or the introduction of larger elements. Abaqus/CAE provides default settings for these rate factors, which are designed to prevent excessive coarsening or prohibitively expensive refinement in a single remesh iteration.

The refinement factor can have a significant effect on the convergence of the adaptive meshing procedure. Once you have settled on sizing method parameters that work well for your application, you may be able to achieve faster and more efficient mesh convergence by increasing the refinement factor. In cases where your adaptivity process is not converging well, however, an increased refinement factor could result in an excessive increase in elements in a remesh iteration.

Abaqus/CAE Usage: Mesh module: **Create Remeshing Rule**: **Constraints**: **Rate Limits**

Reconciling overlapping remeshing rules

Abaqus/CAE imposes no restrictions on the region or the steps associated with your remeshing rules. You can apply multiple remeshing rules and, hence, sizing functions to the same region at the same time. Similarly, you can specify remeshing rules that overlap one another. When Abaqus/CAE generates the new mesh, it considers all of the remeshing rules at all of the locations and uses the smallest calculated element size to drive the meshing algorithm.

12.4 Analysis continuation after mesh replacement

- “Mesh-to-mesh solution mapping,” Section 12.4.1

12.4.1 MESH-TO-MESH SOLUTION MAPPING

Product: Abaqus/Standard

Reference

- *MAP SOLUTION

Overview

Mapping a solution from one mesh to another is a step in a remeshing analysis technique, where a mesh that has deformed significantly from its original configuration is replaced by a mesh of better quality and the analysis continues. The solution mapping technique:

- is used when elements become so severely distorted during an analysis that they no longer provide a good discretization of the problem;
- maps the solution from an old, deformed mesh to a new mesh so that the analysis can continue; and
- can be used only with continuum elements.

Refer to "Adaptivity techniques," Section 12.1.1 for a high-level discussion comparing this and other Abaqus adaptivity methods.

When to remesh

Abaqus/Standard uses a Lagrangian formulation: the mesh is attached to the material and, thus, deforms with the material. When the strains become large in geometrically nonlinear analyses, the elements may become so severely distorted that they no longer provide a good discretization of the problem. Severe distortion may occur in rubber elasticity problems or in plastic or viscoplastic calculations, especially when modeling manufacturing processes. When severe distortion occurs, it is necessary to remesh: to create a new mesh better designed to continue the analysis and to map the old-model solution onto this mesh.

You must decide when remeshing is needed. This decision can be assisted by looking at the magnitude of strains that have occurred during the phase of the analysis using a particular mesh, as discussed later. When remeshing is required, a new mesh for the deformed object must be generated using the mesh generation capability in Abaqus or an external mesh generator. The analysis is then continued as a new problem using the new mesh. In most cases it will be desirable to transfer the solution from the old mesh to the new mesh.

Discontinuity in the solution

Whenever the solution is mapped from another mesh, you can expect that there will be some discontinuity in the solution because of the change in the mesh and as a consequence of the solution mapping algorithm. If the discontinuity is significant, it is an indication that the meshes are too coarse or that the remeshing should have been done at an earlier stage before too much distortion occurred.

The remeshing technique works well, provided that the meshes are sufficiently fine for the problem and that the remeshing is done before the elements become too distorted.

Remeshing criterion

The first requirement for remeshing is some indication that the mesh is becoming distorted in regions where this distortion could cause the solution to be inaccurate. One possible criterion for remeshing would be extreme element distortion in areas where high strain gradients need to be resolved accurately. Inaccuracy is less of a concern if the distorted elements have moved into an area where further changes in the strain field are uniform; the elements can represent states of constant strain accurately no matter how distorted they are. Ultimately, however, the decision to remesh is a matter of judgment.

Generating a new mesh

Once you have decided that the current mesh is inadequate, a new mesh that is more suitable to the current state of the problem must be generated by using the mesh generation capabilities in Abaqus or an external mesh generator. Deformed configuration plots may be useful to provide data about the current shape of the object being modeled. Usually the external surface can be defined for use in a mesh generator from the results file output at the sets of nodes that form the surfaces of the body. See "Erosion of material (sand production) in an oil wellbore," Section 1.1.22 of the Abaqus Example Problems Guide and "Upsetting of a cylindrical billet: quasi-static analysis with mesh-to-mesh solution mapping (Abaqus/Standard) and adaptive meshing (Abaqus/Explicit)," Section 1.3.1 of the Abaqus Example Problems Guide.

Remeshing a contact problem

In a region of contact the new mesh must conform closely to the shape of the surface from the old analysis. This requirement is especially important for problems involving contact between two deformable bodies; if the surfaces defined by the new mesh are even slightly different from the surfaces in the old analysis, the contact algorithms may fail to converge.

Specifying the solution to be interpolated onto the new mesh

The simulation is continued by interpolating the solution onto the new mesh from the output databases generated with the old mesh.

Specifying the time at which the solution must be read

Solution transfer will occur, by default, from the latest step and increment for which solution variables are available. Alternatively, you can specify the step and increment at which the old solution will be read.

Input File Usage: *MAP SOLUTION, STEP=*step*, INC=*increment*

Obtaining equilibrium

An initial step should be included to allow Abaqus/Standard to check for equilibrium after this interpolation has been done. By default, Abaqus/Standard resolves the stress unbalance linearly over

the step (see the discussion on establishing equilibrium when an initial stress field is applied in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1). You can choose to have the stress unbalance resolved in the first increment instead.

Input File Usage: Use the following option to have Abaqus/Standard resolve the stress unbalance linearly over the step:

∗MAP SOLUTION, UNBALANCED STRESS=RAMP

Use the following option to have Abaqus/Standard resolve the stress unbalance in the first increment of the step:

∗MAP SOLUTION, UNBALANCED STRESS=STEP

Translating and rotating the old-job mesh

The mesh from the old job can be repositioned prior to performing the mapping by giving a translation and/or rotation relative to the global origin. Specify a translation by giving a translation vector. Specify a rotation by giving two points to define a rotation axis plus a right-handed angular rotation around that axis.

Input File Usage: ∗MAP SOLUTION, STEP=*step*, INC=*increment*
translation vector data
rotation axis and angular rotation data

Required output from the old job

The files required for restart and the output database must be requested for the old job. Nodal displacement results are not output automatically from the old job; you must explicitly request output of the displacement variable U for all nodes, as described in "Node output" in "Output to the output database," Section 4.1.3. Alternatively, you can request preselected field output and obtain node displacement output sufficient for solution mapping.

In fully coupled procedures you must request nodal output of the coupled field variable to the output database (see Table 12.4.1–1).

Table 12.4.1–1 Output database nodal output requirements for fully coupled procedures.

Procedure	Nodal output variable
"Fully coupled thermal-stress analysis," Section 6.5.3	NT11
"Coupled pore fluid diffusion and stress analysis," Section 6.8.1	POR
"Geostatic stress state," Section 6.8.2	POR

Identifying the old job

Specify the name of the old job from which restart and results data will be obtained by using the **oldjob** parameter in the command for running Abaqus or by answering a request made by the command procedure (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2). The

files required from the old job include: the restart file (**.res**), the output database (**.odb**), the model database (**.mdl**), the state database (**.stt**), and the part (**.prt**) file.

Solution mapping algorithm

Solution mapping operates by interpolating results from nodes in the old mesh to points (either nodes or integration points) in the new mesh. The first step, therefore, involves associating solution variables with nodes in the old mesh. For nodal solution variables, such as nodal temperature or pore pressure, the association is already made. For integration point variables Abaqus obtains the solution variables at the nodes of the old mesh by extrapolating values from the integration points to the nodes of each element and then averaging these values over all similar elements abutting each node.

Next, the location of each point in the new mesh is obtained with respect to the old mesh. The new mesh points include integration points in all cases and nodes in procedures that record nodal state in addition to displacements (for example, nodal temperatures in coupled temperature-displacement procedures).

1. The element (in the old mesh) in which the point lies is found, and the point's location in that element is obtained. (This procedure assumes that all points in the new mesh lie within the bounds of the old mesh: warning messages are issued if this is not so, and the values of the variables are set to zero.)
2. The variables are then interpolated from the nodes of the old element to the points in the new model.

All necessary variables are interpolated automatically in this way so that the solution can proceed with the new mesh.

Solution diffusion

This algorithm introduces some diffusion in the mapped solution. The effect of the diffusion scales with the solution gradient in the old mesh; hence, even for regions of the model where the mesh does not change from the old to the new model, diffusion due to the mapping can result in significantly different mapped quantities when the old-mesh solution gradient is high. You can moderate this effect by refining the old mesh in regions where solution gradients are high or by remeshing earlier.

Procedures

The solution mapping capability is supported for the following procedures:

- "Static stress analysis," Section 6.2.2
- "Quasi-static analysis," Section 6.2.5
- "Fully coupled thermal-stress analysis," Section 6.5.3
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1
- "Geostatic stress state," Section 6.8.2

Initial conditions

The solution mapped from the initial analysis forms the initial conditions for the remeshed analysis. Initial conditions such as temperature for a pure stress/displacement analysis can be specified. Any other specified initial conditions will be ignored.

Boundary conditions

Boundary conditions are not carried over from the old mesh to the new mesh. The boundary conditions applied at the beginning of the remeshed analysis should normally be the same as those in effect at the step and increment selected from the initial analysis. Although boundary conditions can be changed, the problem may fail to converge if the structure is far from an equilibrium state.

There are no restrictions on applying boundary conditions in a mapped solution analysis. Boundary conditions can be applied to all available degrees of freedom in the same way as they are applied in an analysis without a mapped solution (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1).

Loads

There are no restrictions on applying loads in a mapped solution analysis. Loads can be applied in the same way as they are applied in an analysis without a mapped solution.

The loads applied at the beginning of the remeshed analysis should normally be the same as those in effect at the end of the initial analysis. Although the loads can be changed, the problem may fail to converge if the structure is far from an equilibrium state.

Predefined fields

Temperature and field variables are mapped from the old mesh to the new mesh. If the number of field variables is changed in the remeshed analysis, the number common to both analyses will be transferred. Predefined fields can be modified in the same way as they are modified in an analysis without solution mapping (see "Predefined fields," Section 34.6.1).

Material options

Any of the mechanical constitutive models available in Abaqus can be used in a mapped solution analysis (see Part V, "Materials"). There is no restriction on agreement between material models in the old and new analyses. The solution mapping algorithm will transfer those variables common to both models. You must ensure that the material models are compatible.

Elements

The solution mapping capability can be used only with continuum elements (see "Solid (continuum) elements," Section 28.1.1).

Output

There is no output specific to a mapped solution analysis. Output can be requested in the same way as in an analysis without a mapped solution. The output variables available in Abaqus are listed in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

```
*HEADING
*NODE
Data lines to define the new-model nodes occupying the space of the old model
in its deformed configuration
*ELEMENT
Data lines to define the new-model elements occupying the space of the old model
in its deformed configuration
...
*MAP SOLUTION, STEP=step, INC=inc
translation and rotation data
*STEP
*STATIC (or *COUPLED TEMPERATURE-DISPLACEMENT or *GEOSTATIC
or *SOILS or *VISCO)
...
*END STEP
```

13. Optimization Techniques

Structural optimization: overview 13.1
Optimization models 13.2

13.1 Structural optimization: overview

- "Structural optimization: overview," Section 13.1.1

13.1.1 STRUCTURAL OPTIMIZATION: OVERVIEW

Structural optimization using Abaqus is an iterative process that helps you refine your designs. The result of a well-designed structural optimization is a component that is lightweight, rigid, and durable. Abaqus provides the following approaches to structural optimization: topology optimization, shape optimization, sizing optimization, and bead optimization. Topology optimization starts with an initial model and determines an optimum design by modifying the properties of the material in selected elements, effectively removing elements from the analysis. Shape and sizing optimization further refine the model. Shape optimization modifies the surface of the component by moving the surface nodes to reduce local stress concentrations. Sizing optimization modifies the sheet thickness of sheet metal components; typically, to increase the stiffness or reduce vibration. Bead optimization creates stiffening beads in a shell model. Topology, shape, sizing, and bead optimization are governed by a set of objectives and constraints.

Optimization is a tool for shortening the development process by adding value to a designer's experience and intuition with an automated procedure. To optimize your model, you need to know what to optimize. It is not sufficient to say that you want to minimize stresses or maximize eigenvalues, your statements must be more specific. For example, you might want to minimize the maximal nodal stresses experienced during two load cases. Similarly, you might want to maximize the sum of the first five eigenvalues. The goal of an optimization is called the objective function. In addition, you can enforce certain values during the optimization. For example, you can specify that the displacement of a given node must not exceed a certain value. An enforced value is called a constraint.

You use Abaqus/CAE to create the model to be optimized and to define, configure, and execute the structural optimization. For more information, see Chapter 18, "The Optimization module," of the Abaqus/CAE User's Guide.

Terminology

Structural optimization introduces its own terminology. The following terms are used throughout the Abaqus documentation and the Abaqus/CAE user interface:

- **Design area:** The design area is the region of your model that the structural optimization modifies. The design area can be the whole model, or it can be a subset of the model containing only selected regions. Given the prescribed conditions (such as boundary conditions, loads, and manufacturing constraints),
  - a topology optimization process removes and adds material from elements in the design area while it attempts to reach an optimal design,
  - a shape optimization modifies the surface of the design area by moving surface nodes,
  - a sizing optimization modifies the thickness of the design area by changing the thickness of shell elements, and
  - a bead optimization moves nodes of shell elements in the design area in the direction of the shell normal.

- **Design variables:** For an optimization problem, the design variables represent the parameters to be changed during the optimization.

  For a topology optimization, the densities of the elements in the design area are the design variables. The Optimization module changes the density during each iteration of the optimization and couples the stiffness of each element with the density. In effect, the optimization removes elements from your model by giving them a mass and stiffness that is small enough to ensure they no longer participate in the overall response of the structure. The model with the revised material properties is then analyzed by Abaqus.

  For a shape optimization, the displacements of the surface nodes in the design area are the design variables. During the optimization, the Optimization module either moves a node outward (growth) or inward (shrinkage) or leaves the position unchanged (neutral). Restrictions influence the amount a surface node can move and the direction in which it can move. The optimization directly modifies only the position of the corner nodes of elements; the Optimization module interpolates the displacement of midside nodes from the movement of the corner nodes.

  For a sizing optimization, the thicknesses of the shell elements in the design area are the design variables. The Optimization module can adjust the thickness of individual shell elements, or you can require clustering—the simultaneous modification of shell thicknesses in specific areas.

  For a bead optimization, the displacements of the nodes of the shell elements that form the stiffening beads in the design area are the design variables. Restrictions limit the amount a node can move and the direction in which it can move.

- **Design cycle:** Optimization is an iterative design process that updates the design variables, executes an Abaqus analysis of the modified model, and reviews the results to determine if an optimized solution has been reached. Each optimization iteration is called a design cycle.

- **Optimization task:** An optimization task contains the definition of your optimization, such as the design responses, objectives, constraints, and geometric restrictions. To run an optimization, you execute an optimization process. An optimization process refers to an optimization task.

- **Design responses:** The inputs to the optimization are called the design responses. Design responses can be read directly from the Abaqus output database (`.odb`) file; for example, stiffness, stress, eigenfrequencies, and displacements. Alternatively, the Optimization module can read data from the output database file and calculate the design responses from your model; for example, its weight, center of mass, or relative displacements.

  A design response is associated with a region of your model; however, it consists of a single scalar value, such as the maximum stress within a region or the total volume of the model. In addition, a design response can be associated with a particular step or load case.

- **Objective functions:** Objective functions define the objective of the optimization. An objective function is a single scalar value extracted from a design response, such as the maximum displacement or the maximum stress. An objective function can be formulated from multiple design responses. If you specify that the objective functions minimize or maximize the design responses, the Optimization module calculates the objective function by adding each of the values determined from the design responses. In addition, if you have multiple objective functions, you can use a weighting factor to scale their influence on the optimization.

- **Constraints:** Constraints are also a single scalar value extracted from a design response; however, a constraint cannot be derived from a combination of design responses. Constraints restrict the value of a design response; for example, you can specify that the volume must be reduced by 45% or the absolute displacement in a region must not exceed 1 mm. You can also apply manufacturing and geometric constraints that are independent of the optimization; for example, a structure must be able to be cast or stamped or the diameter of a bearing surface cannot be changed.
- **Stop conditions:** A global stop condition defines the maximum number of iterations an optimization can perform. A local stop condition specifies that the optimization should end when a local minimum (or maximum) has been reached.

Structural optimization with Abaqus/CAE

The following steps are required to incorporate structural optimization into your Abaqus/CAE model:

- You create an Abaqus model that can be optimized. For example, the design area must include only supported elements and materials. See "Creating Abaqus optimization models," Section 13.2.3.
- You create an optimization task. See "Creating and configuring an optimization task," Section 18.6 of the Abaqus/CAE User's Guide.
- You create design responses. See "Design responses," Section 13.2.1.
- You use the design responses to create objective functions and constraints. See "Objectives and constraints," Section 13.2.2.
- You create an optimization process and submit it for analysis. See "What is an optimization process?," Section 19.5.1 of the Abaqus/CAE User's Guide.

Based on the definition of the optimization task and the optimization process, the Optimization module iteratively:

- prepares the design variables (element densities or surface node positions) and updates the Abaqus finite element model, and
- executes an Abaqus/Standard analysis.

These iterations or design cycles continue until either:

- the maximum number of design cycles is reached, or
- the specified stop conditions are reached.

Figure 13.1.1–1 shows the interaction of Abaqus and the optimization process.

Topology optimization

Topology optimization starts with an initial design (the original design area), which also contains any prescribed conditions (such as boundary conditions and loads). The optimization process determines a new material distribution by changing the density and the stiffness of the elements in the initial design while continuing to satisfy the optimization constraints, such as the minimum volume or the maximum displacement of a region. In addition, you can apply a number of manufacturing constraints that ensure

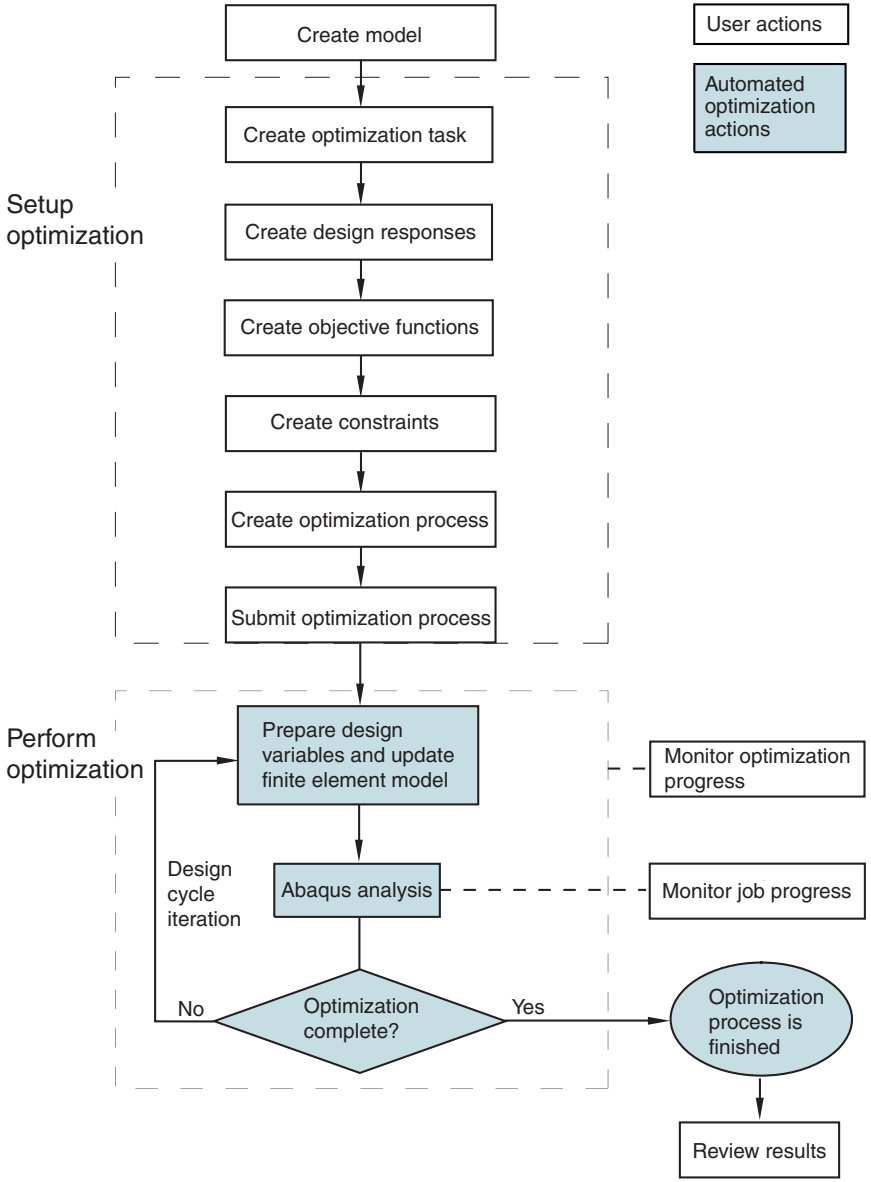


Figure 13.1.1–1 User actions and automated Abaqus/CAE actions in the optimization process.

the proposed design can be created using standard production processes, such as casting and stamping. You can also freeze selected regions and apply member size, symmetry, and coupling constraints.

Figure 13.1.1–2 show the progression of a topology optimization of an automotive control arm during 17 design cycles. The objective function in the optimization is trying to minimize the maximum strain energy calculated from all the elements in the arm, in effect maximizing the structural stiffness of the arm. The constraint is forcing the optimization to reduce the volume by 57% from the initial value. During the optimization the density and the stiffness of the elements in the center of the arm are reduced so that the elements are, in effect, "removed" from the analysis. However, the elements are still present, and they could play a role in the analysis if their density and stiffness increase as the optimization continues. A geometry restriction forces the optimization to create a model that could be cast and removed from a mold—the Optimization module cannot create voids and undercuts.

An example of using topology optimization is provided in "Topology optimization of an automotive control arm," Section 11.1.1 of the Abaqus Example Problems Guide. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

General versus condition-based topology optimization

Topology optimization supports two algorithms—the general algorithm, which is more flexible and can be applied to most problems, and the condition-based algorithm, which is more efficient but has limited capabilities. By default, the Optimization module uses the general algorithm; however, you can choose which algorithm to use when you create the optimization task. Each algorithm has a different approach for determining the optimized solution.

Algorithms

General topology optimization uses an algorithm that adjusts the density and stiffness of the design variables while trying to satisfy the objective function and the constraints. The general algorithm is partly described in Bendsøe and Sigmund (2003). In contrast, condition-based topology optimization uses a more efficient algorithm that uses the strain energy and the stresses at the nodes as input data and does not need to calculate the local stiffness of the design variables. The condition-based algorithm was developed at the University of Karlsruhe, Germany and is described in Bakhtiary (1996).

Elements with intermediate densities

The general algorithm generates intermediate elements in the final design (their relative density is between zero and one). In contrast, the condition-based optimization algorithm generates elements in the final design that are either void (their relative density is very close to zero) or solid (their relative density is equal to one).

Number of optimization design cycles

The number of design cycles used by the general optimization algorithm is unknown before the optimization starts, but normally the number of design cycles is between 30 and 45. The condition-based optimization algorithm is more efficient and searches for a solution until it reaches the maximum number of optimization design cycles (15 by default).

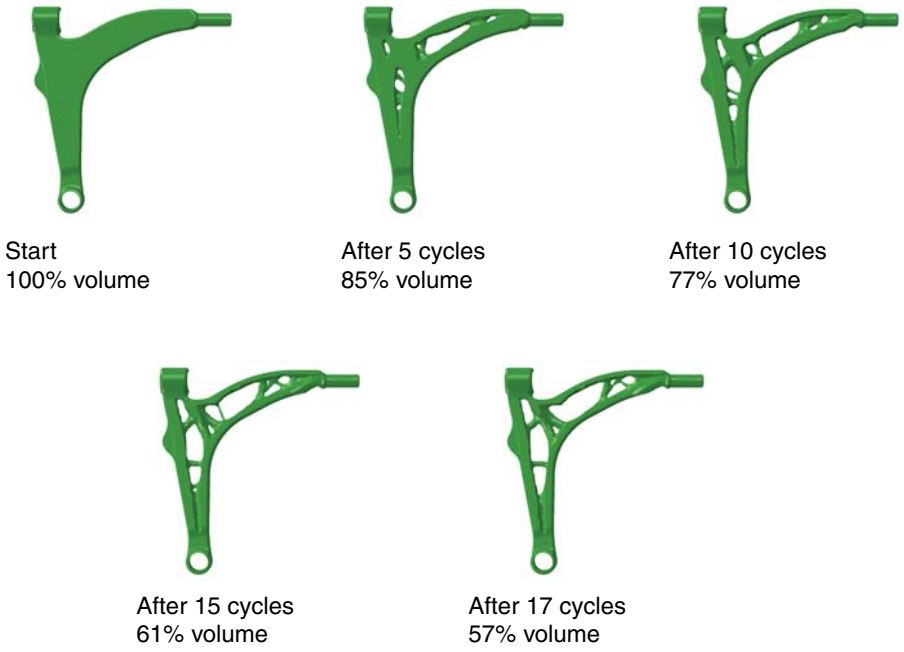


Figure 13.1.1–2 The progression of a topology optimization.

Analysis types

The general algorithm supports the responses of linear and nonlinear static and linear eigenfrequency finite element analyses. Both algorithms support geometrical nonlinearities and contact, and many nonlinear materials are also supported.

Furthermore, prescribed displacements are allowed in the Abaqus model for static topology optimization. However, prescribed displacements are not allowed for modal analysis. You can use topology optimization on a structure that uses a composite material; however, the individual laminates of a composite material cannot be modified using topology optimization. For example, you cannot change the orientation of the fibers.

Objective functions and constraints

The general topology optimization algorithm can use one objective function and several constraints, where the constraints are all inequality constraints. A variety of design responses can be used to define the objective and the constraints, such as strain energy, displacements and rotations, reaction and internal forces, eigenfrequencies, and material volume and weight. The condition-based topology optimization

algorithm is more efficient; however, it is less flexible and supports only strain energy (a measure of stiffness) as the objective function and the material volume as an equality constraint.

Tables in "The available design responses for each type of optimization" in "Design responses," Section 13.2.1 list the available design responses for condition-based and general topology optimization and describe which design responses can be used as objectives and/or constraints.

Shape optimization

Shape optimization uses an algorithm that is similar to the algorithm used by condition-based topology optimization. You use shape optimization at the end of the design process when the general layout of a component is fixed, and only minor changes are allowed by repositioning surface nodes in selected regions. A shape optimization starts with a finite element model that needs minor improvement or with the finite element model generated by a topology optimization.

Typically, the objective of a shape optimization is to minimize stress concentrations using the results of a stress analysis to modify the surface geometry of a component until the required stress level is reached. Shape optimization tries to position the surface nodes of the selected region until the stress across the region is constant (stress homogenization). Figure 13.1.1–3 shows a region at the base of a connecting rod where the surface nodes have been moved by shape optimization to reduce the effect of a stress concentration.

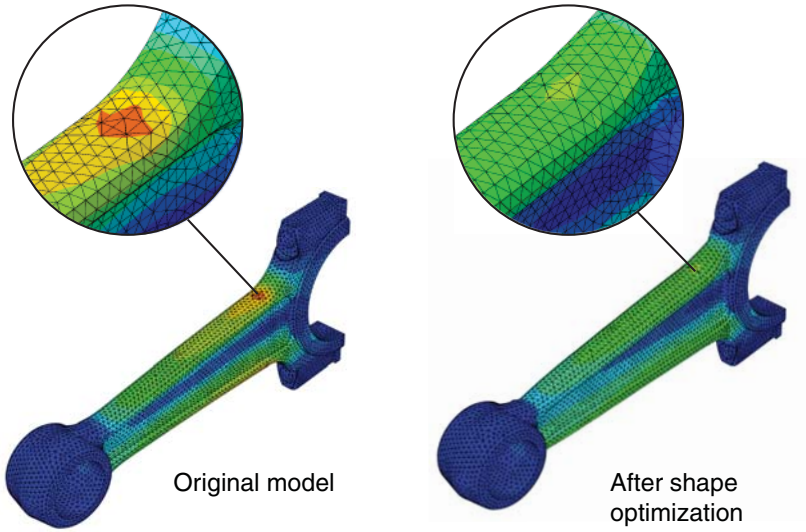


Figure 13.1.1–3 The effect of shape optimization.

A table in "The available design responses for each type of optimization" in "Design responses," Section 13.2.1 lists the available design responses for shape optimization and describes which design responses can be used as objectives and/or constraints. In addition, you can apply a number of manufacturing geometric restrictions that ensure the proposed design can continue to be produced using

casting or stamping processes. You can also freeze selected regions and apply member size, symmetry, and coupling restrictions.

An example of using shape optimization is provided in "Shape optimization of a connecting rod," Section 11.2.1 of the Abaqus Example Problems Guide. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying mesh smoothing to a shape optimization

During a shape optimization, the Optimization module modifies the surface of your model. If the Optimization module modifies only the surface nodes without adjusting the inner nodes, the layer of surface elements becomes distorted. Therefore, the results of the Abaqus analysis are no longer reliable, and the quality of the optimization suffers. To maintain the quality of the surface elements, the Optimization module can apply mesh smoothing to selected regions, which adjusts the position of the inner nodes in relation to the movement of the surface nodes. You must have a good quality finite element mesh before you start the shape optimization, especially in areas where you expect the shape to change.

The Optimization module can apply mesh smoothing to the standard continuum elements, such as triangular, quadrilateral, and tetrahedral elements. Other element types are ignored during the mesh smoothing. You can specify the relative quality of the smoothed mesh, and you can specify the range of angles (quadrilateral and triangular elements) or the range of aspect ratios (tetrahedral elements) that define an element that is considered good quality. Elements that are considered poor are given a quality rating. The poorer an element is rated, the greater the consideration it will be given in improving the element quality.

Mesh smoothing can be computationally expensive. The mesh smoothing algorithm is element-based; and computing time increases in regions with many elements with limited degrees of freedom, such as regions with small tetrahedral elements. You should apply mesh smoothing only to regions where you expect the surface nodes to move—regions that will benefit from mesh smoothing. The nodes in the regions to which you apply mesh smoothing must be free to move. For example, you should not apply mesh smoothing to fixed nodes or to frozen regions.

You can apply limits to the result of mesh smoothing by applying minimum and maximum growth restrictions to the selected region. See "Creating a growth restriction" in "Creating a geometric restriction in a shape optimization," Section 18.10.3 of the Abaqus/CAE User's Guide, for more information.

Mesh smoothing can be applied to regions that are included in the design region and to regions that are outside the design region. In particular, you can prevent element distortion by applying mesh smoothing to the region of transition between the design region and the rest of your model. However, the design region must be contained within the region to which you apply mesh smoothing.

Free surface nodes are defined as the nodes that lie outside the design area and are not included in a geometric restriction. By default, the Optimization module fixes all degrees of freedom of all of the free surface nodes, and they are not modified during the mesh smoothing operation. Alternatively, you can choose to allow the free surface nodes to move along with a specified number of layers of nodes adjacent to the nodes in the design area. (A "layer" of nodes is created from only corner nodes; midside nodes are not taken into consideration.)

You should allow free surface nodes to move in regions that are adjacent to the design area to create a smooth transition between optimized and non-optimized regions. However, in some cases you will want free surface nodes to remain fixed; for example, on a planar face that does not play a role in your optimized model and must remain planar.

By default, a constrained Laplacian mesh smoothing algorithm is used. Alternatively, if you have a relatively small model (less than 1000 nodes in the mesh smooth area), you can select a local gradient mesh smoothing algorithm. In each iteration the local gradient mesh smoothing algorithm identifies the elements with the worst element quality and improves them by displacing the nodes. Local gradient mesh smoothing usually generates elements having the optimal shape, where the optimal is defined as the ratio of the element volume (area for shell elements) to the corresponding power of its diameter. For larger models the local gradient mesh smoothing algorithm tends to stop before the optimal mesh quality is reached because the computation time becomes excessive. When the mesh smoothing ends prematurely, only the elements with the worst element quality will be smoothed.

Sizing optimization

Similar to shape optimization, you use sizing optimization at the end of the design process when the general layout of a component is fixed, and only minor changes are allowed by changing the shell thickness in selected regions. A sizing optimization starts with a finite element model that needs minor improvement or with the finite element model generated by a topology optimization. The Optimization module uses a general algorithm for solving sizing optimization problems based on the method of moving asymptotes, which is described in Svanberg (1985).

Typically, the objective of a sizing optimization is to maximize the stiffness of a component while satisfying a weight objective.

A table in "The available design responses for each type of optimization" in "Design responses," Section 13.2.1 lists the available design responses for sizing optimization and describes which design responses can be used as objectives and/or constraints. In addition, you can apply geometric restrictions that freeze selected areas and apply member size and symmetry constraints. You can also provide upper and lower bounds for the shell element thickness and group regions into "clusters" of equal shell thickness. You can use the Visualization module to view the variation in shell thickness after a sizing optimization.

An example of using sizing optimization is provided in "Sizing optimization of a gear shift control holder," Section 11.3.1 of the Abaqus Example Problems Guide. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying clustering to a sizing optimization

When you configure a sizing optimization, you can specify that selected regions should contain "clusters" of shell elements of equal thickness. You can use clustering to generate strengthening ribs or rings in the sheet metal structure you are optimizing or to define borders between regions of equal thickness. Clustered regions can be reproduced in manufacturing using sheets of constant thickness; for example, a vehicle "body in white" formed by welding and stamping individual sheet metal structures. To allow for

maximum design flexibility, you should first optimize your structure without specifying clustering and use the initial design to decide which regions to cluster in your final optimization.

Figure 13.1.1–4 shows a sheet metal arm that was optimized with "free" sizing optimization that allows the thickness to be modified in the design area regardless of the thickness of adjacent shell elements. Figure 13.1.1–5 shows the same model with a clustered ring of shell elements of equal thickness in the design area. The part generated by free optimization is stiffer than the part generated by clustered optimization, but the part generated by free optimization would be impractical to manufacture.

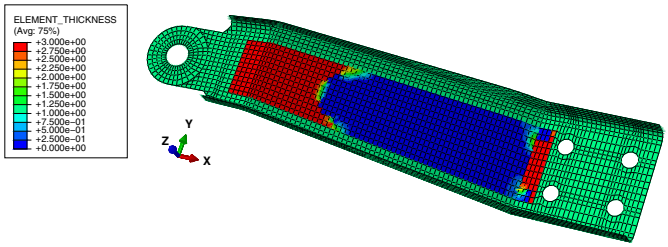


Figure 13.1.1–4 Absolute value of shell thickness after free optimizing.

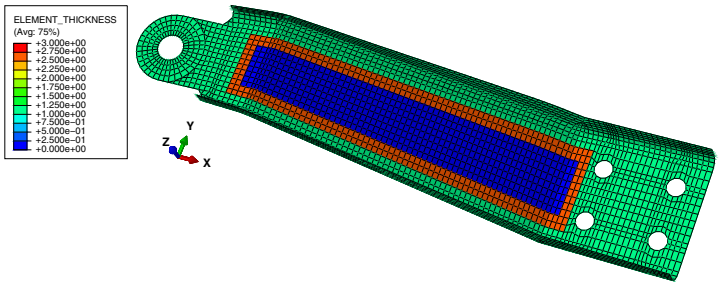


Figure 13.1.1–5 Absolute value of shell thickness after optimizing with clustering.

Bead optimization

Bead optimization adds stiffening beads to a shell structure to increase the moment of inertia, which leads to a greater stiffness or higher eigenfrequencies. The resulting beads are easy to reproduce in a sheet metal stamping process and add no mass and little cost to the finished product.

Figure 13.1.1–6 shows the von Mises stress in a plate that is fixed at the edges and has a pressure load applied to the center. The image on the left shows the original configuration. The image on the right shows how condition-based bead optimization has introduced circular stiffening beads and significantly reduced the stress at the center of the plate.

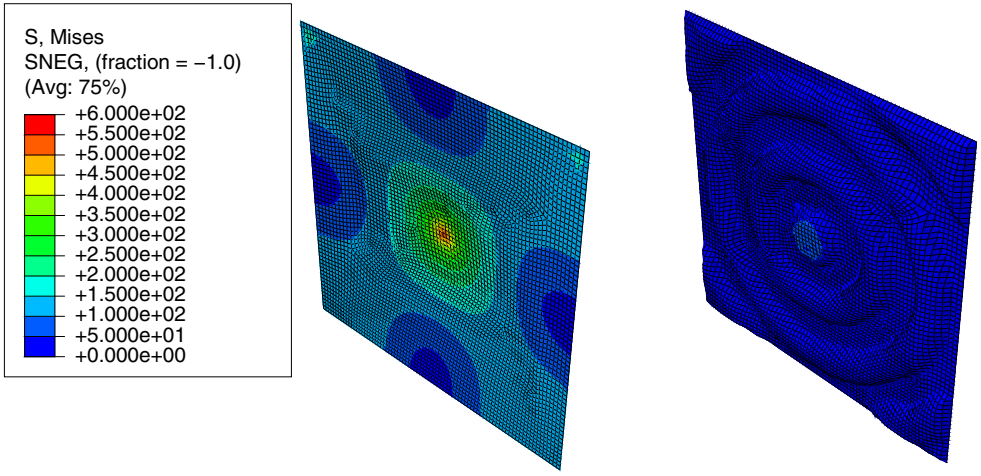


Figure 13.1.1–6 The effect of condition-based bead optimization.

Typically, the objective of a bead optimization is to maximize the stiffness of a component or to minimize the displacement of critical nodes. Bead optimization does not always result in improved mechanical behavior. The following conditions should be true for the design problem to be suited to bead optimization:

- the design area should be subjected to bending or a bending mode, and
- the design area should not be in a predominately membrane stress state. The addition of a bead in this case may make the structure softer.

An example of using bead optimization is provided in "Bead optimization of a plate," Section 11.4.1 of the Abaqus Example Problems Guide. The example includes a Python script that you can run from Abaqus/CAE to create the model and configure the optimization.

Applying mesh smoothing to a condition-based bead optimization

During a bead optimization, the Optimization module moves the nodes of the shell elements in the direction of the shell normal. To maintain the quality of the shell elements during a condition-based bead optimization, the Optimization module can apply mesh smoothing to the design area to limit the displacement of the nodes. Two forms of mesh smoothing are provided—node smoothing and element smoothing. Node smoothing prevents large displacements by limiting the displacement of the nodes at the boundary between the design area and the rest of the model or where active design variable constraints are restraining the displacement of nodes. Element smoothing prevents nodes from overlapping in regions of high curvature where the bead is being formed. You must have a good quality finite element mesh before you start the bead optimization, especially in areas where you expect beads to be formed.

General versus condition-based bead optimization

Bead optimization supports two algorithms—the general algorithm, which is more flexible and can be applied to most problems but does not generate a distinct bead pattern, and the condition-based algorithm, which is more efficient at creating beads but has limited capabilities. By default, the Optimization module uses the condition-based algorithm; however, you can choose which algorithm to use when you create the optimization task. Each algorithm has a different approach for determining the optimized solution.

Figure 13.1.1–7 shows the plate model optimized with general bead optimization. While the stresses have been similarly reduced, the seemingly random bead structure that is generated would be hard to manufacture, unlike the regular circular beads generated by the condition-based bead optimization.

Algorithms

General, or sensitivity-based, bead optimization uses an algorithm that makes it possible to define very complex optimization tasks. It has been shown in industrial size examples that the method is very powerful and attractive, especially for dynamic problems. The algorithm does not use a bead filter, and the results of the optimization are not necessarily a distinct bead pattern. In contrast, condition-based bead optimization uses a special bending hypothesis developed at the Karlsruhe University, Institute of Product Engineering (IPEK) and described in Emmrich (2004). The algorithm determines the orientation of the maximum bending stress for each point in the design and uses filters to generate beads along the bending trajectories.

Controlling the bead height and width

For a condition-based bead optimization, you directly limit the bead height by creating a bead height design response that is referenced by a constraint. You can enter a value for the bead width, or you can allow Abaqus/CAE to calculate the width from the following:

bead width $= 2 \times \max(2 \times$ height of bead, $3.2 \times$ average element edge length$)$

Typically, the bead height is small in comparison to the element size, in which case the bead width is a function of the average element length. Otherwise, if the bead height is large in comparison to the element size, the bead width is a function of the bead height.

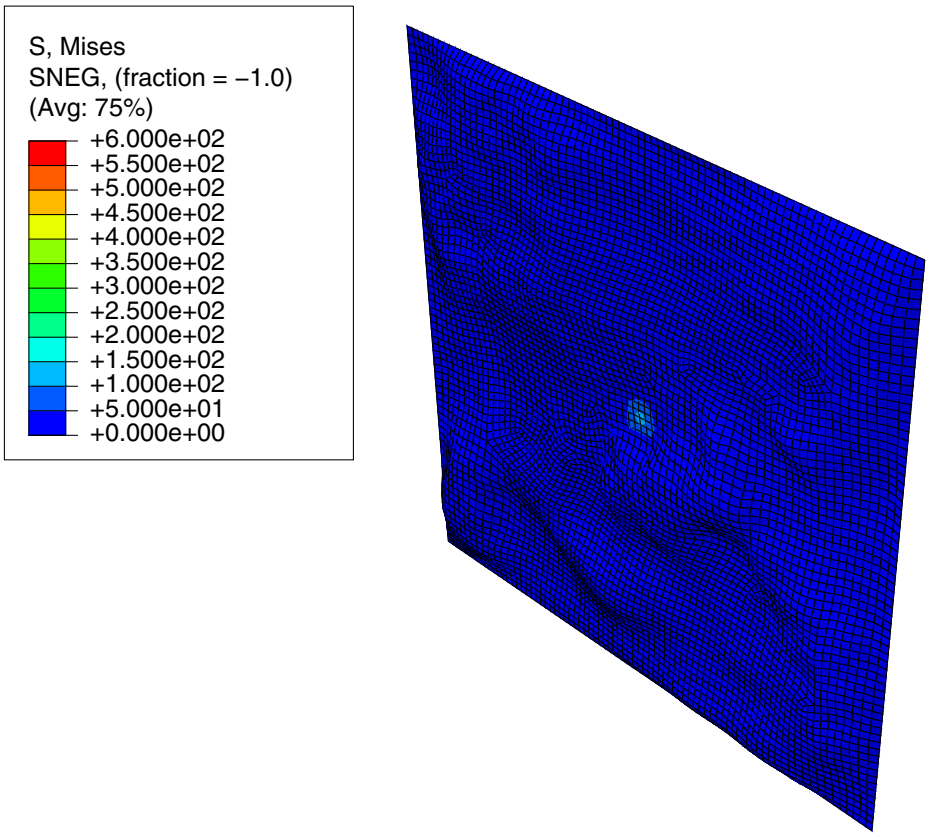


Figure 13.1.1–7 The effect of general bead optimization.

For a general bead optimization, you must limit the bead height by creating geometric restrictions that limit the displacement in the growth and shrink directions. The growth and shrink geometric restrictions are required because they are the only restrictions on the displacement of the nodes. You cannot control the bead width for a general bead optimization.

Number of optimization design cycles

The number of design cycles used by the general algorithm is unknown before the optimization starts, but the general algorithm normally requires about 20 design cycles to converge on a solution. The condition-based algorithm is more efficient and always uses three design cycles.

Analysis types

The general algorithm supports the responses of linear static, linear eigenfrequency, and frequency response finite element analyses. The condition-based algorithm is recommended only for static analyses; however, the algorithm supports all the Abaqus analysis types that generate a stress tensor.

Objective functions and constraints

The general bead optimization algorithm can use one objective function and several constraints, where the constraints are all inequality constraints. A variety of design responses can be used to define the objective and the constraints, such as strain energy, displacements and rotations, reaction and internal forces, eigenfrequencies, and moment of inertia. The condition-based bead optimization algorithm is more efficient; however, it is less flexible and supports only strain energy (a measure of stiffness) or eigenfrequency as the objective function and the bead height as an equality constraint.

Tables in "The available design responses for each type of optimization" in "Design responses," Section 13.2.1 list the available design responses for general and condition-based bead optimization and describe which design responses can be used as objectives and/or constraints. In addition, you can apply geometric restrictions that freeze selected areas and apply bead size and symmetry constraints.

13.2 Optimization models

- “Design responses,” Section 13.2.1
- “Objectives and constraints,” Section 13.2.2
- “Creating Abaqus optimization models,” Section 13.2.3

13.2.1 DESIGN RESPONSES

Product: Abaqus/CAE

References

- "Structural optimization: overview," Section 13.1.1
- "Configuring design responses," Section 18.7 of the Abaqus/CAE User's Guide

Overview

A design response:

- is a single scalar value, such as the volume of your structure;
- is calculated by the Optimization module by reading results and model data from the output database file;
- can be referred to from objective functions and constraints (for example, you can create an objective function that tries to minimize the displacement at a node or a constraint that forces the weight of the structure to be reduced by at least 50%); and
- is available only for certain analysis procedures (for example, you must perform an eigenvalue extraction analysis if you select a design response that tries to maximize the lowest eigenfrequencies).

Design responses

The Optimization module provides a number of different design responses.

Bead height

The bead height has the most significant effect on the stiffness of a shell structure that is being optimized with bead optimization. Usually, the greater the bead height the greater the stiffness, although manufacturing restrictions will limit how deep you can draw a bead. When the optimization starts, the bead height is initially zero. As the optimization proceeds, it moves the nodes in the shell normal direction until they reach the bead height. A bead height design response is available only as a constraint in a condition-based beam optimization.

Abaqus/CAE Usage: Optimization module: **Task→*condition-based bead task***, **Design Response→Create**: **Single-term**, **Variable: Bead height**

Center of gravity

You can use the center of gravity of a selected region as a design response in an optimization. You can choose the center of gravity in the three principal directions:

$$x_g = \frac{\int \rho x dV}{\int \rho dV},$$
$$y_g = \frac{\int \rho y dV}{\int \rho dV}, \text{and}$$
$$z_g = \frac{\int \rho z dV}{\int \rho dV}$$

When the Optimization module calculates the center of gravity, the elements are scaled with the current relative density defined in your Abaqus model.

For example, you might want to constrain the center of gravity in the *Y*-direction so that it remains within a minimum and maximum range during the optimization. The design response can consider the center of gravity of the whole model or a region of the model.

If you select a local coordinate system, the Optimization module uses both the direction of the axes and the position of the origin to recalculate the center of gravity. The Optimization module applies the global coordinate system if you do not select a local coordinate system.

When the Optimization module calculates the center of gravity, it treats shell and membrane regions as three-dimensional regions by applying the thickness of the region. The Optimization module calculates the center of gravity using only the element types that are supported by topology optimization. As a result, the center of gravity calculated by the Optimization module might not be the same as the center of gravity calculated by Abaqus/Standard or Abaqus/Explicit; for example, if your model contains wire regions.

Abaqus/CAE Usage: Optimization module: **Task→*general topology or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Center of gravity**

Damage

Typically, you use shape optimization to modify the surface geometry of a component to minimize stress concentrations. In most cases reducing the stress levels leads to a significant increase in durability. However, it is possible that the regions of peak stress identified by a static analysis differ from the regions of maximum damage identified from a durability (or damage) analysis, and using shape optimization alone to modify the surface geometry may decrease the durability. To avoid this situation, you can incorporate a durability solver in the optimization loop to ensure that you are both reducing stress levels and increasing durability.

You can create a design response that uses damage only in a shape optimization, and it can be used only as an objective function—minimization of the maximum damage in the critical areas. The Optimization module provides damage values as equivalent stress values. You can create a damage design response where the durability analysis runs only once in the optimization loop. Alternatively, you can create a damage design response when the durability analysis contains load case information and runs multiple times in the optimization loop.

Abaqus/CAE Usage: Optimization module: **Task→*shape task***, **Design Response→Create**: **Single-term**, **Variable: Damage (single)**

Optimization module: **Task→*shape task***, **Design Response→Create**: **Single-term**, **Variable: Damage (multiple)**

Displacement and rotation

In most optimization problems you will use displacement and/or rotation to define your objective function or constraints. For example, the maximum displacement of a vertex could be either an objective or a constraint of an optimization. The performance of the optimization is improved if you apply displacements and rotations to only vertices or to small regions. In addition, performance is improved if you assign regions that are used to define displacements or reactions as frozen regions (the Optimization module will not remove elements from frozen regions during the optimization).

Table 13.2.1–1 lists the available displacement and rotation variables.

Abaqus/CAE Usage: Optimization module: **Task→*general topology, general bead, or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Displacement**

Energy stiffness measure

The energy stiffness is a measure that has no physical meaning but can be used as an objective function or a constraint in an optimization to optimize the stiffness of a structure that is subjected to both external loading and prescribed displacements.

To optimize the stiffness of a structure with only external loading, the strain energy should be minimized:

$$\min\left(\frac{P\cdot u}{2}\right),$$

where P is the external loading and u is the resulting deflection of the loaded nodes. If only external loading is present, the energy stiffness measure is equal to the total strain energy, also called the compliance.

In contrast, if a load case is driven by prescribed displacements, the elastic energy, or compliance, will decrease only if the structure is made softer. To optimize a structure with only prescribed displacements, the strain energy should be maximized:

$$\max\left(\frac{R\cdot u*}{2}\right),$$

where u is the prescribed displacement at the nodes and R is the resulting reaction force at the displaced nodes. If only prescribed displacements are present, the energy stiffness measure is equal to the negative of the total strain energy.

The strain energy with both external loads and prescribed displacements is given as

$$\theta=\frac{P\cdot u}{2}+\frac{R\cdot u*}{2}.$$

The energy stiffness measure is used only for optimization (it has no physical meaning) and is given as

Table 13.2.1–1 Displacement and rotation variables for a general and sizing topology optimization.

	Displacement	**Rotation**
i-direction	u_i	θ_i
Absolute	$\sqrt{u_i^2 + u_j^2 + u_k^2}$	$\sqrt{\theta_i^2 + \theta_j^2 + \theta_k^2}$
Absolute in *i*-direction	$\sqrt{u_i^2}$	$\sqrt{\theta_i^2}$

$$\min\left(\frac{P \cdot u}{2} - \frac{R \cdot u*}{2}\right).$$

You cannot use the energy stiffness measure as a design response in a model that is experiencing thermal loading or mass-dependent loading, such as gravity. An energy stiffness design response must be applied to the entire model. When energy stiffness is used as an objective function, you must select a target that tries to minimize the sum of the weighted differences between the design response and the reference value, regardless of whether an external load and/or prescribed displacements are being applied to the structure.

Abaqus/CAE Usage: Optimization module: **Task→*general topology or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Energy stiffness measure**

Equivalent strain

If your model is undergoing large deformations, a measure of the stress is not always a good indicator of the model's response. For example, a structure undergoing plastic deformation will, for an ideal plastic material, experience a large constant stress over the plastic area. In these circumstances a measure of the strain is a more reliable indicator of the model's response. You can choose from the following equivalent strains:

- Elastic
- Plastic
- Total (the sum of the elastic and plastic)

Abaqus/CAE Usage: Optimization module: **Task→*shape task***, **Design Response→Create**: **Single-term**, **Variable: Strain**

Equivalent stresses

Equivalent stresses are the most commonly used objective function of a shape optimization. All of the stress values that are calculated by the Optimization module, whether nodal or from Gauss points or elements, are interpolated to the nodes. For example, you can try to optimize your model with an objective function that tries to minimize the maximum von Mises stresses in a region with stress concentrations or tries to minimize contact pressure in a region with contact. The Optimization module considers only the maximum value of an equivalent stress within a region. The Optimization module

issues warnings for nodes that do not have the appropriate stress values. For example, if you select an objective response of contact stress, the Optimization module issues warnings about nodes of elements that are not in contact. If your Abaqus model contains multiple load cases, the design response is evaluated by summing the stress values from each load case.

You can choose from the following equivalent stresses:

- von Mises
- Maximum principal and absolute maximum principal
- Minimum principal and absolute minimum principal
- Second principal
- Beltrami
- Drucker Prager
- Galilei
- Kuhn
- Mariotte
- Mises
- Sandel
- Sauter
- Tresca

You can choose from the following equivalent contact stresses:

- Contact stress pressure
- Total shear contact stress
- Shear contact stress in the 1-direction
- Shear contact stress in the 2-direction
- Total contact stress

Abaqus/CAE Usage: Optimization module: **Task→*shape task***, **Design Response→Create**: **Single-term**, **Variable: Stress** or **Contact stress**

Internal forces and moments

You can use nodal internal forces and moments of the whole model or a region of the model as an objective function or a constraint in a general topology or sizing optimization.

Table 13.2.1–2 lists the available nodal internal force and moment variables.

Table 13.2.1–2 Nodal internal force and moment variables for a general and sizing topology optimization for the elements e attached to the nodes i.

	Nodal internal force	**Nodal internal moment**
i-direction	$\sum K_e u_i$	$\sum K_e u_i$
Absolute	$\lvert\sum K_e u_i\rvert$	$\lvert\sum K_e u_i\rvert$
Absolute in *i*-direction	$\sqrt{u_i^2}$	$\sqrt{u_i^2}$

You must use the global coordinate system with an absolute internal force or with absolute internal moment. Your structure must have stiffness in the direction of the force used in the optimization; otherwise, the internal force will be zero in this direction.

Abaqus/CAE Usage: Optimization module: **Task→*general topology, general bead, or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Internal force** or **Internal moment**

Modal eigenfrequency analysis

Modal eigenvalues are the simplest dynamic responses in structural analysis. Typical uses of data from an eigenfrequency analysis during a topology optimization include the following:

- maximize the lowest eigenfrequencies,
- maximize a selected eigenfrequency,
- constrain an eigenfrequency to be higher or lower than a given value,
- maximize or minimize an eigenfrequency at a certain mode, and
- perform a bandgap optimization that force modes away from a certain frequency.

The Optimization module supports two approaches for optimizing the eigenfrequencies:

- single eigenfrequencies from modal analysis and
- the Kreisselmaier-Steinhauser formulation.

The Kreisselmaier-Steinhauser formulation is the more efficient of the two approaches and should be used whenever possible. The only advantage of optimizing single eigenfrequencies is that you can use the sum of the eigenfrequencies as a constraint in a general topology or sizing optimization, which you cannot do with the Kreisselmaier-Steinhauser formulation.

When you are trying to maximize the lowest eigenfrequency, it is recommended that you consider not only the first eigenfrequency but also at least the next two highest natural frequencies. During the optimization, the various natural frequencies are weighted by their distance from the lowest natural frequency—the closer a natural frequency approaches the first natural frequency during the optimization, the more it is weighted. You should use the Kreisselmaier-Steinhauser eigenvalue formulation if you are trying to maximize the lowest eigenfrequency or, in particular, if you are trying to maximize more

than one of the lowest eigenfrequencies. You do not need to use mode tracking if you are using the Kreisselmaier-Steinhauser formulation to maximize the lowest eigenfrequency, but you should use mode tracking for the higher modes because the modes might switch. For example, while the model is being optimized, the frequency of the first mode is maximized and the second eigenmode can become the mode with the lowest frequency.

Abaqus/CAE Usage: Optimization module: **Task→*general topology, general bead, or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Eigenfrequency from modal analysis** or **Eigenfrequency calculated with Kreisselmaier-Steinhauser formula**

Moment of inertia

You can use a moment of inertia design response in an optimization to minimize the rotational inertia about a selected axis. You can use the sum of the moment of inertia of the whole model or a region of the model as an objective function or a constraint in a general topology or sizing optimization.

You can choose the moment of inertia in the three principal directions or the three principal planes:

$$
\begin{aligned}
I_x &= \int \rho \left(y^2 + z^2\right) dV, \\
I_y &= \int \rho \left(x^2 + z^2\right) dV, \\
I_z &= \int \rho \left(x^2 + y^2\right) dV, \\
I_{xy} &= I_{yx} = -\int \rho xy dV, \\
I_{xz} &= I_{xz} = -\int \rho xz dV, \\
I_{yz} &= I_{yz} = -\int \rho yz dV.
\end{aligned}
$$

If you select a local coordinate system, the Optimization module uses the direction of the axes to recalculate the center of gravity. The Optimization module applies the global coordinate system if you do not select a local coordinate system.

When the Optimization module calculates the moment of inertia, it treats shell and membrane regions as three-dimensional regions by applying the thickness of the region. The Optimization module calculates the moment of inertia using only the element types that are supported by topology optimization. As a result, the moment of inertia calculated by the Optimization module might not be the same as the moment of inertia calculated by Abaqus/Standard or Abaqus/Explicit; for example, if your model contains beam or truss elements (wire regions in Abaqus/CAE).

The moment of inertia with respect to any two orthogonal axes is zero if you have selected either of the axes to be an axis of symmetry.

Abaqus/CAE Usage: Optimization module: **Task→*general topology, general bead, or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Moment of inertia**

Nodal strain energy density

The nodal strain energy density, $u = \sigma_{ij}\varepsilon_{ij}$, is a local point-wise strain energy that can provide a better representation of failure than stresses in nonlinear materials.

Abaqus/CAE Usage: Optimization module: **Task→*shape task***, **Design Response→Create**: **Single-term**, **Variable: Strain energy density**

Reaction forces and moments

Nodal reaction forces and moments can be used as a design response only in general and sizing topology optimization. As with displacements, the performance of the optimization is improved if you apply reaction forces or moments to only vertices or to small regions and assign those regions as frozen regions (the Optimization module will not remove elements during the optimization).

Table 13.2.1–3 lists the available nodal reaction force and moment variables.

Table 13.2.1–3 Nodal reaction force and moment variables for a general and sizing topology optimization for the elements e attached to the nodes i.

	Nodal reaction force	**Nodal reaction moment**
i-direction	$\sum K_e u_i$	$\sum K_e u_i$
Absolute	$\lvert\sum K_e u_i\rvert$	$\lvert\sum K_e u_i\rvert$
Absolute in *i*-direction	$\sqrt{u_i^2}$	$\sqrt{u_i^2}$

You cannot use a reference coordinate system with an absolute reaction force or with an absolute reaction moment. Your structure must have stiffness in the direction of the force used in the optimization; otherwise, the reaction force will be zero in this direction.

Abaqus/CAE Usage: Optimization module: **Task→*general topology, general bead, and sizing task***, **Design Response→Create**: **Single-term**, **Variable: Reaction force** or **Reaction moment**

Scaled centroidal von Mises stress

You can use the scaled element centroidal von Mises stress of the whole model or a region of the model as an objective function in an optimization. You must avoid regions with stress singularities caused by external loads or boundary conditions.

The scaled element centroidal von Mises stress is defined as

$$= \text{Max} \left| \frac{\left(\sigma_{vMises}\right)^2}{\left(f\left(\rho_i\right)\sigma_y\right)^2} \cdot \sigma_y \right|,$$

where σ_{vMises} is the element centroidal von Mises stress, σ_y is the reference stress, and ρ_i is a factor for interpolating the stresses of elements that have a reduced current relative density because of the topology optimization. The weighting factor and the interpolation are required for convergence during the optimization.

The von Mises stress is calculated at the centroid of the element to avoid stress singularities that might be present in the initial model or might appear in an optimized structure before it is smoothed. You cannot compare the scaled element centroidal von Mises stress with the von Mises stress calculated by Abaqus. The two measures are equal only when the element is solid and has a relative density of 1.0.

You can provide the reference stress when you create the objective function, or the Optimization module can calculate the reference stress during the initial optimization iteration. If you provide the reference stress, the value should not be too low or numerical singularities will result. The reference stress is given as

$$\sigma_{\text{y}} = \text{Min} \left\{ \alpha_1 \text{Max} \left| \frac{\left(\sigma_{vMises}\right)^2}{\left(f\left(\rho_i\right)\sigma_{ref_1}\right)} . \sigma_{ref_1} \right| + \alpha_2 \text{Max} \left| \frac{\left(\sigma_{vMises}\right)^2}{\left(f\left(\rho_i\right)\sigma_{ref_2}\right)^2} . \sigma_{ref_2} \right| \right\}.$$

You can define multiple load cases for the scaled element centroidal von Mises stress measure. Static linear analysis is supported. Static nonlinear analysis supports only contact nonlinearities. Nonlinear materials and geometrical nonlinearities, such as large deformations, are not supported.

Abaqus/CAE Usage: Optimization module: **Task→*general topology task***, **Design Response→Create**: **Single-term**, **Variable: Stress**

Strain energy

The compliance of a structure is a measure of its overall flexibility or stiffness and is defined as the sum of the strain energy of all the elements, $\sum u^t k u$ for linear models, where u is the displacement vector and k is the global stiffness matrix. Compliance is the reciprocal of stiffness, and minimizing the compliance is equivalent to maximizing the global stiffness. If a load case is driven by forces or pressures, you should choose to minimize the strain energy to maximize the global stiffness. However, if a load case is driven by a thermal field, strain energy decreases when the optimization modifies the structure to make it softer. As a result, you should always choose to maximize the strain energy because attempting to minimize the strain energy can result in a stiff structure. In addition, you should always choose to maximize the strain energy if prescribed displacements are applied to your model.

Topology optimization considers the total strain energy for all of the elements; therefore, if you choose strain energy as an objective function, you must apply the objective to the entire model. You cannot use strain energy as a constraint in your optimization.

Abaqus/CAE Usage: Optimization module: **Task→*condition-based topology or general bead task***, **Design Response→Create**: **Single-term**, **Variable Strain energy**

Volume

The volume is defined as the sum of the volume of the elements in the design area, $\sum V_e$, where V_e is the element volume. During a topology optimization, the elements are scaled with the current relative density defined in your Abaqus model. For most optimization problems, you must apply a volume constraint. For example, if you are trying to minimize the strain energy (maximize the stiffness) and do not apply a volume constraint, the Optimization module simply fills the entire design area with material.

Abaqus/CAE Usage: Optimization module: **Task→*condition-based topology task***, **Design Response→Create**: **Single-term**, **Variable: Volume**

Weight

The weight is defined as the sum of the weight of all the elements in the design area, $\sum W_e$, where W_e is the element weight. The Optimization module scales elements using the current relative density. For most optimization problems, you must apply either a volume or a weight constraint. Using weight instead of volume allows you to constrain the optimized model to a specified physical weight and accounts for regions composed of materials with different densities. The Optimization module uses only supported element types when calculating the weight.

Abaqus/CAE Usage: Optimization module: **Task→*general topology or sizing task***, **Design Response→Create**: **Single-term**, **Variable: Weight**

Design response operators

You must specify the operation that the Optimization module will use to arrive at a single scalar value for the design response, although some restrictions apply. For example, a volume design response can only use the sum of the volume within the design area. A design response that calculates the von Mises stress must use the maximum value of the stress within a region of the model. (None of the operators are relevant when the Optimization module calculates a dynamic frequency design response.) The following design response operators are provided by the Optimization module:

- **Minimum or maximum:** The minimum or maximum value within the selected region. The Optimization module allows only the maximum operator for stress, contact stress, and strain design responses.
- **Sum:** The sum of all the values within the selected area. The Optimization module allows only the sum operator for volume, weight, moment of inertia, and gravity design responses.

The available design responses for each type of optimization

The design responses you can create are dependent on the type of structural optimization you are performing—topology, shape, sizing, or bead. The type of optimization also controls whether you can use a design response as an objective, as a constraint, or as both.

Table 13.2.1–4 Design responses for condition-based topology optimization.

Design response	Objective	Constraint
Strain energy	√	
Volume		√

Table 13.2.1–5 Design responses for general topology optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmaier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	√
Energy stiffness measure	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√
Scaled centroidal von Mises stress	√	√
Strain energy	√	
Volume		√
Weight	√	√

Table 13.2.1–6 Design responses for shape optimization.

Design response	Objective	Constraint
Contact stress	√	
Damage	√	

Design response	Objective	Constraint
Eigenfrequency from Kreisselmaier-Steinhauser formula	√	
Equivalent strain	√	
Equivalent stress	√	
Nodal strain energy density	√	
Volume		√
Weight		√

Table 13.2.1–7 Design responses for sizing optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmaier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	√
Energy stiffness measure	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√
Strain energy	√	
Volume		√
Weight	√	√

Table 13.2.1–8 Design responses for general bead optimization.

Design response	Objective	Constraint
Center of gravity	√	√
Displacement and rotation	√	√
Eigenfrequency from Kreisselmaier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	√
Internal and reaction forces and moments	√	√
Moment of inertia	√	√
Strain energy	√	

Table 13.2.1–9 Design responses for condition-based bead optimization.

Design response	Objective	Constraint
Bead height		√
Eigenfrequency from Kreisselmaier-Steinhauser formula	√	
Eigenfrequency from modal analysis	√	
Strain energy	√	

Operating on design responses

You can define a design response that is a combination of the single values generated by multiple design responses; for example, you can add values or find the maximum of several values. You can also define a design response that is the result of an operation on another design response; for example, the difference between the value of the design response at different nodes.

For example, you can create two design responses that correspond to the displacement in the 1-direction of two selected vertices. Alternatively, you can create a design response that is the difference between the displacement in the 1-direction of two selected vertices. You can then define a constraint that

forces the design response to be close to zero. In effect, the constraint forces the two selected vertices to move together in the 1-direction.

Abaqus/CAE Usage: Optimization module: **Design Response→Create**: **Combined-term**

Additional references

- Bakhtiary, N., P. Allinger, M. Friedrich, F. Mulfinger, J. Sauter, O. Müller, and J. Puchinger, "A New Approach for Size, Shape and Topology Optimization," SAE International Congress and Exposition, Detroit, Michigan, USA, February 26–29, 1996.
- Bendsøe, M. P., E. Lund, N. Ohloff, and O. Sigmund, "Topology Optimization - Broadening the Areas of Application," Control and Cybernetics, vol. 34, pp. 7–35, 2005.
- Bendsøe, M. P., and O. Sigmund, *Topology Optimization: Theory, Methods and Applications,* Springer-Verlag, Berlin Heidelberg New York, 2003.
- Bendsøe, M. P., and O. Sigmund, "Material Interpolations in Topology Optimization," Archive of Applied Mechanics, vol. 69, pp. 635–654, 1999.
- Clausen, P. M., and C. B. W. Pedersen, Non-Parametric Large Scale Structural Optimization, ECCM 2006 III European Conference on Computational Mechanics, Lisbon, Portugal, June 5–9, 2006.
- Cook, R. D., D. S. Malkus, and M. E. Plesha, *Concepts and Applications of Finite Element Analysis,* John Wiley & Sons Inc., 1989.
- Emmrich, D., "Entwicklung einer FEM-basierten Methode zur Gesaltung von Sicken für biegebeanspruchte Leitstützstrukturen im Konstruktionsprozess," Forschungsberichte des Instituts für Produktentwicklung, 13. Karlsruhe, 2004.
- Hansen, L. V., "Topology Optimization of Free Vibrations of Fiber Laser Packages," Structural and Multidisciplinary Optimization, vol. 29(5), pp. 341–348, 2005.
- Olhoff, N., and J. Du, Topology Optimization of Vibrating Bi-Material Plate Structures with Respect to Sound Radiation, IUTUAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives, M. P. Bendsøe, N. Olhoff, and O. Sigmund, eds., pp. 147–156, Springer, 2006.
- Pedersen, C. B. W., and P. Allinger, Industrial Implementation and Applications of Topology Optimization and Future Needs, IUTUAM Symposium on Topological Design Optimization of Structures, Machines and Materials: Status and Perspectives, M. P. Bendsøe, N. Olhoff, and O. Sigmund, eds., pp. 147–156, Springer, 2006.
- Sigmund, O., and J. S. Jensen, "Systematic Design of Phononic Band Gap Materials and Structures by Topology Optimization," Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences, vol. 361, pp. 1001–1019, 2003.
- Stolpe, M., and K. Svanberg, "An Alternative Interpolation Scheme for Minimum Compliance Optimization," Structural and Multidisciplinary Optimization, vol. 22, pp. 116–124, 2001.
- Svanberg, K., "The Method of Moving Asymptotes—A New Method for Structural Optimization," International Journal for Numerical Methods in Engineering, vol. 24, pp. 359–373, 1987.

13.2.2 OBJECTIVES AND CONSTRAINTS

Product: Abaqus/CAE

References

- "Structural optimization: overview," Section 13.1.1
- "Creating objective functions," Section 18.8 of the Abaqus/CAE User's Guide
- "Creating constraints," Section 18.9 of the Abaqus/CAE User's Guide
- "Configuring geometric restrictions," Section 18.10 of the Abaqus/CAE User's Guide
- "Creating local stop conditions," Section 18.11 of the Abaqus/CAE User's Guide

Overview

For an optimization problem:

- an objective function defines the objective of the optimization;
- a constraint imposes limitations on the optimization and defines a feasible design;
- geometric restrictions impose limitations on the topology or shape of the structure that can be generated by the optimization; and
- stop conditions define when an optimization task is considered complete.

Objective functions

Objective functions define the objective of the optimization. An objective function is a single scalar value that is formulated from a set of design responses. For example, if the design responses are defined from the strain energy of the nodes in a region, the objective function could minimize the sum of the design responses; i.e., minimize the sum of the strain energy, in effect maximizing the stiffness of the region.

An optimization problem can be stated as:

$$\min\left(\Phi\left(U\left(x\right),x\right)\right),$$

where Φ is the objective function that depends on the state variables, U, and the design variables, x.

The formula for the objective function that tries to minimize N design responses can be stated as:

$$\Phi_{\min}=\min\left(\sum_{i=1}^{N}W_i\left(\varphi_i-\varphi_i^{ref}\right)\right),$$

where each design response, φ_i, is given a weight, W_i, and a reference value, φ_i^{ref}. The formula for the objective function that tries to maximize N design responses can be stated as:

$$\Phi_{\max} = \max\left(\sum_{i=1}^{N} W_i\left(\varphi_i - \varphi_i^{ref}\right)\right).$$

The default weighting factor is 1.0. For a topology optimization the default reference value is 0.0; for shape and sizing optimization the default reference value is calculated by the Optimization module. For the most common optimization problems you do not need to change the default values of the weighting factor and the reference value. However, in some cases you may have to change the weighting factor to balance the effect of an objective function that is dominating the optimization. You should be aware that changing the weighting factor can have a significant impact on the final design. In addition, a design response that is dominant at the start of the optimization may have less effect as the Optimization module modifies your model.

An objective function that tries to minimize the maximum design response is an important optimization formulation. During each design cycle the Optimization module first determines which of the set of weighted design responses has the maximum value and then tries to minimize that design response. In many problems, minimizing the maximum design response provides a satisfactory solution because it reduces the maximum of a number of design responses. For example, if your design responses are defined from the stress in multiple regions of your model, minimizing the maximum design response attempts to minimize the stress in the region that is exhibiting the maximum stress. The formula can be stated as:

$$\Phi_{minmax} = \min\left(\max_i\left\{W_i\left(\varphi_i - \varphi_i^{ref}\right)\right\}\right).$$

The design responses provided with the Optimization module are listed in "Design responses," Section 13.2.1.

Defining the target of an objective function

The target of an objective function can be minimized or maximized. Alternatively, the target of an objective function can be set to minimize the maximum, such that the design response targets the maximum value, and the objective attempts to minimize that maximum value. In all cases, the weighting and reference values of the design responses are accounted for.

Abaqus/CAE Usage: Optimization module: **Objective Function→Create**: **Target**

Constraints

As outlined in the previous section, an optimization problem can be stated as:

$$\min\left(\Phi\left(U\left(x\right), x\right)\right),$$

where Φ is the objective function that depends on the state variables, U, and the design variables, x. Constraints, Ψ, can be applied to the optimization problem, and constraints, K_i, can be applied to the design variables:

$$\Psi_i\left(U\left(x\right),x\right) \leq 0,$$
$$K_i\left(x\right) \leq 0,$$

where $\Psi_i\left(U\left(x\right),x\right) \leq \Psi_i^*$ and Ψ_i is the design response that is constrained by the value Ψ_i^*. In addition, $K_i\left(x\right) \leq K_i^*$, where K_i is an expression for the layout of the design variables, such as manufacturability, and K_i^* is the constraint on the design variables.

The Optimization module can arrive at a solution that optimizes the objective function; however, if the constraints are not satisfied, the result of the optimization may not be a feasible design. A constraint is based on a design response and, similar to a design response, is formulated from a single scalar value. Most optimizations have constraints that prevent the optimization from arriving at a trivial solution. For example, if you are trying to maximize the stiffness of a structure, the Optimization module will simply fill the entire design area if you do not apply any constraints. However, if you apply a weight constraint that limits the weight to 50% of its initial value, the Optimization module is forced to seek an optimum solution that both optimizes the stiffness objective and satisfies the weight constraint. You can apply only volume constraints to both topology optimization and to shape optimization; you cannot use volume as an objective function. You cannot apply multiple constraints of the same type, such as volume, to the whole model or to a single region.

Abaqus/CAE Usage: Optimization module: **Constraint→Create**

Applying constraints to regions

You can apply different constraints to different regions of your model. In addition, those regions can have different material properties or a material property can vary within a region. When the Optimization module calculates the design response, it considers varying material properties within the region. You cannot apply multiple volume constraints to the whole model or to a single region.

Geometric restrictions

Geometric restrictions are constraints that are applied directly to the design variables. Geometric restrictions allow you to model design limitations and manufacturing limitations.

Defining a frozen area

You can specify that a region within the optimization region is excluded from the optimization by freezing the region. For example, you could exclude a circular shaft that forms a bearing surface or a boss that is used to attach the structure to a rigid surface. You must freeze regions that are used to apply prescribed conditions. To simplify this operation, you can request that the Optimization module automatically freeze regions that are used to apply prescribed conditions and loads when you create an optimization process.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Frozen area**

Specifying minimum and maximum member size

In most cases you should try to avoid the generation of thin trusses in the structure by defining a minimum member size. However, the Optimization module cannot ensure that the optimized structure will not contain regions with a diameter that is smaller than the minimum member size. The minimum member

size must be larger than the average element edge length. The maximum member size must be larger than twice the element length; otherwise, the optimization algorithm may experience issues with element connectivity. A coarse mesh and a fine mesh lead to an optimization with the topological equivalent result if you specify the same minimum member size for both cases. The Optimization module will not generate a thin region where prescribed conditions have been applied to the structure. Removing material from these regions may result in the structure collapsing.

If your structure will be cast, you may want to avoid the generation of overly thick parts by specifying a region with a maximum member size. The optimization process will avoid creating a thick region by generating several thinner regions. You do not need to specify both a maximum and a minimum member size. The Optimization module assumes the value that you enter for the maximum member size also applies to the minimum member size and will generate trusses of the specified size. The combination of a maximum member size constraint with a restraint that imposes a pull direction, such as a moldable or stampable manufacturing constraint, is allowed only for a general topology optimization. (The "pull direction" is the direction in which the two halves of a mold separate or the direction in which a stamping tool moves.)

Computational time increases significantly when you specify regions with a minimum or maximum member size. Therefore, you should apply the member size restrictions only to regions where thin or thick members must be avoided. You should run an optimization without member size restrictions to identify such regions.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Member size**

Applying manufacturing restrictions

The topology optimization process always creates a structural layout that satisfies the objective function and the constraints; however, the design may be impossible to create using standard manufacturing techniques, such as casting and forging. You can apply geometric restrictions that force the topology optimization process to consider only designs that can be manufactured. For example, when you are using topology optimization you can force the Optimization module to create a castable shape that can be extracted from a mold or a stampable shape that can be created with a tool and die.

Maintaining a moldable structure

In cases where bending and torsion loads are applied, topology optimization is likely to generate a model with hollow areas or with undercuts that cannot be manufactured. You can prevent the topology optimization from generating cavities and undercuts by specifying the following:

- A forgeable structure that can be removed from the forging die, as shown in Figure 13.2.2–1.
- A moldable structure that can be removed from two halves of a mold, as shown in Figure 13.2.2–2. In contrast, Figure 13.2.2–3 illustrates parts with a cavity and an undercut that are not moldable.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Demold control**; **Demold technique**, **Demolding with a central plane**

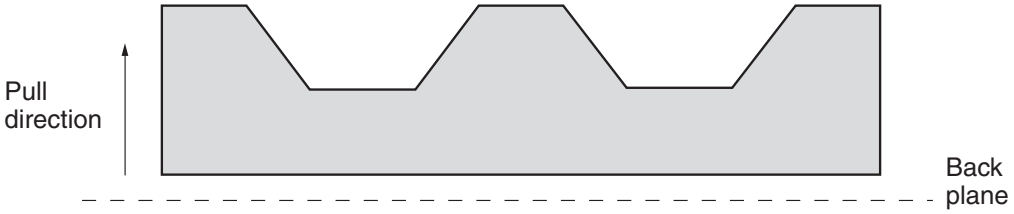


Figure 13.2.2–1 A forgable part.

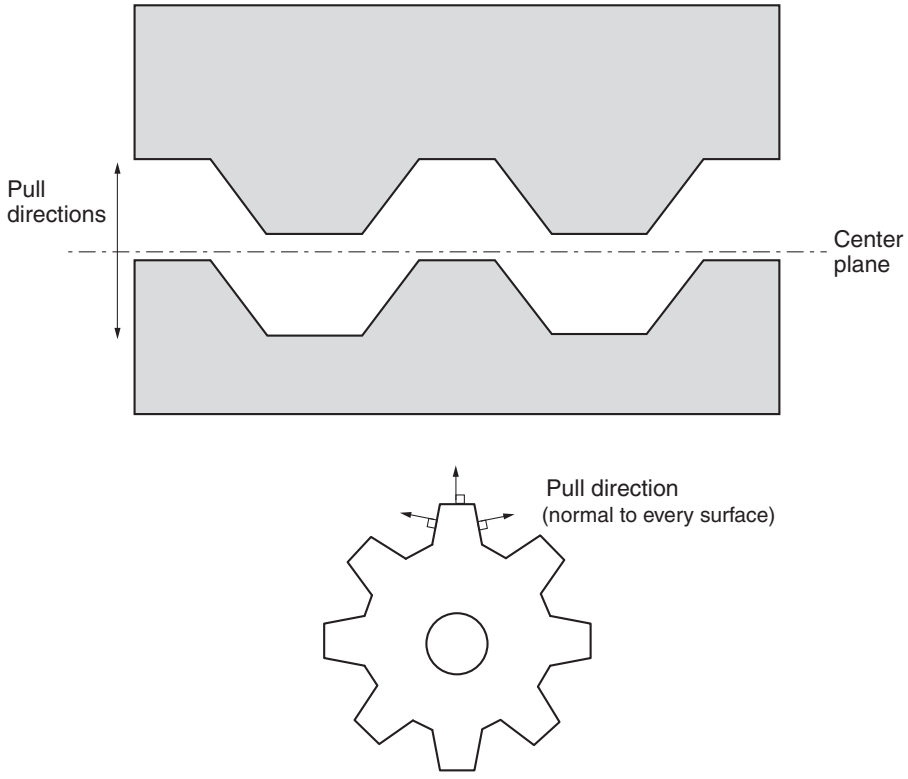


Figure 13.2.2–2 A moldable part.

Optimization module: **Geometric Restriction→Create**: **Demold control**; **Demold technique**, **Demolding at the region surface**

Optimization module: **Geometric Restriction→Create**: **Demold control**; **Demold technique**, **Forging**

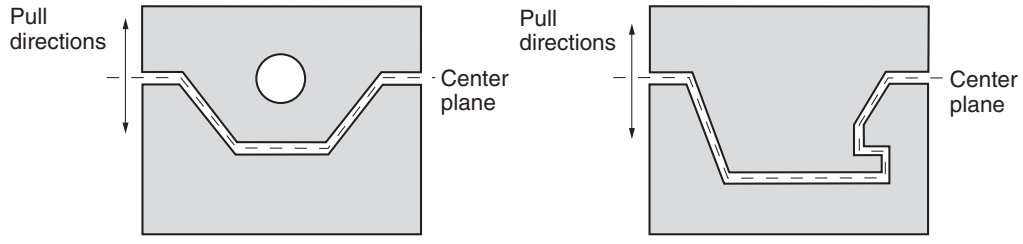


Figure 13.2.2–3 Cavities and undercuts prevent a part from being moldable.

Maintaining a stampable structure

You can specify that the structure is to be manufactured by a stamping process. If the optimization process removes one element from the structure, it also removes all elements positioned either behind or in front of the element (with respect to the pull direction), as shown in Figure 13.2.2–4.

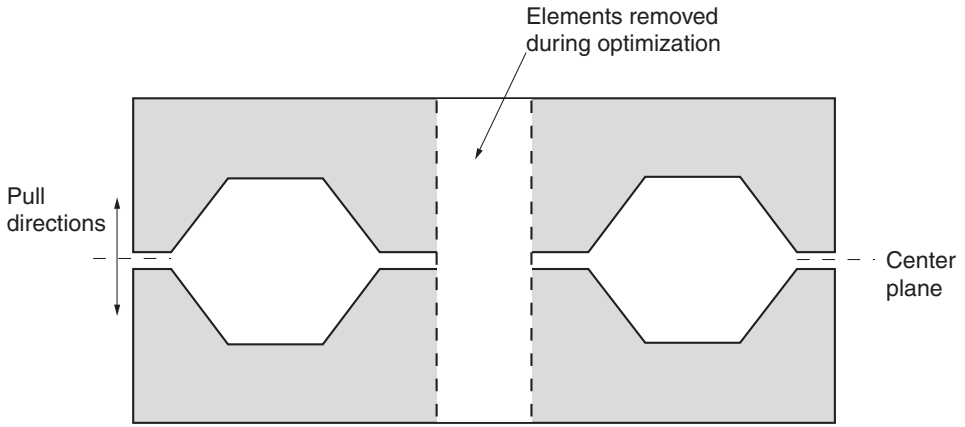


Figure 13.2.2–4 A stampable structure.

The rate at which the Optimization module modifies the element properties should not be set too high if the stamping restriction is activated in a condition-based topology optimization; otherwise, supports or trusses generated by the optimization may become unattached from the rest of the structure.

Abaqus/CAE Usage: Use the following option to create a stamping geometric restriction in a topology optimization:

Optimization module: **Geometric Restriction→Create**: **Demold control**; **Demold technique**, **Stamping**

Use the following option to create a stamping geometric restriction in a shape optimization:

Optimization module: **Geometric Restriction→Create**: **Stamp control**

Use the following option to specify the rate at which the Optimization module modifies the element properties:

Optimization module: **Task→Create**: **Advanced** , **Size of increment for volume modification**

Specifying bounds on shell thickness

You can specify upper and lower bounds on the thickness of shell elements when you are configuring a sizing optimization. The value can be an absolute thickness or a fraction of the initial thickness.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Thickness control**

Specifying a symmetric structure

Introducing symmetry constraints into your model can significantly increase the speed at which the Optimization module calculates the optimized structure. You can use the Optimization module to apply the following symmetry constraints:

- Symmetry about an axis or plane (reflection symmetry)
- Symmetry about a point
- Rotational symmetry
- Cyclic symmetry (replication of an area with a given distance)

You can apply a symmetry restriction to unstructured meshes or to tetrahedron meshes in a topology optimization. The elements should be approximately the same size because the resulting symmetry is based on the resolution of the coarsest part of the mesh. In addition, the Optimization module may fail to create the symmetric conditions if the difference in the element size is too large.

To define symmetry for a shape optimization, the Optimization module assembles nodes that are approximately symmetric into a symmetry group (normally there are two symmetric nodes in each symmetry group). The Optimization module then determines the master node of the symmetry group and calculates the design displacements of the client nodes in such a way that they move symmetrically to the plane of the master node.

If you are performing a topology optimization, your meshed Abaqus model does not have to be symmetric before the optimization starts. Conversely, if you are performing a shape optimization, your meshed Abaqus model should be symmetric before the optimization starts to allow the Optimization module to identify symmetric nodes and maintain their symmetry when the surface nodes are moved.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Planar symmetry**, **Point symmetry**, **Rotational symmetry**, or **Cyclic symmetry**

Specifying the minimum width

A sizing optimization determines the optimal element thickness when modeling a sheet metal structure with shell elements. Specifying the minimum width of a region containing elements of the same thickness prevents narrow slivers of elements with equal thickness appearing in the solution after a sizing optimization. Specifying a minimum width also prevents oscillations in the shell thickness or a "checkerboard" pattern of element thicknesses.

A coarse mesh and a fine mesh lead to the same optimized solution if the same minimum width is specified in both cases; thus, the optimized solution is independent of the mesh resolution. However, in all cases the minimum width must be larger than the average length of the element edges.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Thickness control**

Specifying clustering

You can specify that selected regions contain clusters of shell elements of equal thickness after a sizing optimization. You can use clustering to generate strengthening ribs or rings in the sheet metal structure you are optimizing or to define borders between regions of equal thickness. Clustered regions can be reproduced in manufacturing using sheets of constant thickness; for example, a vehicle "body in white" formed by welding and stamping individual sheet metal structures. To allow for maximum design flexibility, you should first optimize your structure without specifying clustering and use the initial design to decide which regions to cluster in your final optimization.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Cluster areas**

Applying additional restrictions during a shape optimization

Shape optimization determines the displacement of each surface node in an effort to homogenize the stress on the surface and satisfy the objective function and any constraints. The Optimization module does not couple the displacement of neighboring nodes; each of the design nodes can move independently of the other design nodes. For example, during the optimization a planar surface can develop into a nonplanar free-form surface. By coupling the design nodes you can force the optimization to maintain the regularity of a plane.

Coupling conditions restrict the range of solutions for the system and reduce the optimization potential. In addition, defining the appropriate coupling conditions can be very time consuming. To simplify your optimization, you should start with an optimization with as few restrictions as possible and only a few coupling conditions and introduce additional coupling conditions only if they are required.

You can apply additional restrictions while the Optimization module is moving surface nodes during a shape optimization:

- The optimized shape can be manufactured by a tool on a lathe cutting into the model along a specified direction.
- The optimized shape can be manufactured by a tool drilling into the model along a specified direction. The hole created by the tool is symmetric about the axis of the tool. In addition, the tool can be withdrawn from the hole.

- Selected faces in the optimized shape can slide along each other and/or cannot penetrate each other.
- Nodes are restricted to move:
  - along a specified vector,
  - a specified distance either inward or outward (shrinkage or growth),
  - along a specified direction,
  - only along selected degrees of freedom, and
  - only in the direction of applied loads.

Abaqus/CAE Usage: Optimization module: **Geometric Restriction→Create**: **Turn control**, **Drill control**, **Penetration check**, **Slide region control**, or **Vector**

Combining geometric constraints

Each geometric constraint that you apply reduces the possibility of Abaqus arriving at an optimized solution. In addition, if you apply too many geometric restrictions, the solution that Abaqus generates may not be the most optimal solution available. Therefore, you should start by allowing Abaqus to perform an optimization with no geometric restrictions applied or with only a limited number. After you have studied the results of the unrestricted, or less-restricted, optimization, you should apply only the restrictions that are required to solve the problem.

You can combine geometric constraints; however, only certain combinations are permissible. Abaqus processes geometrical constraints in the following order:

- Minimum member size
- Symmetry constraints
- Manufacturing constraints
- Maximum member size

Applying one constraint may weaken the effect of another constraint. For example, you cannot define symmetry about a plane in conjunction with a demold pull direction that is not parallel with the axis or plane of symmetry.

The following manufacturing restriction combinations are permissible:

- You can combine symmetry about a plane with a pull direction provided the pull direction is perpendicular or parallel to the plane of symmetry.
- You can combine rotational symmetry with a pull direction provided the pull direction is parallel to the axis of rotation.
- You can combine two symmetries about a plane provided the planes are perpendicular.
- You can specify a minimum member size that is greater than the maximum member size. Abaqus first processes the minimum member size requirement and creates relatively thick supports. The thick supports are subsequently divided into smaller parallel members when Abaqus processes the maximum size requirement.

Stop conditions

Stop conditions are examined after each design cycle and determine whether an optimization should end because the maximum number of design cycles has been reached or because the optimization has converged on an optimal solution. The Optimization module provides both global and local stop conditions; however, local stop conditions are supported only by shape optimization and are rarely required.

Global stop conditions

The global convergence stop condition defines the maximum number of design cycles that should be performed. To limit the number of design cycles, you must define a global stop condition for each optimization task. The default value for the maximum number of design cycles depends on the type of optimization, as shown in Table 13.2.2–1.

Table 13.2.2–1 Default maximum number of design cycles.

Optimization Type	Default maximum number of design cycles
Condition-based topology optimization	15
General topology optimization	50
Shape optimization	10
Sizing optimization	50
Condition-based bead optimization	3
General bead optimization	20

Abaqus/CAE Usage: Job module: **Optimization→Create**: **Maximum cycles**

Local stop conditions

Local stop conditions indicate if a shape optimization has converged on an optimal solution. Local stop conditions apply to the displacements or stresses in a region of your model and define when the goals of an optimization have been reached. A local stop condition compares a single scalar value of displacement or equivalent stress to a reference value. The single scalar value can be either the maximum or minimum value over a region or the sum of all the values. The reference value can be taken from the value of the single scalar value after the previous iteration or after the first iteration. In addition, you can modify the reference value by a fixed amount or by a percentage. For example, you can specify a local stop condition that ends the optimization if the sum of the displacements within a region is smaller than 1% of the sum of the displacements after the first optimization cycle. You can define one or two local stop conditions, and you can specify if either or both (default) of the local stop conditions must be met for the Optimization module to end the optimization.

Examples of local stop conditions include the following:

- If you have specified that the displacement or stress should be minimized (or maximized), a local stop condition can end the optimization if the value of the displacement or stress increases (or decreases) after an optimization cycle.
- When the optimization approaches the optimum solution, you can expect only small changes in the value of the displacement or stress. A local stop condition can end the optimization if the relative change in the displacement or stress falls below a tolerance limit after an optimization cycle.
- When the optimization approaches the optimum solution, you can expect only small changes in the sum of the displacements and, therefore, only minor modifications to the model. A local stop condition can end the optimization if the change in the sum of the displacements falls below a tolerance limit after an optimization cycle. You can use the sum of the displacements as a stop condition for optimizations with and without constraints. In addition, this stop condition is suitable for a variety of objective functions, such as stress or frequency.

Abaqus/CAE Usage: Optimization module: **Stop Condition→Create**

13.2.3 CREATING Abaqus OPTIMIZATION MODELS

Product: Abaqus/CAE

References

- "Structural optimization: overview," Section 13.1.1
- "Understanding optimization," Section 18.3 of the Abaqus/CAE User's Guide

Overview

For each design cycle the optimization process:

- generates new material and element properties during topology optimization;
- modifies nodal coordinates during shape optimization;
- sends the modified model to an Abaqus analysis; and
- reads the results of the analysis.

Preparing the Abaqus model

You should take care to ensure that your Abaqus model is supported by structural optimization. Any restrictions imposed by the use of structural optimization, such as the supported element types, apply only to the design area; regions outside the design area do not play a role in the optimization.

- You must ensure that your Abaqus model can be analyzed and produces the expected mechanical results before you attempt to optimize your model.
- You should account for nonlinearities only if your model is truly nonlinear; the optimization will be significantly less expensive computationally if your Abaqus model is linear. You may want to ensure that an optimization of a linear version of your model produces reasonable results before you introduce geometric or material nonlinearities.
- An optimization takes multiple design cycles to complete, and the time required to reach an optimized solution can be significant. As a result, you must configure your Abaqus model to minimize computational time; for example, by removing small details that are not important to the optimization.
- The Optimization module does not support the use of parts and assemblies in the Abaqus input file. When you run an optimization task, the Optimization module generates a flattened input file that does not use parts and assemblies.
- The Optimization module reads data from the output database (`.odb`) files that are created during each design cycle. The Optimization module requests data only from the end of each step. To minimize the size of the output database files, you should also request data only from the end of each step.

Support for analysis types

The following Abaqus analysis types are supported by topology, shape, sizing, and bead optimization:

- Static stress/displacement, general analysis
- Static stress/displacement, linear perturbation analysis
- Extract natural frequencies and modal vectors

Support for geometric nonlinearities

You can specify that geometric nonlinearity should be accounted for only during static stress/displacement analyses.

Elements that have limited stiffness, such as elements with hyperelastic material properties, can deform excessively during topology optimization in a nonlinear analysis. This deformation can lead to an adverse effect on the convergence and result in the termination of the analysis. You should be aware of this potential issue when applying topology optimization using hyperelastic materials.

Sizing optimization supports geometric nonlinearity only if the maximum elemental effective total strain for the design elements is less than 2%. Sizing optimization supports geometric nonlinearity outside the design area where any magnitude of total strain for an element is allowed.

Bead optimization does not support geometric nonlinearity.

Support for multiple load cases

If your model is undergoing a sequence of loads, you can significantly reduce the computational cost by defining a multiple load case analysis within a single step.

Support for multiple models

A design response can include steps or load cases from multiple Abaqus models. You can incorporate multiple models into your optimization when linear perturbations about a base state are no longer sufficient as load cases. For example, you can simulate nonlinear load cases (which are not supported by Abaqus/CAE) by creating multiple copies of your nonlinear model and by creating a step in each model during which different loads and boundary conditions are applied. For a meaningful optimization, it is expected that each model will have the sameAbaqus/CAE geometry and the same mesh.

Support for temperature loading

General topology and sizing optimization support constant temperature loading.

Support for acceleration loading

General topology optimization supports prescribed acceleration loading from

- gravity,
- rotational body forces, and
- centrifugal forces.

Coriolis forces are not supported.

Support for contact during the optimization

You can avoid contact in optimized regions of your model by defining geometric restrictions, such as casting or minimum member size restrictions. In some cases, you cannot specify the exact boundary conditions early in the design phase. In addition, nonlinear boundary conditions, such as contact definitions, can change if the Optimization module changes the topology of the model.

The optimization process is more efficient if you create an Abaqus model with the appropriate contact definitions and allow Abaqus to calculate the contact. The contact conditions are included in the optimization through the forces at the nodes and the stresses in the elements, and both topology and shape optimization permit contact conditions in the Abaqus model.

You can define a contact surface directly on the edge of the design space in topology optimization. However, if the design edge belongs to a contact surface in shape optimization, you must invert the shape optimization algorithm by entering a negative growth scale factor. You may encounter convergence difficulties in your Abaqus model if you have a complex contact problem or if the optimization results in large changes in the model.

Restrictions on an Abaqus model used for topology optimization

Topology optimization determines the optimal material distribution in the design space, given the prescribed conditions applied to the model along with the objective function and constraints. Your optimization must apply appropriate constraints and restrictions; otherwise, the Optimization module can extensively alter the topology of the component. The resolution of the structure that has been optimized with topology optimization is very dependent on the discretization. A fine mesh produces a structure with a higher resolution than a coarse mesh; however, it will also substantially increase the processing time required. You must determine the appropriate compromise between structural resolution and processing time.

During topology optimization the Optimization module modifies the material definition of the elements in the design area. As a result, you must provide the initial density of the materials in the design area, even if it is not required by the Abaqus analysis.

Restrictions on an Abaqus model used for shape optimization

Abaqus performs a shape optimization by modifying the boundaries or surfaces of a component. The optimization uses the stress condition to calculate new coordinates for nodes on the surface of the component and then adjusts the underlying mesh accordingly. The mesh quality must be sufficient to ensure that the analysis results are mostly unchanged by the movement of the surface nodes. High stress gradients must not be present within an element.

When the Optimization module is performing a shape optimization on a shell structure, it optimizes the form of the shell structure and not its thickness. The nodal position along shell edges can be modified; however, Abaqus does not modify the shell definition.

Restrictions on an Abaqus model used for sizing optimization

Abaqus performs a sizing optimization by modifying the thickness of shell elements in the design region. The element thickness must be uniform, and only single-layered shells are supported. Prescribed

displacements are allowed in a static stress/displacement analysis; however, they are not supported in a frequency analysis.

Restrictions on an Abaqus model used for bead optimization

Abaqus performs a bead optimization by moving nodes of shell elements in the direction of the shell normal in the design region. The element thickness must be uniform, and only single-layered shells are supported. Prescribed displacements are allowed in a static stress/displacement analysis; however, they are not supported in a frequency analysis.

Supported materials in the design area

The material models supported by structural optimization in the elements in the design area depend on the type of optimization—condition-based topology optimization, general topology optimization, or shape optimization.

Materials supported by condition-based topology optimization

Condition-based topology optimization in Abaqus supports linear elastic, plastic, and hyperelastic material models.

Support for linear elastic material models

The following linear elastic material models are supported by condition-based topology optimization:

- Linear elastic materials with isotropic behavior.
- Linear elastic materials with fully anisotropic behavior.
- Linear elastic materials with orthotropic behavior. All of the behavior models are supported, except for orthotropic shear behavior for warping elements and coupled and uncoupled traction behavior for cohesive elements.

Support for plastic material models

Metal plasticity material properties—the plastic part of the material model for elastic-plastic materials that use the Mises or Hill yield surface—are supported by condition-based topology optimization. Isotropic hardening is supported; however, cyclic loading is not supported—each material point can be unloaded only once and should not become elastoplastic again.

Support for hyperelastic material models

All of the hyperelastic material models are supported by condition-based topology optimization, except for the Marlow material model and the hyperelastic material models with test data.

Support for temperature and field variable dependency

Condition-based topology optimization supports materials that have temperature and field variable dependency.

Materials supported by general topology optimization

General topology optimization in Abaqus supports linear elastic, plastic, and hyperelastic material models.

Support for linear elastic material models

The following linear elastic material models are supported by general topology optimization:

- Linear elastic materials with isotropic behavior.
- Linear elastic materials with fully anisotropic behavior.
- Linear elastic materials with orthotropic behavior. All of the behavior models are supported, except for orthotropic shear behavior for warping elements and coupled and uncoupled traction behavior for cohesive elements.

Support for plastic material models

Metal plasticity material properties—the plastic part of the material model for elastic-plastic materials that use the Mises or Hill yield surface—are supported by general topology optimization. Isotropic hardening is supported; however, cyclic loading is not supported—each material point can be unloaded only once and should not become elastoplastic again.

Support for hyperelastic material models

All of the hyperelastic material models are supported by general topology optimization, except for the Marlow material model and the hyperelastic material models with test data.

Support for temperature and field variable dependency

Materials that have temperature and field variable dependency are supported by general topology optimization.

Material support in shape optimization

All of the Abaqus material models are supported by shape optimization.

Material support in sizing optimization

Nonlinear materials in the design area are not supported by sizing optimization. All of the Abaqus material models, including nonlinear materials, are supported outside the design area.

Material support in bead optimization

Nonlinear materials in the design area are not supported by bead optimization. All of the Abaqus material models, including nonlinear materials, are supported outside the design area.

Support for coordinate systems

In most cases, you will use the same coordinate system to define your model and the optimization task. However, the Optimization module allows you refer to a different coordinate system when you are defining a design response.

Supported element types

The Abaqus elements that are supported as design elements by topology and shape optimization are listed in Table 13.2.3–1 through Table 13.2.3–4. The tables also list the Abaqus elements that support the reaction and internal force design responses. The shell elements that are supported as design elements by sizing and bead optimization are listed in Table 13.2.3–5 and Table 13.2.3–6, respectively. Unsupported elements are ignored during optimization and remain unchanged. Structural optimization does not place any restrictions on the type of elements that you use outside the design area.

Supported two-dimensional solid elements

Topology optimization (both condition-based and general) and shape optimization support the two-dimensional solid elements listed in Table 13.2.3–1.

Table 13.2.3–1 Supported two-dimensional solid elements.

CPE3[1], CPE3H, CPE4[1], CPE4H, CPE4I, CPE4IH, CPE4R[1], CPE4RH,
CPE6H, CPE6M, CPE6MH
CPE8[1], CPE8H, CPE8R[1], CPE8RH
CPS3[1], CPS4[1], CPS4I, CPS4R[1], CPS6[1], CPS6M, CPS6MT, CPS8[1]. CPS8R[1]
CPEG3, CPEG3H, CPEG4, CPEG4H, CPEG4I, CPEG4IH, CPEG4R, CPEG4RH, CPEG6, CPEG6H, CPEG6M, CPEG6MH, CPEG8, CPEG8H, CPEG8R, CPEG8RH
CPE3T, CPE4T, CPE4HT, CPE4RT, CPE4RHT, CPE6MT, CPE6MHT, CPE8T, CPE8HT, CPE8RT, CPE8RHT
CPS3T, CPS4T, CPS4RT, CPS8T, CPS8RT
CPEG3T, CPEG3HT, CPEG4T, CPEG4RT, CPEG4RHT, CPEG6MT, CPEG6MHT, CPEG8T, CPEG8HT, CPEG8RHT
[1] Can include reaction and internal force design responses.

Supported three-dimensional solid elements

Topology optimization (both condition-based and general) and shape optimization support the three-dimensional solid elements listed in Table 13.2.3–2.

Table 13.2.3–2 Supported three-dimensional solid elements.

C3D4[1], C3D4H, C3D8[1]
C3D6[1], C3D6H
C3D8H, C3D8I, C3D8IH, C3D8R[1], C3D8RH
C3D10[1], C3D10H, C3D10M, C3D10MH
C3D15[1], C3D15H
C3D20[1], C3D20H, C3D20R[1], C3D20RH
C3D4T, C3D6T, C3D8T, C3D8HT, C3DHRT, C3D8RHT, C3D10MT, C3D10MHT, C3D20T, C3D20HT, C3D20RT, C3D20RHT
[1] Can include reaction and internal force design responses.

Supported axisymmetric solid elements

Topology optimization (both condition-based and general) and shape optimization support the axisymmetric solid elements listed in Table 13.2.3–3.

Table 13.2.3–3 Supported axisymmetric solid elements.

CAX3[1], CAX3H, CAX4[1], CAX4H, CAX4I, CAX4IH, CAX4R[1], CAX4RH
CAX8[1], CAX8H, CAX8R[1], CAX8RH
CGAX3, CGAX3H, CGAX4, CGAX4H, CGAX4R, CGAX4RH, CGAX8, CGAX8H, CGAX8R, CGAX8RH
CAX3T, CAX4T, CAX4HT, CAX4RT, CAX4RHT, CAX8T, CAX8HT, CAX8RT, CAX8RHT
CGAX3T, CGAX3HT, CGAX4T, CGAX4HT, CGAX4RT, CGAX4RHT, CGAX8T, CGAX8HT, CGAX8RT, CGAX8RHT
[1] Can include reaction and internal force design responses.

Additional supported elements

Table 13.2.3–4 lists the general membrane, three-dimensional conventional shell, and beam elements that are supported by optimization.

Table 13.2.3–4 Additional supported elements

General membrane elements (topology and shape optimization)	M3D3[1], M3D4[1], M3D4R[1], M3D6[1], M3D8[1], M3D8R[1]
Three-dimensional conventional shell elements (topology optimization only)	STRI3, S3, S3R, STRI65, S4, S4R, S4R5, S8R, S8R5, S8RT
Three-dimensional conventional shell elements (shape optimization only)	STRI3[1], S3[1], S3R[1], S4[1], S4R[1], S8R[1]
Beam elements (shape optimization only)	B21[2], B21H[2], B31[2], B31H[2]
[1] Can include reaction and internal force design responses.	
[2] You can include beam elements in shape optimization only to define a neighboring component that is used to restrict the movement of nodes in the optimized region.	

Supported three-dimensional conventional shell elements

Sizing optimization supports only the three-dimensional conventional shell elements listed in Table 13.2.3–5.

Table 13.2.3–5 Supported three-dimensional conventional shell elements for sizing optimization.

S3, S3R, S4, S4R, S8R
STRI65[1]
[1] You must request that rotational degrees of freedom be written to the output database.

Condition-based bead optimization supports all Abaqus plate and shell elements. However, general bead optimization supports only the three-dimensional conventional shell elements listed in Table 13.2.3–6.

Table 13.2.3–6 Supported three-dimensional conventional shell elements for general bead optimization.

S3, S3R
STRI3
S4, S4R
S8R

14. Eulerian Analysis

Eulerian analysis 14.1

14.1 Eulerian analysis

- "Eulerian analysis," Section 14.1.1
- "Defining Eulerian boundaries," Section 14.1.2
- "Eulerian mesh motion," Section 14.1.3
- "Defining adaptive mesh refinement in the Eulerian domain," Section 14.1.4

14.1.1 EULERIAN ANALYSIS

Products: Abaqus/Explicit Abaqus/CAE

References

- "Eulerian surface definition," Section 2.3.5
- "Eulerian elements," Section 32.14.1
- *EULERIAN SECTION
- *INITIAL CONDITIONS
- *SURFACE
- "Creating Eulerian sections," Section 12.13.3 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- "Defining a material assignment field," Section 16.11.10 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- Chapter 28, "Eulerian analyses," of the Abaqus/CAE User's Guide

Overview

In a traditional Lagrangian analysis nodes are fixed within the material, and elements deform as the material deforms. Lagrangian elements are always 100% full of a single material, so the material boundary coincides with an element boundary.

By contrast, in an Eulerian analysis nodes are fixed in space, and material flows through elements that do not deform. Eulerian elements may not always be 100% full of material—many may be partially or completely void. The Eulerian material boundary must, therefore, be computed during each time increment and generally does not correspond to an element boundary. The Eulerian mesh is typically a simple rectangular grid of elements constructed to extend well beyond the Eulerian material boundaries, giving the material space in which to move and deform. If any Eulerian material moves outside the Eulerian mesh, it is lost from the simulation.

Eulerian material can interact with Lagrangian elements through Eulerian-Lagrangian contact; simulations that include this type of contact are often referred to as coupled Eulerian-Lagrangian (CEL) analyses. This powerful, easy-to-use feature of Abaqus/Explicit general contact enables fully coupled multi-physics simulation such as fluid-structure interaction.

Applications

Eulerian analyses are effective for applications involving extreme deformation, up to and including fluid flow. In these applications, traditional Lagrangian elements become highly distorted and lose accuracy. Liquid sloshing, gas flow, and penetration problems can all be handled effectively using Eulerian analysis. Eulerian-Lagrangian contact allows the Eulerian materials to be combined with traditional nonlinear Lagrangian analyses.

An example of using Eulerian analysis for a severe deformation analysis is discussed in "Rivet forming," Section 2.3.1 of the Abaqus Example Problems Guide; using coupled Eulerian-Lagrangian contact for a fluid-structure interaction application is illustrated in "Impact of a water-filled bottle," Section 2.3.2 of the Abaqus Example Problems Guide.

Eulerian volume fraction

The Eulerian implementation in Abaqus/Explicit is based on the volume-of-fluid method. In this method, material is tracked as it flows through the mesh by computing its Eulerian volume fraction (EVF) within each element. By definition, if a material completely fills an element, its volume fraction is one; if no material is present in an element, its volume fraction is zero.

Eulerian elements may simultaneously contain more than one material. If the sum of all material volume fractions in an element is less than one, the remainder of the element is automatically filled with "void" material. Void material has neither mass nor strength.

Material interfaces

Volume fraction data are computed for each Eulerian material in an element. Within each time increment, the boundaries of each Eulerian material are reconstructed using these data. The interface reconstruction algorithm approximates the material boundaries within an element as simple planar facets (the Eulerian method is implemented only for three-dimensional elements). This assumption produces a simple, approximate material surface that may be discontinuous between neighboring elements. Therefore, accurate determination of a material's location within an element is possible only for simple geometries, and fine grid resolution is required in most Eulerian analyses.

The discontinuities in an Eulerian material surface can lead to physically unrealistic configurations when visualizing the results of an Eulerian analysis. Abaqus/CAE can apply a nodal averaging algorithm to estimate a more realistic, continuous surface during visualization. For more information on visualizing the material interfaces in an Eulerian model, see "Viewing output from Eulerian analyses," Section 28.7 of the Abaqus/CAE User's Guide.

Eulerian section definition

An Eulerian section definition lists all of the materials that may appear within an Eulerian element. Void material is automatically included in this list.

The material list supports an optional material instance name. Material instance names are required to uniquely identify materials that you use more than once. Repeated materials are useful, for example, in mixing simulations where the motion of a material interface is to be computed: the water in a tank could be divided by creating water material instances named "water_left" and "water_right," and the evolution of the interface between these materials could be simulated.

By default, all Eulerian elements are initially filled with void material, regardless of the section assignment. You must introduce nonvoid material into your Eulerian mesh using an initial condition (see "Initial conditions" below).

Eulerian mesh deformation

The Eulerian time incrementation algorithm is based on an operator split of the governing equations, resulting in a traditional Lagrangian phase followed by an Eulerian, or transport, phase. This formulation is known as "Lagrange-plus-remap." During the Lagrangian phase of the time increment nodes are assumed to be temporarily fixed within the material, and elements deform with the material. During the Eulerian phase of the time increment deformation is suspended, elements with significant deformation are automatically remeshed, and the corresponding material flow between neighboring elements is computed.

At the end of the Lagrangian phase of each time increment, a tolerance is used to determine which elements are significantly deformed. This test improves performance by allowing those elements with little or no deformation to remain inactive during the Eulerian phase. The inactive elements typically have no impact on the visualization of an Eulerian analysis; however, plotting an Eulerian mesh using a very large deformation scale factor may reveal slight deformations for elements within the deformation tolerance.

Eulerian material advection

As material flows through an Eulerian mesh, state variables are transferred between elements by advection. The variables are assumed to be linear or constant in each old element, then these values are integrated over the new elements after remeshing. The new value of the variable is found by dividing the value of each integral by the material volume or mass in the new element.

Second-order advection

Second-order advection assumes a linear distribution of the variable in each old element. To construct the linear distribution, a quadratic interpolation is constructed from the constant values at the integration points of the middle element and its adjacent elements. A trial linear distribution is found by differentiating the quadratic function to find the slope at the integration point of the middle element. The trial linear distribution in the middle element is limited by reducing its slope until its minimum and maximum values are within the range of the original constant values in the adjacent elements. This process is referred to as flux limiting and is essential to ensure that the advection is monotonic.

Second-order advection is used by default, and it is recommended for all problems, ranging from quasi-static to transient dynamic shock.

Input File Usage: *EULERIAN SECTION, ADVECTION=SECOND ORDER

Abaqus/CAE Usage: The second-order advection method is used by default in Abaqus/CAE.

First-order advection

First-order advection assumes a constant value of the variable in each old element. This method is simple and computationally efficient; however, it tends to diffuse sharp gradients over time. Therefore, this technique should be used only as a computationally efficient alternative for quasi-static simulations.

Input File Usage: *EULERIAN SECTION, ADVECTION=FIRST ORDER

Abaqus/CAE Usage: The first-order advection method cannot be specified in Abaqus/CAE.

Reducing the stable time increment based on the advection speed

The stable time increment size is adjusted automatically to prevent material from flowing across more than one element in each increment. When the material velocity approaches the speed of sound (for example, in simulations involving blast and shocks), further restrictions on the time increment size may be needed to maintain accuracy and stability. You can specify a flux limit ratio to restrict the stable time increment size such that material can flow across only a fraction of an element in each increment. The default flux limit ratio is 1.0, and recommended values range from 0.1 to 1.0.

Input File Usage: *EULERIAN SECTION, FLUX LIMIT RATIO=*maximum ratio*

Abaqus/CAE Usage: The flux limit ratio cannot be modified in Abaqus/CAE.

Initial conditions

You can apply initial conditions to Eulerian nodes and elements in the same way that they are used for Lagrangian nodes and elements. Initial stress, temperature, and velocity are common examples. In addition, most Eulerian analyses require the initialization of Eulerian material.

By default, all Eulerian elements are initially void. You can use initial conditions to fill Eulerian elements with one or more of the materials listed in the Eulerian section definition. By selectively filling elements, you can create the initial shape of each Eulerian material.

To fill an Eulerian element, you must define an initial volume fraction for each available material instance. Material is filled until a volume fraction of 1.0 is reached; any excess material is ignored. The initial conditions apply only at the beginning of an analysis; during the analysis the materials deform according to the applied loads, and the volume fractions are recalculated accordingly.

Input File Usage: *INITIAL CONDITIONS, TYPE=VOLUME FRACTION

Abaqus/CAE Usage: Load module: **Create Predefined Field**: **Step: Initial**: choose **Other** for the **Category** and **Material Assignment** for the **Types for Selected Step**

Boundary conditions

By default, Eulerian material can flow freely into and out of the Eulerian domain through mesh boundaries. You can constrain degrees of freedom at Eulerian nodes to restrict material flow. For example, you can define typical fluid “stick” or “sliding” walls using constraints normal and/or tangential to the boundary. Since Eulerian nodes are automatically repositioned during the Eulerian transport phase, you cannot apply prescribed displacement boundary conditions to them.

You can use prescribed velocity or acceleration conditions on Eulerian nodes to control material flow. Prescribed velocity or acceleration is implemented in an Eulerian frame, so material velocity will reach the prescribed value as the material passes the Eulerian node. If velocity is directed outward at an Eulerian mesh boundary, either by prescribed condition or naturally as a result of dynamic equilibrium, material may flow out of the Eulerian domain. This material is lost from the simulation, and corresponding decreases in total mass and energy will occur.

Similarly, if velocity is directed inward at a boundary, inflow of material into the Eulerian domain will occur. When materials flow into an element through a boundary face, the material content and the state of each inflowing material are equal to that which presently exists within the element. For example, if a boundary element contains 60% hot water and 40% cold air and the interface normal is parallel to the boundary face, inflow velocity will introduce a mixture of 60% hot water and 40% cold air. In this case corresponding increases in total mass and energy will occur.

You can also define inflow and outflow conditions at an Eulerian domain boundary, as described in "Defining Eulerian boundaries," Section 14.1.2.

Loads

You can apply loads to Eulerian nodes, elements, and faces in the same way as to their Lagrangian counterparts. Eulerian loads act in an Eulerian frame: they affect Eulerian material as it passes the point of load application.

Material options

You can define material properties for Eulerian analysis in the same way as for Lagrangian analysis. Liquids and gases can be modeled using equation of state materials (see "Equation of state," Section 25.2.1). Brittle cracking is not supported because the number of cracks is not a continous quantity and cannot be easily remapped. Hyperelastic materials can be used in an Eulerian analysis, but due to inaccuracies introduced to the deformation gradient during material transport, these materials might not fully recover their original configuration after loads are removed; the same inaccuracies also affect user-defined materials. The low-density foam material model ("Low-density foams," Section 22.9.1) is not supported.

Eulerian analysis allows materials to undergo extreme strain without the mesh distortion limitations of Lagrangian analysis. Therefore, it is especially important to define your material behavior through the entire strain range, which often requires definition of a failure behavior.

Isotropic material failure is supported using a damage variable to characterize the failure level. Element deletion is suppressed for Eulerian sections because undamaged material may flow into "failed" elements. Shear failure models are not supported.

Rayleigh mass proportional damping is not supported.

Elements

The Eulerian method is implemented in the multi-material element type EC3D8R and the multi-material thermally coupled element type EC3D8RT. The underlying mechanical response formulation of these elements is based on the Lagrangian C3D8R element with extensions to allow multiple materials and to support the Eulerian transport phase. The formulation applies the same strain to each material in the element, then allows the stress and other state data to evolve independently within each material. These stresses are combined using volume fraction data to create element averaged values, which are integrated to obtain nodal forces. Similarly, the thermal response formulation for the thermally coupled element is based on the Lagrangian element C3D8RT with the extension to allow multiple materials with different thermal properties and to support temperature advection. All the materials have the same temperature,

and the thermal properties (such as thermal conductivity and thermal capacitance) are volume averaged before being used in solving one single heat transfer equation for the multi-material model.

Element averaged values of other state data are computed similarly for output purposes.

The Eulerian EC3D8R and EC3D8RT elements require eight nodes. Degenerate elements are not supported. The Eulerian method is not implemented for two-dimensional elements. Axisymmetry can be simulated using a wedge-shaped mesh and symmetry boundary conditions.

By default, the Eulerian elements use viscous hourglass control. Hourglass control is disabled by default for incompressible liquids modeled using equation of state material types. These choices can be modified using section controls (see "Section controls," Section 27.1.4).

Constraints

Since Eulerian nodes are automatically repositioned during the Eulerian transport phase, you cannot use Eulerian nodes in Lagrangian modeling features such as elements, connectors, and constraints. However, constraints between Eulerian materials and Lagrangian parts can be modeled using tied contact interfaces.

Interactions

Eulerian material instances interact with each other with a sticky behavior. This sticking occurs because of the kinematic assumption that a single strain field is applied to all materials within an element. Tensile stress can be transmitted across an interface between two Eulerian materials, and no slip occurs at these interfaces. This Eulerian-to-Eulerian contact behavior can be reasonable in some situations, such as in a simulation of a lead bullet penetrating a steel plate. Ablation of the bullet surface against the steel is captured by the sticky behavior within the Eulerian elements at the bullet-steel interface. Relative motion along this interface will occur only due to shearing of the lead material.

Eulerian-to-Eulerian contact occurs by default in an Eulerian analysis; you do not need to define contact interactions between Eulerian materials.

More complex contact interactions can be simulated when one of the contacting bodies is modeled using Lagrangian elements. This powerful capability supports applications such as fluid-structure interaction, where an Eulerian fluid contacts a Lagrangian structure.

The implementation of Eulerian-Lagrangian contact is an extension of general contact in Abaqus/Explicit. The general contact property models and defaults apply to Eulerian-Lagrangian contact (see "Mechanical contact properties: overview," Section 37.1.1). For example, by default, tensile stresses are not transmitted across an Eulerian-Lagrangian contact interface, and the interface friction coefficient is zero. Specifying automatic contact for an entire Eulerian-Lagrangian model allows for interactions between all Lagrangian structures and all Eulerian materials in the model. You can also use Eulerian surfaces (see "Eulerian surface definition," Section 2.3.5) to create material-specific interactions or to exclude contact between particular Lagrangian surfaces and Eulerian materials.

Input File Usage: Use both of the following options to define contact between all Lagrangian bodies and all Eulerian materials:

```
*CONTACT
*CONTACT INCLUSIONS, ALL EXTERIOR
```

Use the following options to include or exclude contact between particular Lagrangian surfaces and Eulerian materials:

*CONTACT
*CONTACT INCLUSIONS
Lagrangian_surface, *Eulerian_surface*
*CONTACT EXCLUSIONS
Lagrangian_surface, *Eulerian_surface*

Abaqus/CAE Usage: Use the following option to define contact between all Lagrangian bodies and all Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**: **Included surface pairs: All* with self**

Use the following options to include contact between particular Lagrangian surfaces and Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**: **Included surface pairs: Selected surface pairs: Edit**, select the Lagrangian surface in the left column and the Eulerian material instance in the right column, then click the arrows to transfer them to the list of included pairs

Use the following options to exclude contact between particular Lagrangian surfaces and Eulerian materials:

Interaction module: **Create Interaction**: **General contact (Explicit)**: **Excluded surface pairs: Edit**, select the Lagrangian surface in the left column and the Eulerian material instance in the right column, then click the arrows to transfer them to the list of excluded pairs

Formulation of Eulerian-Lagrangian contact

The Eulerian-Lagrangian contact formulation is based on an enhanced immersed boundary method. In this method the Lagrangian structure occupies void regions inside the Eulerian mesh. The contact algorithm automatically computes and tracks the interface between the Lagrangian structure and the Eulerian materials. A great benefit of this method is that there is no need to generate a conforming mesh for the Eulerian domain. In fact, a simple regular grid of Eulerian elements often yields the best accuracy.

If the Lagrangian body is initially positioned inside the Eulerian mesh, you must make sure that the underlying Eulerian elements contain void after material initialization. During the analysis the Lagrangian body pushes material out of the Eulerian elements that it passes through, and they become filled with void. Similarly, Eulerian material flowing toward the Lagrangian body is prevented from entering the underlying Eulerian elements. This formulation ensures that two materials never occupy the same physical space.

If the Lagrangian body is initially positioned outside the Eulerian mesh, at least one layer of void Eulerian elements must be present at the Eulerian mesh boundary. This creates a free surface on the Eulerian material inside the Eulerian mesh boundary and provides a source for void material to replace Eulerian material that is driven out of interior elements. Several layers of void elements are typically

used above free surfaces to allow simulation of crater formation and backsplashing before this material leaves the Eulerian domain.

Eulerian-Lagrangian contact also supports failure and erosion in the Lagrangian body. Lagrangian element failure can open holes in a surface through which Eulerian material may flow. When modeling erosion of a solid Lagrangian body, the interior faces of the solid body must be included in the contact surface definition (see "Modeling surface erosion" in "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1).

Eulerian-Lagrangian contact constraints are enforced using a penalty method, where the default penalty stiffness parameter is automatically maximized subject to stability limits.

Eulerian-Lagrangian contact supports thermal interactions when using coupled temperature-displacement Eulerian element EC3D8RT in a dynamic coupled thermal-stress analysis. However, gap radiation and gap conductance as a function of clearance are not supported.

Output

The set of element output variables EVF gives the Eulerian volume fraction for each material in the Eulerian section definition, including void. It is important to request output for EVF in all Eulerian analyses because visualization of Eulerian material boundaries is based on the material volume fractions.

Material-specific Eulerian field output variables are distinguished by appending material names to the base field name. For example, if you request output variable S (stress components) in an Eulerian analysis involving material instances named "steel" and "tin," you will see results for individual material stresses named "S_steel" and "S_tin."

Several volume fraction averaged field data are also available for output. For example, output variable SVAVG gives a single value of stress for each element computed as a volume fraction average of stress over all materials present in the element. Use of these volume fraction averaged output data has the advantage of substantially reducing the size of the output database for the case where several materials are defined in the Eulerian section. See "Abaqus/Explicit output variable identifiers," Section 4.2.2, for a complete list of Eulerian-specific output variables.

Output variables EVF and SVAVG are included in the PRESELECT variable list when Eulerian elements appear in the model.

You can also request integrated volume (VOLEUL) and integrated mass (MASSEUL) over a particular Eulerian element set. These output variables are material specific and are distingushed by having the material names appended to the variable name.

Limitations

Eulerian analyses are subject to the following limitations:

- Boundary conditions: You cannot apply prescribed nonzero displacement boundary conditions to Eulerian nodes.
- Lagrangian attachments: You cannot attach Lagrangian elements to Eulerian nodes. Use tied contact interfaces instead.
- Constraints: You cannot apply Lagrangian constraints (MPCs, etc.) to Eulerian nodes. Use tied contact interfaces instead.

- Materials: Brittle cracking and shear failure models are also not supported. Rayleigh mass proportional damping is not supported.
- Elements: The Eulerian formulation is implemented only for EC3D8R and EC3D8RT elements.
- Element import: Eulerian elements are not available for import.
- Double-sided contact: Penetration of Eulerian material through the contact interface can occur in some cases involving Eulerian material contacting Lagrangian shell or membrane elements. This type of contact introduces complexity because the sign of the outward normal direction must be determined on the fly as material approaches the Lagrangian element; contact with either side of the element is potentially allowable. You should simplify the contact problem wherever possible by using Lagrangian solid elements instead of shell or membrane elements, since the outward normal direction at solid element faces is unique. For example, if a model involves Eulerian material flowing around a rigid Lagrangian obstacle, mesh the obstacle with solid elements rather than shell elements.
- Contact penetration: In some cases Eulerian material may penetrate through the Lagrangian contact surface near corners. This penetration should be limited to an area equal to the local Eulerian element size. Penetration can be minimized by refining the Eulerian mesh or adding a fillet to the Lagrangian mesh with radius equal to the local Eulerian element size.
- Contact types: Eulerian-Lagrangian contact does not support Lagrangian beam elements, Lagrangian pipe elements, Lagrangian truss elements, or analytical rigid surfaces.
- Contact import: Import of the Eulerian-Lagrangian contact states is not supported.
- Thermal contact: Gap radiation and gap conductance as a function of clearance are not supported.
- Contact output: Contact variables are output only for the Lagrangian side of Eulerian-Lagrangian interfaces.
- Surface loads: You cannot use the Eulerian material surface type for general surface loading. However, distributed loads such as pressure can be applied to surfaces defined on Eulerian element faces.
- Mass scaling: You cannot apply mass scaling to Eulerian elements.
- Heat transfer: Use coupled temperature-displacement EC3D8RT Eulerian elements to model a fully coupled thermal-stress analysis. Adiabatic conditions are assumed in Eulerian materials when EC3D8R elements are used.
- Output: Total strain (LE) is not available for Eulerian elements in field or history output, but it can be accessed via the utility routine **VGETVRM**.
- Subcycling: You cannot include Eulerian elements in subcycling zones.

Input file template

The following example illustrates a coupled Eulerian-Lagrangian analysis of a Lagrangian boat floating on Eulerian water. A conforming mesh is assumed, so Eulerian material initialization is achieved by whole element filling. Material-specific interactions between the Lagragian body and the Eulerian materials are implemented: a contact interaction is defined between the boat and water, but contact

between the boat and air is ignored. Output is requested for Eulerian volume fractions, Eulerian element-averaged stress, and material stress.

```
*HEADING
…
*ELEMENT, TYPE=C3D8R, ELSET=BOAT_ELSET
element definitions for Lagrangian boat
*ELEMENT, TYPE=EC3D8R, ELSET=ALL_EULERIAN
element definitions for whole Eulerian mesh
*ELSET, NAME=AIR_ELSET
data lines giving Eulerian elements that are initially filled with air
*ELSET, NAME=WATER_ELSET
data lines giving Eulerian elements that are initially filled with water
**
*MATERIAL, NAME=AIR
material definition for air
*MATERIAL, NAME=WATER
material definition for water
**
*EULERIAN SECTION, ELSET=ALL_EULERIAN
AIR
WATER
**
*INITIAL CONDITIONS, TYPE=VOLUME FRACTION
AIR_ELSET, AIR, 1.0
WATER_ELSET, WATER, 1.0
*INITIAL CONDITIONS, TYPE=STRESS, GEOSTATIC
data lines to define water pressure due to gravity
**
*SURFACE, NAME=WATER_SURFACE, TYPE=EULERIAN MATERIAL
WATER
*SURFACE, NAME=BOAT_SURFACE
BOAT_ELSET
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
BOAT_SURFACE, WATER_SURFACE
**
```

```
*OUTPUT, FIELD
*ELEMENT OUTPUT
EVF, SVAVG, PEEQVAVG
*END STEP
```

Additional references

- Benson, D. J., “Computational Methods in Lagrangian and Eulerian Hydrocodes,” Computer Methods in Applied Mechanics and Engineering, vol. 99, pp. 235–394, 1992.
- Benson, D. J., “Contact in a Multi-Material Eulerian Finite Element Formulation,” Computer Methods in Applied Mechanics and Engineering, vol. 193, pp. 4277–4298, 2004.
- Peery, J. S., and D. E. Carroll, “Multi-Material ALE methods in Unstructured Grids,” Computer Methods in Applied Mechanics and Engineering, vol. 187, pp. 591–619, 2000.

14.1.2 DEFINING EULERIAN BOUNDARIES

Products: Abaqus/Explicit Abaqus/CAE

References

- "Eulerian analysis," Section 14.1.1
- "Eulerian elements," Section 32.14.1
- *EULERIAN BOUNDARY
- "Defining an Eulerian boundary condition," Section 16.10.21 of the Abaqus/CAE User's Guide

Overview

In an Eulerian analysis you can define independent inflow and outflow conditions at an Eulerian boundary. An Eulerian boundary condition:

- can be used to control the flow of material into the Eulerian domain;
- can be used to define a pressure field at the boundary of an Eulerian domain;
- can be used to apply a nonreflecting boundary condition at the truncated artificial boundary to simulate an infinite domain; and
- is associated with a surface defined on the Eulerian mesh boundary where inflow or outflow occurs.

Defining the Eulerian boundary

Eulerian boundaries must be defined at surfaces on the Eulerian mesh boundary. You cannot define multiple Eulerian boundaries at the same surface.

Input File Usage: *EULERIAN BOUNDARY
surface name

Abaqus/CAE Usage: Load module: **Create Boundary Condition**: **Category: Other**, **Types for Selected Step: Eulerian boundary**: select region

Defining the inflow condition

You can use the inflow condition to control the flow of material into the Eulerian domain.

Free inflow

If no Eulerian boundary is defined, material can flow into the Eulerian domain freely; and the material content and the state of each inflow material are equal to that which presently exists within the element. If an Eulerian boundary is defined, free inflow is the default inflow condition.

Input File Usage: *EULERIAN BOUNDARY, INFLOW=FREE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: Free**

No inflow

You can specify an Eulerian boundary where no inflow can occur—no material or void can flow into the Eulerian domain through the specified boundary. The normal component of the velocity is set to zero if the velocity is directed inward at the boundary, while the tangential component of the velocity remains unchanged.

Input File Usage: *EULERIAN BOUNDARY, INFLOW=NONE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: None**

Void inflow

You can also specify a boundary through which inflow can occur but the influx volume contains only void. Due to the inflow of void, an Eulerian domain that is initially completely full of material might become partially full during the analysis.

Input File Usage: *EULERIAN BOUNDARY, INFLOW=VOID

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Inflow**, **Inflow: Void**

Defining the outflow condition

The outflow condition can be used to simulate an unbounded domain by reducing reflection at the outflow boundary or to prescribe a pressure field at the boundary.

Free outflow

If no Eulerian boundary condition is specified, material can flow out of the Eulerian domain freely; and the material content and the state of each outflow material are equal to that which presently exists within the element. If an Eulerian boundary condition is defined, free outflow is the default behavior if the void inflow condition is specified at the same surface.

Input File Usage: *EULERIAN BOUNDARY, OUTFLOW=FREE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Free**

Nonreflecting outflow

A nonreflecting outflow condition can be used in boundary value problems defined in unbounded domains or problems in which the region of interest is small in size compared to the surrounding medium. Like the infinite element formulation described in “Using solid medium infinite elements in dynamic analyses” in “Infinite elements,” Section 28.3.1, the nonreflecting outflow condition introduces additional normal and shear tractions on the domain boundary that are proportional to the normal and shear components of the velocity of the boundary. These boundary damping constants are chosen to minimize the reflection of dilatational and shear wave energy back into the finite element mesh. This condition does not provide perfect transmission of energy out of the mesh except in the case of plane body waves impinging orthogonally on the boundary in an isotropic medium. However, it usually provides acceptable modeling for most practical cases. An exception is the case when significant material transport occurs through the boundary, in which case this condition is not suitable to be used.

Input File Usage: *EULERIAN BOUNDARY, OUTFLOW=NONREFLECTING

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Nonreflecting**

Equilibrium outflow

Equilibrium outflow is another outflow condition that can effectively reduce spurious reflection at artificial outflow boundaries in unbounded domains. It is assumed that the stress is zero-order continuous across the element faces on the boundary. Traction is applied to these element faces to balance the nodal forces created by the stress in the boundary elements. This condition is usually applied at the outflow boundary where the pressure distribution is unknown.

Input File Usage: *EULERIAN BOUNDARY, OUTFLOW=NONUNIFORM PRESSURE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Equilibrium**

Zero-pressure outflow

It is common in flow problems to specify a zero pressure at the outlet of the flow. Since the normal traction on the boundary contains the contribution from both the pressure and the shear stress, the natural boundary condition, also known as the "do-nothing condition," is not sufficient to provide such a condition if the shear behavior of the flow is also considered. The zero pressure outflow condition applies a traction that counteracts the shear contribution and, thus, generates a uniformly distributed pressure field on the boundary. You can apply a distributed surface load (see "Surface tractions and pressure loads" in "Distributed loads," Section 34.4.3) on the same boundary to specify a nonzero pressure. This is the default outflow condition if the inflow condition is not specified.

Input File Usage: *EULERIAN BOUNDARY, OUTFLOW=ZERO PRESSURE

Abaqus/CAE Usage: Eulerian boundary condition editor: **Flow type: Outflow**, **Outflow: Zero pressure**

Using Eulerian boundaries in restart analyses

You can define a new Eulerian boundary in a restart analysis, but you cannot specify a void inflow condition at this boundary. In addition, you cannot change the inflow condition at an existing Eulerian boundary to the void inflow condition in a restart analysis.

14.1.3 EULERIAN MESH MOTION

Products: Abaqus/Explicit Abaqus/CAE

References

- "Eulerian surface definition," Section 2.3.5
- "Eulerian analysis," Section 14.1.1
- *EULERIAN MESH MOTION
- *EULERIAN SECTION
- *SURFACE
- "Defining an Eulerian mesh motion boundary condition," Section 16.10.22 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

In a traditional Eulerian analysis, material flows through an Eulerian mesh that is fixed in space. Since it is stationary, the Eulerian mesh must be large enough to enclose the entire trajectory of interest. In some simulations, such as a tumbling liquid-filled bottle, this trajectory can be long, requiring a large Eulerian mesh whose elements are mostly empty. The Eulerian mesh motion feature allows the Eulerian mesh to move in space, following, expanding, and contracting to enclose a target object. This can greatly reduce mesh size and, hence, simulation cost. Mesh motion can also simplify modeling by ensuring that the entire trajectory of interest, which may be unpredictable, is indeed covered by the Eulerian mesh.

Activating mesh motion

You can independently activate mesh motion for each Eulerian section in a model. The motion applies to all of the elements in the section.

Input File Usage: *EULERIAN MESH MOTION, ELSET=*name*

Abaqus/CAE Usage: Load module: **BC→Create**, **Category: Other**, **Types for Selected Step: Eulerian mesh motion**: select an Eulerian part instance

Computing mesh motion

The motion of the Eulerian mesh is computed using an internally constructed bounding box that encloses the entire Eulerian section. The bounding box has six degrees of freedom: translation of the box center and scaling of each of the three box dimensions.

The bounding box is constructed in a local coordinate system. Its six degrees of freedom are also defined in this local system. The local coordinate directions remain fixed in space during the simulation. If no local coordinate system is specified, the local system coincides with the global system.

Input File Usage: *EULERIAN MESH MOTION, ORIENTATION= *name*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Bounding Box** **Csys**: **Edit** or **Create**

Defining the target object

You use a surface to define the target object that the Eulerian mesh will follow. By default, the Eulerian mesh bounding box (and, hence, the Eulerian mesh) moves to enclose the surface at all times, subject to any constraints specified on the mesh motion. If the surface type is Lagrangian, the Eulerian mesh bounding box moves to enclose the surface nodes (see Figure 14.1.3–1). If the surface type is Eulerian, the Eulerian mesh bounding box moves to enclose the Eulerian material named in the surface definition (see Figure 14.1.3–2).

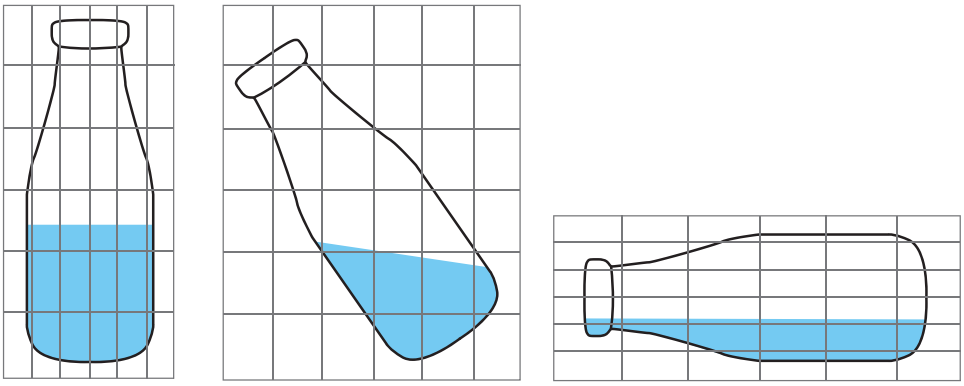


Figure 14.1.3–1 Mesh motion, where the target object is the Lagrangian bottle.

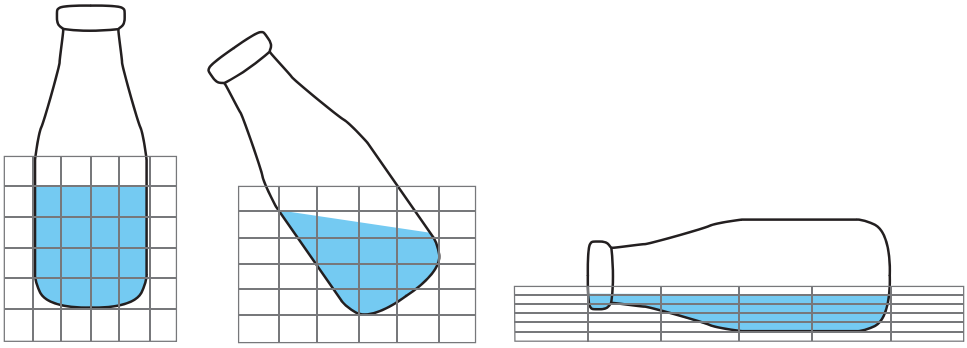


Figure 14.1.3–2 Mesh motion, where the target object is the Eulerian liquid.

The Eulerian mesh may not fully enclose the target object due to:

- constraints on the bounding box motion;

- a misalignment of the bounding box local orientation;
- a mismatch between the shape of the mesh boundary and the bounding box (i.e., the Eulerian mesh is not a rectangular box); or
- an inadequately sized or positioned initial Eulerian mesh.

Input File Usage: *EULERIAN MESH MOTION, SURFACE=*name*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Object to Follow**: *name*

Constraining Eulerian mesh motion

Once the motion of the bounding box is computed, the translations and scaling factors are applied directly to the Eulerian mesh. Several types of constraints are available to restrict these motions. Conflicts between competing constraints are resolved in the following order of precedence:

1. constraining the center and faces of the mesh bounding box,
2. limiting the rate of mesh motion,
3. turning off mesh contraction,
4. centering the mesh bounding box on the target's center of mass or bounding box center,
5. preventing mesh expansion or contraction outside the scale factor limits,
6. limiting aspect ratio changes, and
7. maintaining a buffer between the mesh and target.

Constraining mesh expansion and contraction

By default, the Eulerian mesh may expand or contract by an unlimited amount in each direction, as necessary to contain the target object. This can be undesirable: expansion creates large Eulerian elements that crudely approximate the shape of Eulerian objects, while contraction leads to decreased stable time increment sizes.

You can apply constraints to limit the expansion and contraction independently in each local direction by specifying lower and/or upper limits on the bounding box size scale factors. For example, a maximum scale factor of 1.0 constrains the box dimension to be no larger than 1.0 times the initial box dimension, effectively prohibiting any expansion, while a minimum scale factor of 0.5 limits the box dimension to be no smaller than half its initial dimension.

Input File Usage: *EULERIAN MESH MOTION
scaling factor limits

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Axis *n***:
Expansion ratio, **Contraction ratio**

Preventing mesh contraction

An additional control is available to prevent incremental contraction. If specified, the box dimensions may increase, but at no point during the simulation may they decrease below their current values. This option prevents oscillations in mesh size during simulations where the mesh is nominally expanding.

Input File Usage: *EULERIAN MESH MOTION, CONTRACT=NO

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: toggle off **Allow mesh contraction**

Constraining mesh translation

You can specify the motion of the center of the bounding box to be either free (default) or fixed in each of the local directions. You can also independently specify free (default) or fixed normal motion of the positive and negative box faces in the local coordinate directions.

Input File Usage: *EULERIAN MESH MOTION
,
face constraints
center constraints

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Axis *n***: **Center position**, **Positive plane position**, **Negative plane position**

Centering the mesh bounding box

If the motion of the mesh bounding box is unconstrained, the center of the bounding box is aligned with the center of a box enclosing the target surface. If the target surface fragments or "emits" low density material, aligning the center of the bounding box with the center of mass of the target may be advantageous.

Input File Usage: Use the following option to center the mesh bounding box on the center of mass of the target object:

*EULERIAN MESH MOTION, CENTER=MASS

Use the following option to center the mesh bounding box on the center of the target object's bounding box:

*EULERIAN MESH MOTION, CENTER=BOUNDING BOX

Abaqus/CAE Usage: The center of the mesh bounding box cannot be changed in Abaqus/CAE; the center of the mesh bounding box corresponds to the center of the target object's bounding box.

Controlling the mesh buffer around the target object

The mesh moves to maintain a buffer of Eulerian elements between the target object and the bounding box. By default, this buffer is equal to twice the maximum Eulerian element size in the mesh. You can specify the buffer size as a multiple of the maximum Eulerian element size. You can also specify that the initial spacing between the target object and the mesh (set to zero where the target initially extends outside of the mesh) is used to compute the buffer size.

Input File Usage: Use the following option to use a buffer equal to the initial spacing between the target object and the mesh:

*EULERIAN MESH MOTION, BUFFER=INITIAL

Use the following option to specify a buffer as a multiple of the maximum Eulerian element size:

*EULERIAN MESH MOTION, BUFFER= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Buffer size: Initial** or **Specify**

Limiting aspect ratio changes

Excessive mesh motion in a single direction can produce badly shaped Eulerian elements. An optional parameter is available to limit the change in maximum aspect ratio of the bounding box. By default, this limit is 10. When the aspect ratio limit is reached, motion in one local direction will induce motion in the other directions to preserve the box aspect ratio. This aspect ratio limit applies to the bounding box dimensions, not the underlying Eulerian element dimensions.

Input File Usage: *EULERIAN MESH MOTION, ASPECT RATIO MAX= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Aspect ratio limit:** *value*

Limiting the rate of mesh motion

The Eulerian mesh must not be allowed to move abruptly. A hard limit on its motion is given by the advective Courant condition, which prohibits mesh velocity larger than the material wave speed. In addition you can limit the mesh velocity to a multiple of the maximum velocity in the target object. By default, this limit is set to 1.01.

Input File Usage: *EULERIAN MESH MOTION, VMAX FACTOR= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Mesh velocity factor:** *value*

Ignoring fragments of Eulerian material

When the target object is an Eulerian material, tiny fragments can drive excessive mesh motion. You can specify a minimum Eulerian volume fraction below which Eulerian material is ignored during the mesh motion calculation. This can be particularly useful for impact calculations, where tiny fragments of an impacting, splattering projectile may be allowed to leave the Eulerian domain. The default minimum volume fraction is 0.5.

Input File Usage: *EULERIAN MESH MOTION, VOLFRAC MIN= *value*

Abaqus/CAE Usage: Load module: Eulerian mesh motion editor: **Controls**: **Volume fraction threshold:** *value*

Limitations

An Eulerian mesh can move only according to the available Eulerian mesh motion options. You cannot apply prescribed displacement boundary conditions to Eulerian nodes.

14.1.4 DEFINING ADAPTIVE MESH REFINEMENT IN THE EULERIAN DOMAIN

Product: Abaqus/Explicit

References

- "Eulerian analysis," Section 14.1.1
- *ADAPTIVE MESH REFINEMENT
- *EULERIAN SECTION
- *EULERIAN MESH MOTION
- *CONTACT CONTROLS ASSIGNMENT

Overview

The adaptive mesh refinement feature:

- can refine elements locally inside an Eulerian mesh;
- allows the user to define various criteria for refinement;
- can remove the refinement automatically once the refinement criteria are no longer met; and
- is available for Eulerian element type EC3D8R only.

Adaptive mesh refinement

In a traditional Eulerian analysis the topology of the Eulerian mesh does not change during the analysis. Although the Eulerian mesh motion feature allows the Eulerian mesh to move in space to cover areas of interest, its ability to create a nonuniformly refined mesh that changes with time is limited. The adaptive mesh refinement feature can locally refine the mesh by subdividing elements identified by user-defined criteria. This refinement can be removed automatically during the analysis once the criteria are no longer satisfied. This feature offers great savings in computational cost compared to a uniformly refined mesh. See "Impact of a copper rod," Section 1.3.10 of the Abaqus Benchmarks Guide, for an example of using the adaptive mesh refinement feature.

Activating adaptive mesh refinement

You can independently activate adaptive mesh refinement for each Eulerian section in a model. The feature applies to all the elements specified in the element set; all the elements in the element set have to be in the same Eulerian section.

Input File Usage: *ADAPTIVE MESH REFINEMENT, ELSET=*name*

Setting the refinement limit

When adaptive mesh refinement occurs, elements are added to the Eulerian mesh. You can limit how many elements can be created by specifying an upper bound ratio of added elements to original elements. The default value of this upper bound ratio is 8.0.

Input File Usage: *ADAPTIVE MESH REFINEMENT, RATIO=*maximum increase in number of elements/original number of elements*

Setting the refinement level

With one level of refinement, each time a user-defined Eulerian element is refined, it is equally divided into eight subelements. These subelements can subsequently be divided again if two levels of refinement are allowed. You can set a limit on the maximum number of levels of refinement. The default maximum level is one.

Input File Usage: *ADAPTIVE MESH REFINEMENT, LEVEL=*maximum level of refinement*

Deactivateing coarsening

You can specify whether refinement can be removed when the refinement criteria are no longer met.

Input File Usage: Use the following option to specify that refinement can be removed once the refinement criteria are no longer met:
*ADAPTIVE MESH REFINEMENT, COARSENING=YES (default)
Use the following option to specify that refinement cannot be removed even when the refinement criteria are no longer met:
*ADAPTIVE MESH REFINEMENT, COARSENING=NO

Defining refinement criteria

You must specify at least one refinement criterion. An element will be selected for refinement if any of the criteria is met. To reduce the numerical artifacts at the mesh transition boundaries (where a fine mesh meets a coarse mesh), the elements adjacent to the selected elements are also refined. The elements are coarsened once the refinement criteria are no longer met. Each selected element can be refined or coarsened by only one level in every increment. Table 14.1.4–1 lists all the refinement criteria available in Abaqus/Explicit.

Table 14.1.4–1 Refinement criteria.

Refinement criterion description	Refinement criterion label	User-specified values
Refine elements containing material interfaces	VF	N/A
Refine elements that are in contact with Lagrangian bodies	CONT	You can specify the value ALL to refine all elements intersecting the Lagrangian surfaces even if contact has not occurred; using this option avoids frequent refining and coarsening with chattering contact. You can also specify the value MAT to refine only elements containing materials that are in contact with the Lagrangian surfaces. If no values are specified, MAT will be used except for materials with Mie-Grüneisen equations of state.
Refine elements in which significant plastic deformation occurs. Not supported for the critical state (clay) plasticity model.	PEEQ	Critical value of the equivalent plastic strain
Refine elements near a sharp density gradient	DENSITY	You can specify two values for this criterion. The first value is the critical value of the density gradient, computed as the ratio between the change of density across element faces and the density of the material inside the element; the second value is the critical density. For an element to be selected, both the density and the density gradient must exceed the critical value.
Refine elements near a sharp pressure gradient	PRESS	You can specify two values for this criterion. The first value is the critical value of the pressure gradient, computed as the ratio between the change of pressure across element faces and the pressure of the material inside the element; the second value is the critical pressure. For an element to be selected, both the pressure and the pressure gradient must exceed the critical value.

Input File Usage: *ADAPTIVE MESH REFINEMENT,
refinement criteria label, *value of the criteria*

Contact

When adaptive mesh refinement is specified in an Eulerian section that is involved in general contact, dynamic seeding is activated by default. Dynamic seeding allows more seeds to be created once the Eulerian elements near a Lagrangian face are refined; these seeds will be deleted once these Eulerian elements are coarsened.

Dynamic seeding can also be useful for Eulerian analyses with no adaptive mesh refinement.

Input File Usage: *CONTACT CONTROLS ASSIGNMENT, SEEDING=DYNAMIC

15. Particle Methods

Discrete element method 15.1
Continuum particle analyses 15.2
Particle generator 15.3

15.1 Discrete element method

- “Discrete element method,” Section 15.1.1

15.1.1 DISCRETE ELEMENT METHOD

Products: Abaqus/Explicit Abaqus/Viewer

References

- "Discrete particle elements," Section 33.1.1
- "Particle generator," Section 15.3.1
- *DISCRETE SECTION
- *DISCRETE ELASTICITY
- *CONTACT
- *INITIAL CONDITIONS
- *PARTICLE GENERATOR

Overview

The discrete element method (DEM):

- is intended for modeling events in which large numbers of discrete particles contact each other;
- models each particle with a single-node element that has a rigid spherical shape, which may represent an individual grain, tablet, shot peen, or other simple body;
- is a versatile tool for modeling particulate material behavior in pharmaceutical, chemical, food, ceramic, metallurgical, mining, and other industries; and
- is not meant for modeling deformation of a continuum, but DEM can be used together with finite elements for modeling discrete particles interacting with deformable continua or other rigid bodies.

Introduction

The discrete element method (DEM) is an intuitive method in which discrete particles collide with each other and with other surfaces during an explicit dynamic simulation. Typically, each DEM particle represents a separate grain, tablet, shot peen, etc. DEM is not applicable to situations in which individual particles undergo complex deformation. Therefore, DEM is unlike, and conceptually simpler than, the smoothed particle hydrodynamic (SPH) method in which groups of particles collectively model a continuum body (see "Smoothed particle hydrodynamics," Section 15.2.1).

For example, DEM is well-suited for particle mixing applications, such as that shown in Figure 15.1.1–1. In this application DEM is used to model the initially separated blue and white particles, and rigid finite elements are used to model two mixing augers and the box-shaped container. The sequence of deformed plots in Figure 15.1.1–1 shows the particle response as the augers turn. DEM results for simulations such as this are often best viewed with animations. Another example of using DEM for a mixing application is described in "Mixing of granular media in a drum mixer," Section 13.1.1 of the Abaqus Example Problems Guide.

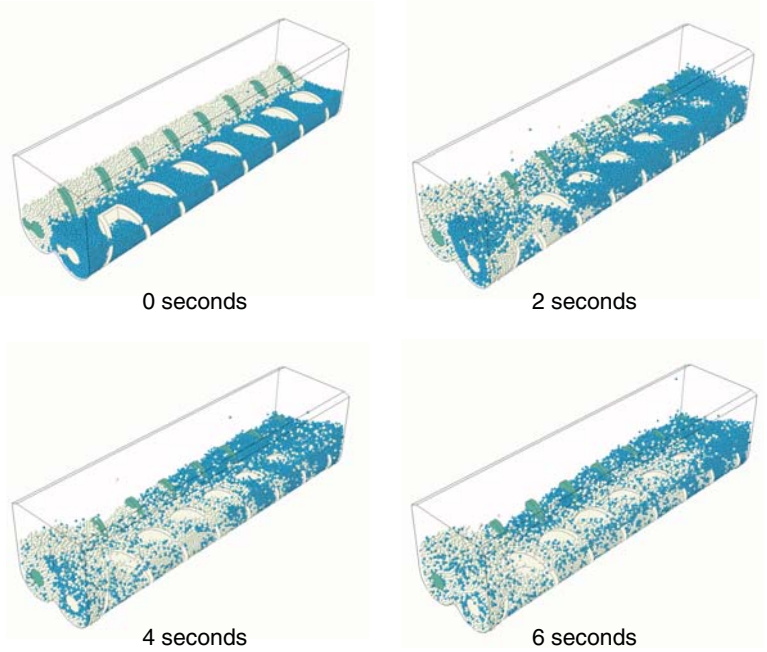


Figure 15.1.1–1 DEM particle mixing example.

Each DEM particle is modeled with a single-node element of type PD3D. These elements are rigid spheres with specified radii. Nodes of PD3D elements have displacement and rotational degrees of freedom. Rotations of DEM particles can significantly influence contact interactions when friction is considered.

General contact definitions are easily extended to include interactions among DEM particles and interactions between DEM particles and finite-element-based (or analytical) surfaces. Large relative motion among particles is typical for DEM applications. Particle-to-particle interactions can involve like or unlike particles. Each particle can be involved in many contact interactions simultaneously. DEM particle interactions use finite contact stiffness, which introduces some compliance into the particle systems. For example, the contact stiffness can be specified such as to reflect the macroscopic stiffness of a packed granular material model with DEM.

For example, consider the interactions between the two spherical particles shown in Figure 15.1.1–2. The three cases show two undeformed spheres just touching, two deformed spheres pushed toward one another with contact strictly enforced, and two rigid spheres pushed toward one another with some penetration. The distance between the centers of the spheres is the same for the cases shown in the middle and on the right in Figure 15.1.1–2. The physical behavior corresponds to the middle case. The case on the right corresponds to a DEM approximation. If the variable δ is defined as

$$\delta = r_1 + r_2 - d,$$

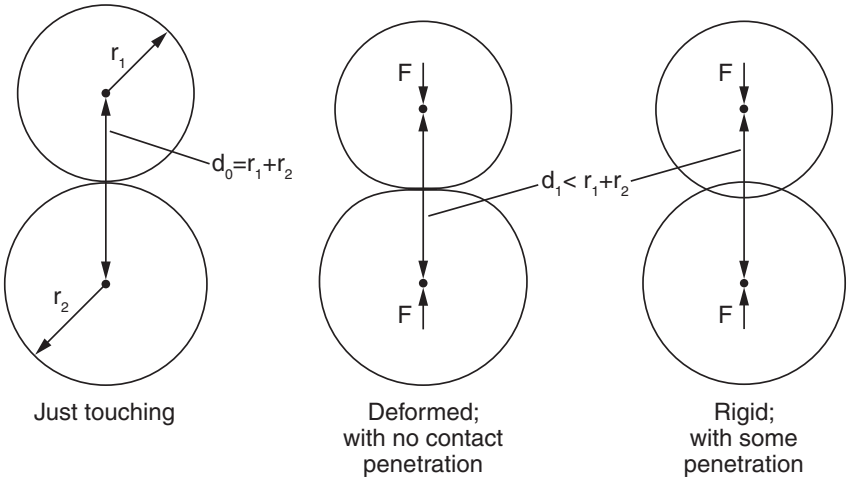


Figure 15.1.1–2 Interactions between spherical particles.

where r_1 and r_2 are the radii of the two spheres and $\boldsymbol{d}$ is the distance between the sphere centers, $\delta = 0$ when the undeformed spheres are just touching and $\delta > 0$ if the distance between the sphere centers is less than the combined radii. For the DEM approximation, δ corresponds to the maximum penetration distance between the particles. You can improve the accuracy for some DEM applications by tuning the contact stiffness relationship (contact force $\boldsymbol{F}$ versus penetration δ) for DEM particles to reflect the Hertz contact solution (middle case in Figure 15.1.1–2). See "Mixing of granular media in a drum mixer," Section 13.1.1 of the Abaqus Example Problems Guide, for further discussion of tuning the contact stiffness.

Applications

DEM is a versatile tool for modeling particulate material behavior in pharmaceutical, chemical, food, ceramic, metallurgical, mining, and other industries. DEM applications include the following categories:

- **Particle packing:** involves processes such as pouring or deposition under gravity (such as sandpiling), vibration after deposition of particles, and compaction.
- **Particle flow:** may occur under gravity only (as in the case of a hopper) or under gravity and other driving forces (such as for mixers and mills).
- **Particle-fluid interaction:** occurs in transport of granular material within a fluid flow, during wavelike motion, and during fluidization (wherein fluid flows upwards through a bed of particles).

DEM can provide insight for many situations that are difficult to investigate with other computational methods or with physical experiments.

Strategies for creating and initializing a DEM model

Particulate media often consist of randomly distributed grains of varying sizes. Generating an initial mesh for a DEM analysis can be challenging. A common strategy for DEM is to specify approximate initial positions of particles with some gaps between them and to allow the particles to settle into position under gravity loading during the first step. For example, this strategy is used for the mixing analysis shown in Figure 15.1.1–1: the augers are kept stationary during the first step in which the particles are allowed to settle, and the augers are turned during the second step to study the mixing behavior (which is the focus of the figure shown). The particle generator can be used to create DEM models (see "Particle generator," Section 15.3.1).

Strategy for reducing solution noise

The solution noise generated by numerous opening and closing contact conditions can be reduced by applying a small amount of mass proportional damping. For more information, see "Alpha damping" in "Discrete particle elements," Section 33.1.1.

Time incrementation considerations

DEM uses the explicit dynamics procedure type. In most cases Abaqus/Explicit automatically controls the time increment size, as discussed in "Automatic time incrementation" in "Explicit dynamic analysis," Section 6.3.3, based on stiffness and mass characteristics of the model. The relationship between the maximum stable time increment size, mass, and stiffness properties is complex. The stable time increment size tends to be proportional to the square root of mass and inversely proportional to the square root of stiffness. However, a stable time increment cannot be computed for each PD3D element because the particles are rigid, so you must specify a fixed time increment size for purely DEM analyses (see "Fixed time incrementation" in "Explicit dynamic analysis," Section 6.3.3). You can use automatic time incrementation for models that have PD3D elements along with regular deformable finite elements.

Contact interactions among DEM particles can influence the appropriate time increment size. DEM analyses without tightly packed particles may simply call for a contact stiffness that is large enough to avoid significant penetrations, rather than a contact stiffness that is highly representative of physical stiffness characteristics of the particles (which are each modeled as rigid with DEM). If you do not specify the contact stiffness, Abaqus/Explicit assigns a default (penalty) contact stiffness based on the time increment size and mass/rotary inertia characteristics of the particles. In such cases you should ensure that the time increment size is small enough to result in a sufficiently large penalty stiffness.

In many cases it is important to carefully set the contact stiffness; for example, to allow DEM results to match Hertz contact behavior between deformable spheres. If you specify the DEM contact stiffness, you must ensure that the time increment used for the analysis is small enough to avoid numerical instabilities. For dense three-dimensional packing of particles where each particle simultaneously contacts many particles, the numerical stability considerations are complex. A general guideline is that the time increment should not exceed $0.4\sqrt{\frac{m}{k}}$, where m and k represent the particle mass and contact stiffness, respectively. In some applications an even smaller time increment, such as $0.1\sqrt{\frac{m}{k}}$, may result in an improved solution.

If particle velocities become very large, the amount of incremental motion can influence the appropriate time increment size. Accurate resolution of particle motion sometimes requires specifying a smaller time increment than the maximum numerical stability time increment.

Initial conditions

Initial conditions pertinent to mechanical analyses can be used in a discrete element method analysis. All of the initial conditions that are available for an explicit dynamic analysis are described in "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Boundary conditions

Boundary conditions are defined as described in "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1. Boundary conditions are rarely applied on individual particles in DEM.

Loads

The loading types available for an explicit dynamic analysis are explained in "Applying loads: overview," Section 34.4.1. Gravity loads are very important for settling and particulate flow analysis in DEM. Concentrated loads are rarely applied on particles.

Elements

The discrete element method uses PD3D elements to model individual particles. These 1-node elements define individual grains of a particulate media, are spherical in shape, and are modeled as rigid (any compliance is built into the contact model). These particle elements use existing Abaqus functionality to reference element-related features such as initial conditions, distributed loads, and visualization. You can define these elements in a similar fashion as you would define point masses or rotary inertia. The coordinates of the node of a particle correspond to the center location of the physical grain of material. PD3D elements are assigned to a discrete section definition, where particle characteristics are specified. For more information, see "Discrete particle elements," Section 33.1.1.

A particle tracking box is established at the beginning of the analysis to define the rectangular region within which the particle search (finding all neighbors for all particles) is performed. You can modify the size of the box as discussed in "Using section controls to define the particle tracking box for DEM and SPH particles" in "Section controls," Section 27.1.4.

Input File Usage: Use the following options to define a discrete element medium:

∗ELEMENT, TYPE=PD3D, ELSET=*particle_body*
element number, *node number*

∗DISCRETE SECTION, ELSET=*element_set_name*

Constraints

Since PD3D elements are Lagrangian elements, their nodes can be involved in other features such as connectors or constraints. Although the PD3D element has a spherical shape, it is possible to model grains of complex shapes by clustering particles together, as illustrated in Figure 15.1.1–3. A cluster is a group of particles that are held together either rigidly or via compliant connections.

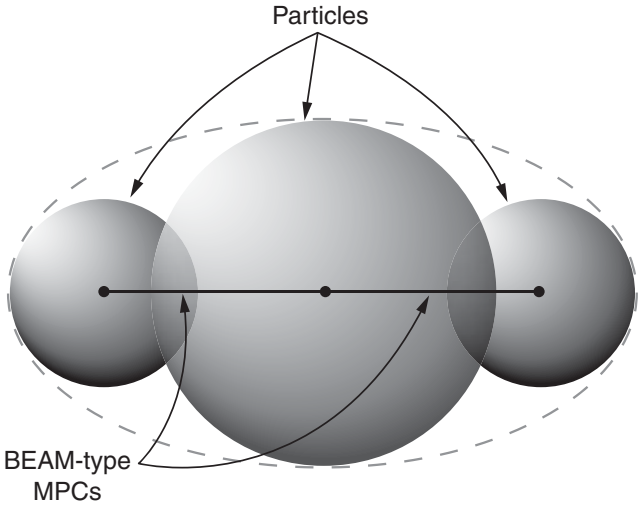


Figure 15.1.1–3 Rigid cluster of particles.

The particles in a cluster may overlap with each other. Contact forces that try to push apart overlapping particles of a cluster will exist unless contact exclusions are specified among particles of a cluster (see "Specifying contact exclusions" in "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1). These contact forces will have no effect on particles held together rigidly but are problematic for clusters with compliant connections.

The particle-cluster approach may not replicate the precise geometry of actual grains. For example, the cluster shown in Figure 15.1.1–3 may approximate an ellipsoidal shape (indicated by the dashed line in the figure). More spherical particles of various sizes can be added to the cluster to obtain a closer approximation of the true shape.

Define BEAM-type multi-point constraints between nodes of a group of particles to create a rigid cluster, or define connector elements between nodes of particles to create a rigid or "deformable" cluster. Appropriate constitutive behavior can be defined for connector elements to capture compliant behavior for particle connections within a cluster. Clusters of overlapping particles that do not involve multi-point constraints or connectors may exhibit nonphysical behavior. For more information, see "General multi-point constraints," Section 35.2.2, and "Connectors: overview," Section 31.1.1.

Interactions

Contact is an essential ingredient for DEM analyses, as discussed above. General contact is used to define contact involving DEM particles. A DEM particle can be involved in multiple contact interactions simultaneously with

- another particle with the same discrete section definition;
- another particle with a different discrete section definition;
- a surface based on finite elements; and
- an analytical rigid surface.

Modeling contact between DEM particles requires that the particles be explicitly included in general contact as element-based surfaces using contact inclusions (see "Element-based surface definition," Section 2.3.2). See "Defining general contact interactions in Abaqus/Explicit," Section 36.4.1, for a discussion of general contact. By default, the particles are not part of the general contact domain, similar to other 1-node elements (such as point masses).

Contact stiffness for DEM is often intended to account for the physical stiffness characteristics of the particles because DEM models each particle as rigid; therefore, nondefault contact property assignments are common for DEM interactions.

Normal and tangential contact forces

Figure 15.1.1–4 is a schematic representation of the contact stiffness and damping between two particles. The spring stiffness K_n acts in the contact normal direction and may represent a simple linear or a nonlinear contact stiffness. The dashpot C_n represents contact damping in the normal direction. The tangential spring stiffness K_t along with the friction coefficient μ represent friction between the particles. The dashpot C_t represents contact damping in the tangential direction.

Figure 15.1.1–4 shows that the tangential contact forces acting on particle surfaces cause moments at particle centers. Interactions involving DEM particles account for moment transfer across the interface.

Hertz normal contact between particles

The Hertz elastic solution relating contact force, F, to the approach distance, δ, for two contacting spherical particles is:

$$F = \frac{4}{3} E^* \sqrt{R} \sqrt{\delta^3},$$

where

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

and

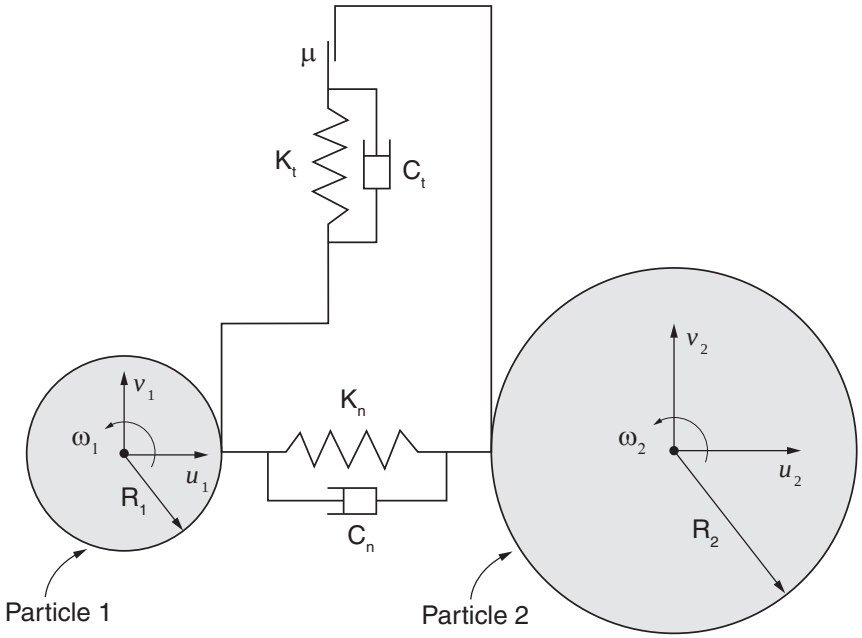


Figure 15.1.1–4 Normal and tangential contact interaction between two discrete elements.

$$\frac{1}{E^*} = \frac{1-{\nu_1}^2}{E_1} + \frac{1-{\nu_2}^2}{E_2}.$$

E_1, ν_1 and E_2, ν_2 are the effective Young's modulus and Poisson's ratio of the two contacting particles, respectively. R_1 and R_2 are the radii of the two contacting particles, respectively. The normal contact stiffness is $K_n = 2E^*\sqrt{R}\sqrt{\delta}$. You must specify the effective Young's modulus and Poisson's ratio for a contacting particle. In addition, you must specify the Hertz-type pressure overclosure.

K_{max} is the limiting value of the normal contact stiffness, K_n, beyond which the normal contact force increases linearly using the slope of the K–δ curve at K_{max}.

Input File Usage: Use the following options to define Hertz-type normal contact behavior:

∗DISCRETE SECTION, ELSET=*element_set_name*
∗DISCRETE ELASTICITY
effective Young's Modulus, *effective Poisson's ratio*
...
∗SURFACE INTERACTION
∗SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=HERTZ
K_{max}

Johnson-Kendall-Roberts adhesive normal contact between particles

The JKR model relates contact force, ***F***, to the contact area, ***a***, as follows:

$$F = \frac{4E^*a^3}{3R^*} - \sqrt{8\pi\Gamma E^* a^3}.$$

The approach distance, δ, for two contacting spherical particles is related to the contact area, ***a***, as follows:

$$\delta = \frac{a^2}{R^*} - \sqrt{\frac{2\pi\Gamma a}{E^*}}.$$

In the above equations

$$R = \frac{R_1 R_2}{R_1 + R_2}$$

and

$$\frac{1}{E^*} = \frac{1-{\nu_1}^2}{E_1} + \frac{1-{\nu_2}^2}{E_2}.$$

In the above equations Γ is the surface energy per unit area of the two contacting particles. Like nonadhesive Hertz contact E_1, ν_1 and E_2, ν_2 are the effective Young's modulus and Poisson's ratio of the two contacting particles, respectively. R_1 and R_2 are the radii of the two contacting particles, respectively. You must specify the effective Young's modulus and Poisson's ratio for a contacting particle. In addition, you must specify the JKR-type pressure overclosure.

K_{max} is the limiting value of the normal contact stiffness, beyond which the normal contact force increases linearly.

Figure 15.1.1–5 shows the force-displacement curve for the JKR adhesion model. Adhesion between particles is triggered upon first contact. The pull-off force, $F_c = -\frac{3}{2}\pi\Gamma R^*$, is the maximum tensile force. Particles continue to experience adhesion force even after physical separation occurs. The adhesion force between two particles goes to zero at a separation distance of

$$\delta_{separation} = -\frac{3}{4}\left(\frac{\pi^2\Gamma^2 R^*}{{E^*}^2}\right)^{\frac{1}{3}}.$$

It can be seen from Figure 15.1.1–5 that adhesive forces are nonzero at $\delta = 0$ and reduce to zero at $\delta = \delta_{F0}$. In some situations it may be desirable to have zero adhesive forces when $\delta = 0$. You can achieve this by applying a horizontal shift of $\delta = \delta_{F0}$ to the force-displacement curve shown in Figure 15.1.1–5. Figure 15.1.1–6 shows the shifted curve. The pull-off force remains unchanged due to the shift, whereas the new separation distance $\delta_{separation_shifted} = \delta_{separation} + \delta_{F0}$. Abaqus automatically computes the horizontal shift δ_{F0} for the "shifted" JKR type adhesive behavior.

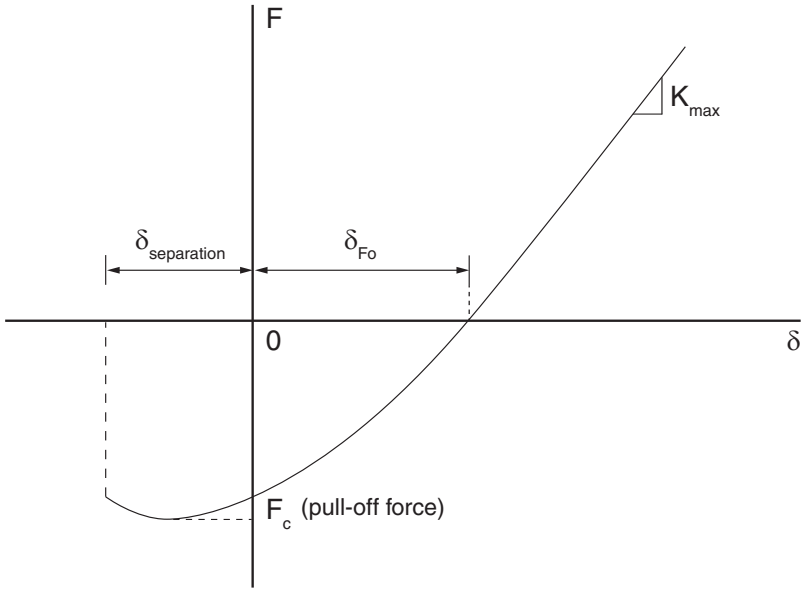


Figure 15.1.1–5 Force versus approach distance relation for the JKR model.

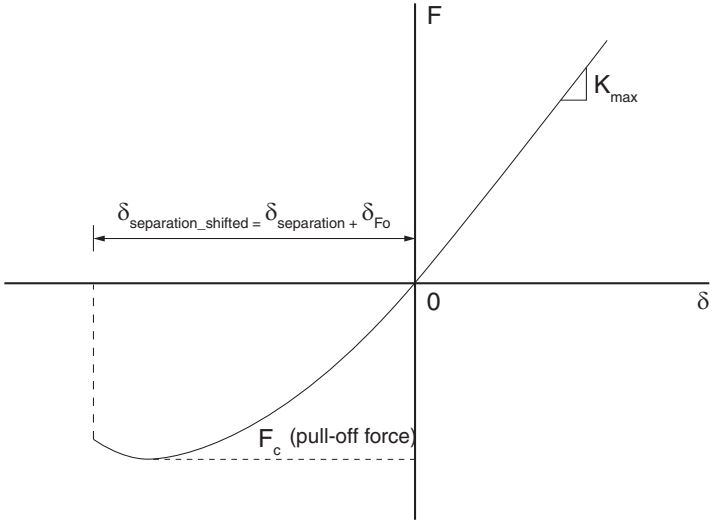


Figure 15.1.1–6 Force versus approach distance relation for the shifted JKR model.

Input File Usage: Use the following options to define JKR-type normal contact behavior:

∗DISCRETE SECTION, ELSET=*element_set_name*
∗DISCRETE ELASTICITY
effective Young's modulus, *effective Poisson's ratio*
...
∗SURFACE INTERACTION
∗SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=JKR
Surface energy per unit area, K_{max}

Use the following options to define the shifted JKR-type normal contact behavior:

∗DISCRETE SECTION, ELSET=*element_set_name*
∗DISCRETE ELASTICITY
effective Young's modulus, *effective Poisson's ratio*
...
∗SURFACE INTERACTION
∗SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=JKR
Surface energy per unit area, K_{max}, *SHIFTED*

Output

No element output is available for PD3D elements. The nodal output includes all output variables generally available in Abaqus/Explicit analyses (see "Abaqus/Explicit output variable identifiers," Section 4.2.2).

Limitations

Discrete element method analyses are subject to the following limitations:

- Volume average output for stress, strain, and other similar continuum element output is not available for DEM analysis.
- Only a spherical shape is supported for PD3D elements.
- PD3D elements cannot be part of a rigid body definition.
- It is not possible to specify cohesive or thermal contact between PD3D elements or between PD3D elements and other elements.
- Rolling friction is ignored for contact between PD3D elements or between PD3D elements and other elements.
- It is not possible to solve problems involving particle-fluid interaction with PD3D elements.
- User-defined surface interaction is not supported for contact between PD3D elements.
- Although supported in Abaqus/Viewer, the functionality is not supported in Abaqus/CAE. You can use the existing functionality in Abaqus/CAE to generate mass elements, write an input file, and then manually edit the input file to convert the mass elements to particles. Alternatively, you can create a mesh using C3D8R elements, write an input file, and then use a script to convert these elements to

particles as described in “Generating particle elements from a solid mesh” in the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base.

DEM computations are distributed across parallel domains except when multiple discrete sections are defined with different alpha damping parameters (which will degrade parallel scalability). DEM analyses are subject to the following limitations if multiple CPUs are used:

- Contact output is not supported for DEM slave nodes.
- Energy history output other than for the whole model is not supported.
- Dynamic load balancing cannot be activated.
- If any DEM particles participate in general contact, all DEM particles must be included in the general contact definition.
- At least 10,000 DEM particles per domain is suggested to achieve good scalability.
- A significant increase in memory usage may be needed if a large number of CPUs is used.

Input file template

The following example illustrates a discrete element method analysis:

```
*HEADING
…
*ELEMENT, TYPE=PD3D, ELSET=name
Element number, node number
…
*DISCRETE SECTION, ELSET=name
Particle radius
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
*NSET, NSET=name, ELSET=name
*SURFACE, NAME=name
,
**
*SURFACE INTERACTION
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
Data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
*CONTACT PROPERTY ASSIGNMENT
**
```

```
*CONTACT CONTROLS ASSIGNMENT
*OUTPUT, FIELD
*END STEP
```

Additional references

- Cundall, P. A., and O. D. Strack, “A Distinct Element Method for Granular Assemblies,” Geotechnique, vol. 29, pp. 47–65, 1979.
- Johnson, K. L., K. Kendall, and A. D. Roberts, “Surface Energy and the Contact of Elastic Solids,” Proceedings of the Royal Society of London, vol. 324, pp. 301–313, 1971.
- Munjiza, A., and K. R. F. Andrews, “NBS Contact Detection Algorithm for Bodies of Similar Size,” International Journal for Numerical Methods in Engineering, vol. 43, pp. 131–149, 1998.
- O’Sullivan, C., and J. D. Bray, “Selecting a Suitable Time Step for Discrete Element Simulations that Use the Central Difference Time Integration Scheme,” Engineering Computations, vol. 21(2/3/4), pp. 278–303, 2004.
- Zhu, H. P., Z. Y. Zhou, R. Y. Yang, and A. B. Yu, “Discrete Particle Simulation of Particulate Systems: A Review of Major Applications and Findings,” Chemical Engineering Science, vol. 63, pp. 5728–5770, 2008.
- Zhu, H. P., Z. Y. Zhou, R. Y. Yang, and A. B. Yu, “Discrete Particle Simulation of Particulate Systems: Theoretical Developments,” Chemical Engineering Science, vol. 62, pp. 3378–3396, 2007.

15.2 Continuum particle analyses

- “Smoothed particle hydrodynamics,” Section 15.2.1
- “Finite element conversion to SPH particles,” Section 15.2.2

15.2.1 SMOOTHED PARTICLE HYDRODYNAMICS

Product: Abaqus/Explicit

References

- "Continuum particle elements," Section 33.2.1
- *SOLID SECTION
- *SECTION CONTROLS
- *INITIAL CONDITIONS

Overview

Smoothed particle hydrodynamics (SPH) is a numerical method that is part of the larger family of meshless (or mesh-free) methods. For these methods you do not define nodes and elements as you would normally define in a finite element analysis; instead, only a collection of points are necessary to represent a given body. In smoothed particle hydrodynamics these nodes are commonly referred to as particles or pseudo-particles.

Introduction

The example shown in Figure 15.2.1–1 contrasts the two approaches. Both discrete representations model the initial configuration of a body of fluid inside a bottle, as described in detail in "Impact of a water-filled bottle," Section 2.3.2 of the Abaqus Example Problems Guide. The model on the left is a traditional tetrahedron mesh of the volume occupied by the fluid. On the right, the same volume is represented by a collection of discrete points. Note that in the latter case there are no edges connecting the points as these points (pseudo-particles) do not require the definition of multiple-node element connectivity, as is the case in the traditional finite element representation on the left. An alternative to directly defining particle elements is to define conventional continuum finite elements and automatically convert them to particle elements at the beginning of the analysis or during the analysis, as discussed in "Finite element conversion to SPH particles," Section 15.2.2.

Smoothed particle hydrodynamics is a fully Lagrangian modeling scheme permitting the discretization of a prescribed set of continuum equations by interpolating the properties directly at a discrete set of points distributed over the solution domain without the need to define a spatial mesh. The method's Lagrangian nature, associated with the absence of a fixed mesh, is its main strength. Difficulties associated with fluid flow and structural problems involving large deformations and free surfaces are resolved in a relatively natural way.

At its core, the method is not based on discrete particles (spheres) colliding with each other in compression or exhibiting cohesive-like behavior in tension as the word particle might suggest. Rather, it is simply a clever discretization method of continuum partial differential equations. In that respect, smoothed particle hydrodynamics is quite similar to the finite element method. SPH uses an evolving interpolation scheme to approximate a field variable at any point in a domain. The value of a variable

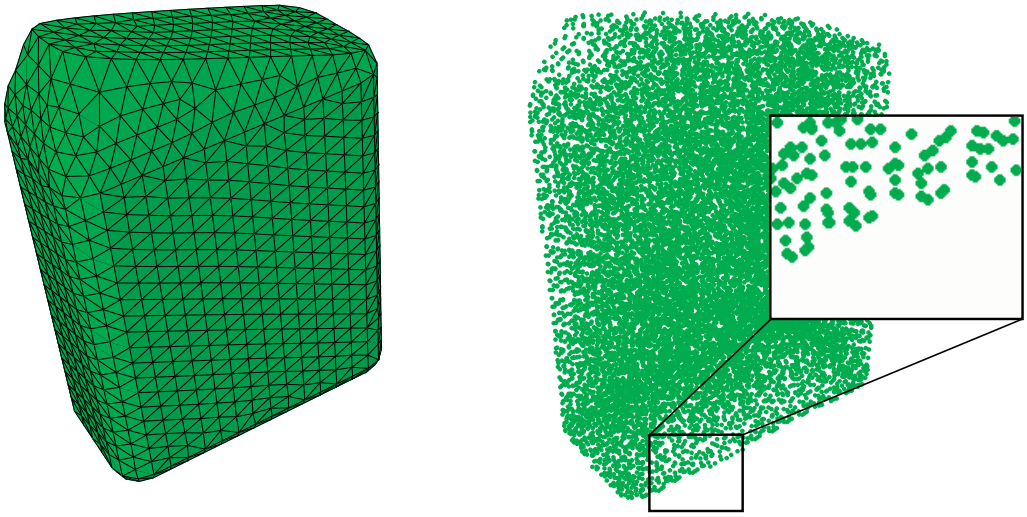


Figure 15.2.1–1 Finite element mesh and SPH particle distribution.

at a particle of interest can be approximated by summing the contributions from a set of neighboring particles, denoted by subscript j, for which the "kernel" function, W, is not zero

$$\langle f(\mathbf{x})\rangle \simeq \sum_j \frac{m_j}{\rho_j} f_j W(|\mathbf{x}-\mathbf{x}_j|, h).$$

An example kernel function is shown in Figure 15.2.1–2. The smoothing length, h, determines how many particles influence the interpolation for a particular point.

The SPH method has received substantial theoretical support since its inception (Gingold and Monaghan, 1977), and the number of publications related to the method is now very large. A number of references are listed below.

The method can use any of the materials available in Abaqus/Explicit (including user materials). You can specify initial conditions and boundary conditions as for any other Lagrangian model. Contact interactions with other Lagrangian bodies are also allowed, thus expanding the range of applications for which this method can be used.

The method is less accurate in general than Lagrangian finite element analyses when the deformation is not too severe and than coupled Eulerian-Lagrangian analyses in higher deformation regimes. If a large percentage of all nodes in the model are associated with smoothed particle hydrodynamics, the analysis may not scale well if multiple CPUs are used.

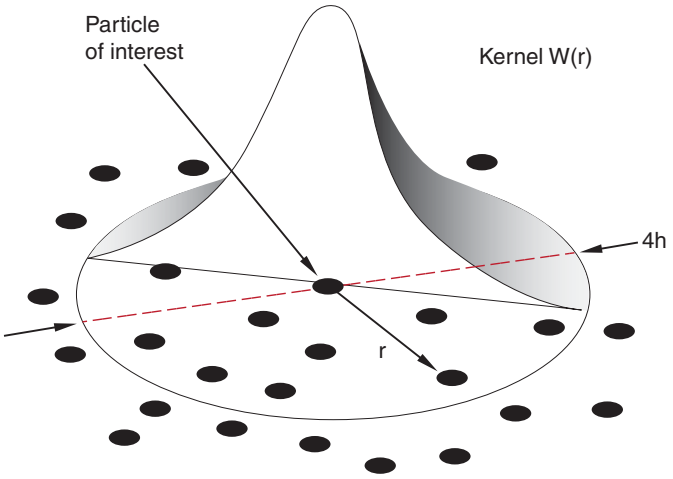


Figure 15.2.1–2 Kernel function.

Applications

Smoothed particle hydrodynamic analyses are effective for applications involving extreme deformation. Fluid sloshing, wave engineering, ballistics, spraying (as in paint spraying), gas flow, and obliteration and fragmentation followed by secondary impacts are a few examples. There are many applications for which both the coupled Eulerian-Lagrangian and the smoothed particle hydrodynamic methods can be used. In many coupled Eulerian-Lagrangian analyses the material to void ratio is small and, consequently, the computational effort may be prohibitively high. In these cases, the smoothed particle hydrodynamic method is preferred. For example, tracking fragments from primary impacts through a large volume until secondary impact occurs can be very expensive in a coupled Eulerian-Lagrangian analysis but comes at no additional cost in a smoothed particle hydrodynamic analysis.

"Impact of a water-filled bottle," Section 2.3.2 of the Abaqus Example Problems Guide, includes an example of using the smoothed particle hydrodynamic method to model the violent sloshing associated with the impact.

Artificial viscosity

Artificial viscosity in smoothed particle hydrodynamics has the same meaning as bulk viscosity for finite elements. Similar to other Lagrangian elements, particle elements use linear and quadratic viscous contributions to dampen high frequency noise from the computed response. In rare cases when the default values are not appropriate, you can control the amount of artificial viscosity included in a smoothed particle hydrodynamic analysis.

Input File Usage: Use the following options to specify scale factors for the linear and quadratic artificial viscosities:

*SECTION CONTROLS
, , , *scale factor for linear artificial viscosity, scale factor for quadratic artificial viscosity*

Initial conditions

"Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, describes all of the initial conditions that are available for an explicit dynamic analysis. Initial conditions pertinent to mechanical analyses can be used in a smoothed particle hydrodynamic analysis.

Boundary conditions

Boundary conditions are defined as described in "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1. The default interaction between SPH particles and Lagrangian surfaces is based on node-to-surface contact. To improve the SPH solution near the Lagrangian boundaries for flow-type problems, you can define the surface behavior explicitly.

Improving the SPH boundary behavior near Lagrangian boundaries

In the SPH method, particles interact with their neighboring particles through a smoothing kernel function. The smoothing kernel function has a finite supporting domain. To achieve accurate SPH simulation results, there should be enough neighboring particles (50–60) within the supporting domain of the smoothing kernel. However, special considerations are needed near the boundaries. When particles are close to Lagrangian boundaries, part of the supporting domain of the smoothing kernel (that beyond the Lagrangian boundaries) will not be filled with any SPH particles. Therefore, the integration accuracy for those SPH particles close to the Lagrangian boundaries may be negatively affected. In addition, the disturbance of the solution close to the boundaries may eventually lead to inaccurate simulation results for the whole field.

To improve the SPH solution near the Lagrangian boundaries, you can specify surface behavior that is based on the ghost particle method (Colagrossi and Landrini, 1977) and define the tangential-type of SPH boundary behavior. When SPH particles are close to Lagrangian boundary surfaces, a virtual plane is formed by taking the average of the adjacent boundary surfaces; based on this virtual plane, ghost particles are created to help improve integration accuracy. The state (pressure, density, velocity, etc.) of the ghost particles is computed from the real physical particles.

Input File Usage: Use one of the following options to specify the SPH surface behavior:

*SPH SURFACE BEHAVIOR, TYPE=FREE SLIP (default)
*SPH SURFACE BEHAVIOR, TYPE=NO SLIP

Loads

The loading types available for an explicit dynamic analysis are explained in "Applying loads: overview," Section 34.4.1. Concentrated nodal loads can be applied as usual. Gravity loads are the only distributed loads allowed in a smoothed particle hydrodynamic analysis.

Smoothed particle hydrodynamic domain

The region within which the particle search (finding all neighbors for all particles) is performed is discussed in "Using section controls to define the particle tracking box for DEM and SPH particles" in "Section controls," Section 27.1.4.

Material options

Any of the material models in Abaqus/Explicit can be used in a smoothed particle hydrodynamic analysis.

Elements

The smoothed particle hydrodynamic method is implemented via the formulation associated with PC3D elements. These 1-node elements are simply a means of defining particles in space that model a particular body or bodies. These particle elements utilize existing functionality in Abaqus to reference element-related features such as materials, initial conditions, distributed loads, and visualization.

You define these elements in a similar fashion as you would define point masses. The coordinates of these points lie either on the surface or in the interior of the body being modeled, similar to the nodes of a body meshed with brick elements. For more accurate results, you should strive to space the nodal coordinates of these particles as uniformly as possible in all directions.

An alternative to directly defining PC3D elements is to define conventional continuum finite element types C3D8R, C3D6, or C3D4 and automatically convert them to particle elements at the beginning of the analysis or during the analysis, as discussed in "Finite element conversion to SPH particles," Section 15.2.2.

The smoothed particle hydrodynamic method implemented in Abaqus/Explicit uses a cubic spline as the interpolation polynomial and is based on the classical smoothed particle hydrodynamic theory as outlined in the references below.

The smoothed particle hydrodynamic method is not implemented for two-dimensional elements. Axisymmetry can be simulated using a wedge-shaped sector and symmetry boundary conditions. There are no hourglass or distortion control forces associated with PC3D elements. These elements do not have faces or edges associated with them.

SPH kernel interpolator and formulations

By default, the smoothed particle hydrodynamic method implemented in Abaqus/Explicit uses a cubic spline as the interpolation polynomial. Alternatively, you can choose a quadratic (Johnson et al, 1996) or quintic (Wendland, 1995) interpolator.

The implementation is based on the classical smoothed particle hydrodynamic theory as outlined in the references below. You also have the option of using a mean flow correction configuration update, commonly referred to in the literature as the XSPH method (see Monaghan, 1992), as well as the corrected kernel of Randles and Libersky, 1997, also referred to as the normalized SPH (NSPH) method. In the XSPH method, the mean velocity filtering coefficient is required for modifying the coordinate update for the particles. A zero value for this coefficient leads to the classical SPH method.

You can control these settings as discussed in "Using section controls for smoothed particle hydrodynamics (SPH)" in "Section controls," Section 27.1.4.

Computing the particle volume

There is currently no capability to automatically compute the volume associated with these particles. Hence, you need to supply a characteristic length that will be used to compute the particle volume, which in turn is used to compute the mass associated with the particle. It is assumed that the nodes are distributed uniformly in space and that each particle is associated with a small cube centered at the particle. When stacked together, these cubes will fill the overall volume of the body with some minor approximation at the free surface of the body. The characteristic length is half the length of the cube side. From a practical perspective, once you have created the nodes, you can use the half-distance between two nodes as the characteristic length. Alternatively, if you know the mass and density of the part, you can compute the volume of the part and divide it by the total number of particles in the part to obtain the volume of the small cube associated with each particle. Half of the cubic root of this small volume is a reasonable characteristic length for this particle set. You can check the mass of individual sets in the model if you request that model definition data be printed to the data (`.dat`) file (see "Model and history definition summaries" in "Output," Section 4.1.1).

Input File Usage: Use the following options to define a smoothed particle hydrodynamic body:

∗ELEMENT, TYPE=PC3D, ELSET=*particle_body*
element number, *node number*

Repeat the data line as often as necessary.

∗SOLID SECTION, ELSET=*particle_body*, MATERIAL=*material_name*
characteristic length associated with particle volume

Smoothing length calculation

Even though particle elements are defined in the model using one node per element, the smoothed particle hydrodynamic method computes contributions for each element based on neighboring particles that are within a sphere of influence. The radius of this sphere of influence is referred to in the literature as the smoothing length. The smoothing length is independent of the characteristic length discussed above and governs the interpolation properties of the method. By default, the smoothing length is computed automatically. As the deformation progresses, particles move with respect to each other and, hence, the neighbors of a given particle can (and typically do) change. Every increment Abaqus/Explicit recomputes this local connectivity internally and computes kinematic quantities (such as normal and shear strains, deformation gradients, etc.) based on contributions from this cloud of particles centered at the particle of interest. Stresses are then computed in a similar fashion as for reduced-integration brick elements, which are in turn used to compute element nodal forces for the particles in the cloud based on the smoothed particle hydrodynamic formulation.

By default, Abaqus/Explicit computes a smoothing length at the beginning of the analysis such that the average number of particles associated with an element is roughly between 50 and 60. The smoothing length can be kept constant or can vary during the analysis. In both cases the average number

of particles per element can either decrease or increase during the analysis depending on whether the average behavior in the model is expansive or compressive, respectively.

If the average number of particles per element decreases below a certain minimum number, the deformation gradient may not be calculated accurately due to the limited support from surrounding particles within the sphere of influence. In this case the deformation gradient is kept constant between the previous time increment and the current time increment. By default, the minimum number of particles associated with one element is 5.

You can control most of these settings as discussed in "Using section controls for smoothed particle hydrodynamics (SPH)" in "Section controls," Section 27.1.4.

Smoothed particle hydrodynamic domain

By default, all particles are tracked as the analysis progresses. Alternatively, you can define a particle tracking box to define the region within which the particle search (finding all neighbors for all particles) is performed as discussed in "Using section controls to define the particle tracking box for DEM and SPH particles" in "Section controls," Section 27.1.4.

Constraints

Since the PC3D elements are Lagrangian elements, their nodes can be involved in other features, such as other elements, connectors, or constraints. Since these elements do not have faces or edges, an element-based surface cannot be defined using PC3D elements. Consequently, constraints that require element-based surfaces (such as fasteners) cannot be defined for particles.

Interactions

Bodies modeled with particles can interact with other finite element meshed bodies via contact. The contact interaction is the same as any contact interaction between a node-based surface (associated with the particles) and an element-based or analytical surface. Both general contact and contact pairs can be used. All interaction types and formulations available for contact involving a node-based surface are allowed, including cohesive behavior. Different contact properties can be assigned via the usual options. By default, the particles are not part of the general contact domain similar to other 1-node elements (such as point masses). The default contact thickness for particles is the same value specified as the characteristic length on the section definition; thus, for contact purposes, particles behave as spheres with radii equal to the radius of a sphere inscribed in the small cube associated with the particle volume as described above.

You should not specify a contact thickness of zero for the nodes associated with PC3D elements or contact may not be resolved robustly. The recommended approach is to use the default or specify a reasonable contact thickness.

Interaction between different bodies all modeled with PC3D elements is allowed. However, this interaction is meaningful only in cases when the colliding smoothed particle hydrodynamic bodies are made of the same fluid-like material, such as a water drop falling in a bucket partially filled with water. In solids-related applications, such as modeling a bullet perforating an armor plate, one of the bodies must be modeled using regular finite elements.

Contact interactions cannot be defined between particles and Eulerian regions.

Input File Usage: Use the following options to define contact between a meshed or analytical surface with a particle-based surface:

∗CONTACT
∗CONTACT INCLUSIONS
node-based particle surface, *element-based/analytical_surface*

Output

The element output available for PC3D elements includes all mechanics-related output for continuum elements: stress; strain; energies; and the values of state, field, and user-defined variables. The nodal output includes all output variables generally available in Abaqus/Explicit analyses.

Particles can be visualized in Abaqus/CAE via circular discs. In contour plots the values of field output variables are shown as circular patches of color. Symbol plots are also available.

Limitations

Smoothed particle hydrodynamic analyses are subject to the following limitations:

- They are less accurate in general than Lagrangian finite element analyses when the deformation is not too severe and the elements are not distorted. In higher deformation regimes coupled Eulerian-Lagrangian analyses are also generally more accurate. The smoothed particle dynamic method should be used primarily in cases when the conventional finite element method or the coupled Eulerian-Lagrangian method have reached their inherent limitations or are prohibitively expensive to perform.
- When the material is in a state of tensile stress, the particle motion may become unstable leading to the so-called tensile instability. This instability, which is strictly related to the interpolation technique of the standard smoothed particle dynamic method, is especially noticeable when simulating the stretched state of a solid. As a consequence, particles tend to clump together and show fracture-like behavior.
- Mass distribution in a body defined with particle elements is somewhat different when compared to the mass distribution of the same body defined with continuum elements, such as C3D8R elements. When particle elements are used, the volume of all particles in that body are the same. Consequently, the nodal mass associated with all particles in that body is the same. If the nodes are not placed in a regular cubic arrangement, the mass distribution is somewhat inexact, particularly at the free surface of the body being modeled.
- Surface loads cannot be specified on PC3D elements. However, distributed loads, such as pressure, can be applied to other finite element surfaces that can apply a pressure onto the particle elements via contact interactions.
- Bodies modeled with particles that were not defined using the same section definition do not interact with each other. Hence, you cannot use smoothed particle hydrodynamics to model the mixing of bodies with dissimilar materials.
- Selective subcycling and the contact pair algorithm are not supported.

- Rough friction should not be used on surface interactions defined for SPH particles.
- The functionality is not directly supported in Abaqus/CAE. However, you can do the following:
  - You can use the existing functionality in Abaqus/CAE to generate mass elements, write an input file, and then manually edit the input file to convert the mass elements to particles.
  - You can use finite element conversion to SPH particles (see "Boundary conditions" in "Finite element conversion to SPH particles," Section 15.2.2) at the beginning of an analysis (a time-based conversion criterion at time equal to zero).
  - You can create a mesh using C3D8R elements, write an input file, and then use a script to convert these elements to particles as described in "Generating particle elements from a solid mesh" in the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base.
- Within a given body (part) defined via one solid section definition, gravity loads and mass scaling cannot be specified selectively on a subset of elements referenced by this definition. Instead, the two features must be applied to all the elements in the element set associated with the solid section definition.

Smoothed particle hydrodynamic computations are distributed across parallel domains in most cases; however, they are all performed by a single domain (with a single processor) for models with any of the following characteristics (which will often dramatically degrade parallel scalability):

- Finite element conversion to SPH particles (see "Boundary conditions" in "Finite element conversion to SPH particles," Section 15.2.2) after the beginning of an analysis. The parallel smoothed particle hydrodynamics implementation supports conversion to SPH particles only at the beginning of an analysis (at time equal to zero).
- Multiple solid sections for PC3D elements
- Normalized kernel specified as a section control
- Predefined field variable (including temperature) dependence of material properties

Smoothed particle hydrodynamic analyses are subject to the following limitations if multiple CPUs are used:

- Contact output is not supported for smoothed particle hydrodynamic slave nodes.
- Element history output is not supported.
- Dynamic load balancing cannot be activated.
- If any SPH particles participate in general contact, all SPH particles must be included in the general contact definition.
- At least 10,000 particles per domain is suggested to achieve good scalability.
- A significant increase in memory usage may be needed if a large number of CPUs are used.

Input file template

The following example illustrates a smoothed particle hydrodynamic analysis of a bottle filled with fluid being dropped on the floor. The plastic bottle and the floor are modeled with conventional shell elements. The fluid is modeled via smoothed particle hydrodynamics using PC3D elements. The nodal coordinates

of the particles are defined such they are all located inside the bottle. Material property definitions are defined as usual for both the fluid and the bottle. Contact interaction is defined between the smoothed particle hydrodynamic particles representing the water (node-based surface) and the interior walls of the bottle and also between the bottle exterior and the floor using element-based surfaces (not shown). Output is requested for stresses (pressure) and density in the fluid.

```
*HEADING
…
*ELEMENT, TYPE=PC3D, ELSET=Fluid_Inside_The_Bottle
Element number, node number
…
*SOLID SECTION, ELSET=Fluid_Inside_The_Bottle, MATERIAL=Water
Element characteristic length associated with particle volume
*MATERIAL, NAME=Water
Material definition for water, such as an EOS material
*ELEMENT, TYPE=S4R, ELSET=Plastic_Bottle
Element definitions for the shells
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
*NSET, NSET=Water_Nodes, ELSET=Fluid_Inside_The_Bottle
*SURFACE, NAME=Water_Surface, TYPE=NODE
Water_Nodes,
*SURFACE, NAME=Bottle_Interior
Plastic_Bottle, SNEG
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
Data lines to define gravity load
**
*CONTACT
*CONTACT INCLUSIONS
Water_Surface, Bottle_Interior
**
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=Fluid_Inside_The_Bottle
S, DENSITY
*END STEP
```

Additional references

- Colagrossi, A., and M. Landrini, "Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics," Journal of Computational Physics, 2003.
- Gingold, R. A., and J. J. Monaghan, "Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars," Royal Astronomical Society, Monthly Notices, vol. 181, pp. 375–389, 1977.
- Johnson, J., R. Stryk, and S. Beissel, "SPH for High Velocity Impact Calculations," Computer Methods in Applied Mechanics and Engineering, 1996.
- Libersky, L. D., and A. G. Petschek, "High Strain Lagrangian Hydrodynamics," Journal of Computational Physics, vol. 109, pp. 67–75, 1993.
- Monaghan, J., "Smoothed Particle Hydrodynamics," Annual Review of Astronomy and Astrophysics, 1992.
- Munjiza, A., and K. R. F. Andrews, "NBS Contact Detection Algorithm for Bodies of Similar Size," International Journal for Numerical Methods in Engineering, vol. 43, pp. 131–149, 1998.
- Randles, P. W., and L. D. Libersky, "Recent Improvements in SPH Modeling of Hypervelocity Impact," International Journal of Impact Engineering, 1997.
- Swegle, J. W., and S. W. Attaway, "An Analysis of Smoothed Particle Hydrodynamics," Sandia National Lab Report SAND93–2513, 1994.
- Wendland, H., "Piecewise Polynomial, Positive Definite and Compactly Supported Radial Functions of Minimal Degree," Advances in Computational Mathematics, 1995.

15.2.2 FINITE ELEMENT CONVERSION TO SPH PARTICLES

Products: Abaqus/Explicit Abaqus/CAE

References

- "Continuum particle elements," Section 33.2.1
- *CONTACT
- *INITIAL CONDITIONS
- *OUTPUT
- *SECTION CONTROLS
- *SOLID SECTION

Overview

You can take advantage of the intrinsic strengths of both Lagrangian finite element and SPH methods when modeling a body. You can define the model with Lagrangian finite elements and convert them to SPH particles either at the beginning of an analysis or after the deformation becomes significant. It is sometimes easier to create the mesh with Lagrangian finite elements, and Lagrangian finite elements are often more accurate for small deformations. SPH methods are well suited for large deformation.

You start by defining a part as usual. You mesh the part with C3D8R, C3D6, or C3D4 reduced-integration elements or a combination of these elements. You then specify that these "parent" elements are to convert to internally generated SPH particles when a user-specified criterion is met. Gravity loads, contact interactions, initial conditions, mass scaling, and output requests associated with the parent elements or nodes of the parent elements will be transferred appropriately to the generated particles upon conversion in an intuitive way as explained below. A special formulation is used to ensure the smoothest possible transition between the two modeling methods. The technique can use any of the materials available in Abaqus/Explicit (including user materials).

Activating the conversion to SPH particles functionality

Two conversion techniques are available for converting Lagrangian finite elements to SPH particles: particles can be generated per parent element, or particles can be generated based on a uniform background grid.

The conversion technique of generating particles per parent element is intended to be used when the deformations in the original finite element mesh are significant and elements may distort. Traditionally, in such cases deletion of the soon-to-be distorted Lagrangian elements would be the only choice to allow the analysis to continue. Converting to SPH particles offers an improvement over the element deletion method because the generated particles are able to provide resistance to deformation beyond finite element distortion levels. Consequently, element deletion cannot be used together with element conversion.

The conversion technique of generating particles based on a uniform background grid is intended to generate a uniform distribution of SPH particles at the beginning of the analysis. This is useful in applications where the SPH functionality is the preferred modeling method (such as when modeling fluids) and a uniform distribution of particles is desired because it usually leads to improved accuracy of results.

The element conversion to particles functionality is not active by default.

Generating particles per parent element

You can control the number of particles generated per parent element and choose between one of four criteria to specify when the conversion is to be triggered.

Input File Usage: *SECTION CONTROLS, ELEMENT CONVERSION=YES

Abaqus/CAE Usage: Mesh module: **Mesh→Element Type**: **Conversion to particles**: **Yes**

Specifying the number of particles to be generated

By default, one particle is generated per parent element. You can control the number of particles generated per element by specifying the number of particles to be generated per parent element isoparametric direction. The total number of particles generated per element depends on the element type that is being converted. For example, if you specify 3 particles to be generated per isoparametric direction, upon conversion 27 particles would be generated from a C3D8R element, 18 from a C3D6 element, and 10 from a C3D4 element, as illustrated in Figure 15.2.2–1. A maximum value of seven particles per direction can be specified. The particles are evenly spaced inside the parent element such that they fill the volume as uniformly as possible. For example, if cubic parent elements are stacked in the user-defined mesh, the particles would be evenly spaced throughout the part.

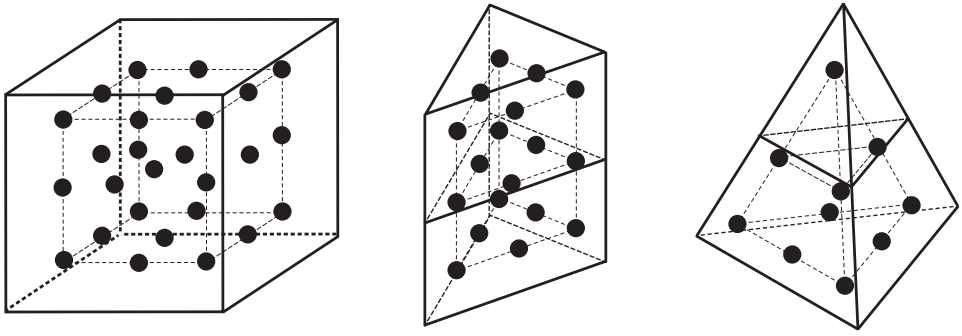


Figure 15.2.2–1 Internally generated particles per parent element illustrated for three particles per isoparametric direction.

You can control the number of particles generated per isoparametric direction as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

Time-based criterion

You can specify the time when the conversion of all the elements in the affected element set is to take place regardless of the deformation levels. This option is intended for applications where the SPH functionality is the preferred modeling method, such as fluid sloshing in a tank or a synthetic bird strike on an aircraft. If the conversion time is specified as zero, the conversion takes place at the beginning of the analysis. For example, fluid sloshing is a good candidate for using a time-based criterion if sloshing is expected to start at the beginning of the analysis. You can specify a later time at which the conversion takes place if extreme deformations do not occur until later in the analysis. A bird strike analysis is a potential candidate as the bird might travel for some time without any deformation prior to hitting the intended target.

You can control the time when the conversion is to occur as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

Strain-based criterion

You can specify the absolute value of the maximum principal strain when the conversion of a given element is to take place. As elements deform, if the absolute value of the maximum principal strain is greater than the specified threshold, the parent elements will convert progressively to SPH particles. This option is intended for applications where the finite element method is the preferred modeling method but severe deformations could occur in certain regions. Examples include blast applications and crushing.

You can control the strain-based threshold upon which conversion is to occur as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

Stress-based criterion

You can specify the absolute value of the maximum principal stress value at which the conversion of a given element takes place. As elements deform, if the absolute value of the maximum principal stress is greater than the specified threshold, the parent elements will convert progressively to SPH particles. This option is intended for the same candidate applications as those discussed for the strain-based criterion.

You can control the stress-based threshold upon which conversion is to occur as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

User subroutine–based criterion

The user subroutine–based criterion provides the flexibility of a user subroutine implementation that allows you to implement your own conversion criterion. Element conversion can be controlled during the course of an Abaqus/Explicit analysis through any of the user subroutines that can actively modify state variables associated with a material point, such as **VUSDFLD** and **VUMAT**. You specify the state variable number controlling the element conversion flag. For example, specifying a state variable number of two indicates that the second state variable is the conversion flag in the user subroutine. The conversion state variable should be set to a value of one or zero. A value of one indicates that the element is active, while a value of zero indicates that Abaqus/Explicit should convert the element to particles. Since user subroutines have access via arguments (or in the case of the **VUSDFLD** subroutine via utility routines)

to material point state data, the functionality provides a comprehensive means to define the conversion state variable.

Input File Usage: Use the following options to define a user subroutine–based conversion criterion:

```
*SECTION CONTROLS, ELEMENT CONVERSION=YES,
CONVERSION CRITERION=USER
...
*MATERIAL
*DEPVAR, CONVERT=variable number
```

Abaqus/CAE Usage: Specifying a user subroutine–based criterion for element conversion is not supported in Abaqus/CAE.

Conversion to particles formulation

When using the conversion technique of generating particles per parent element, particles are generated internally at the beginning of the preprocessing phase of the analysis, and they are placed in an inactive or dormant state. The particles are attached to the parent elements in a similar fashion as the nodes of embedded elements are attached (see "Embedded elements," Section 35.4.1), and they follow the motion of the parent element nodes in an average sense. The inertial properties of the particles in this inactive state (while the parent finite elements are active) are disregarded automatically to avoid doubling the momentum at a given location. Similar to SPH particles defined directly as PC3D elements, particles generated from parent element sets associated with different section definitions will not interact with each other.

Upon conversion a number of internally generated particles per parent element are activated, as illustrated for various element types in Figure 15.2.2–1. The computational cost of the analysis can increase significantly after conversion takes place if a large number of particles are generated per element since a larger number of active elements needs to be processed. In addition, the computational cost increases because the stable time increment associated with the internally generated particles decreases as the particle density increases.

Upon conversion the state information (such as stress or equivalent plastic strain) associated with the element being converted is transferred to the generated particles to ensure the smoothest possible transition. The activated particles interact via the SPH formalism with both the previously activated particles and the neighboring inactive particles that are still embedded in active parent elements.

Generating particles based on a uniform background grid

This conversion technique allows you to generate a uniform distribution of particles based on a uniform background grid. Using an initially uniform distribution of particles is usually recommended since it can help improve the accuracy of the results.

The origin of the background grid is placed at the center of mass of the element set to be converted. The finite elements in the set are converted to SPH particles at the beginning of the analysis. The particles are generated only at grid line intersections that are inside the volume of a parent element. All generated

particles have the same volume and mass. Any parent element that does not contain a grid intersection has no generated particles associated with it.

You can control the spacing and orientation of the uniform background grid as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

Input File Usage: *SECTION CONTROLS, ELEMENT CONVERSION=BACKGROUND GRID

Abaqus/CAE Usage: Generating particles based on a uniform background grid is not supported in Abaqus/CAE.

Specifying the thickness of generated particles

The thickness of the particles is primarily used for resolving initial overclosures between the particles and the surfaces in the general contact domain. By default, Abaqus/Explicit determines the particle thickness such that no initial overclosures exist, as illustrated in Figure 15.2.2–2(b). You can specify a uniform thickness, although the particle thickness can expand beyond the original surface of the finite element mesh, as illustrated in Figure 15.2.2–2(a), which could lead to initial overclosures. The thickness of the particles does not affect the particle's volume and mass.

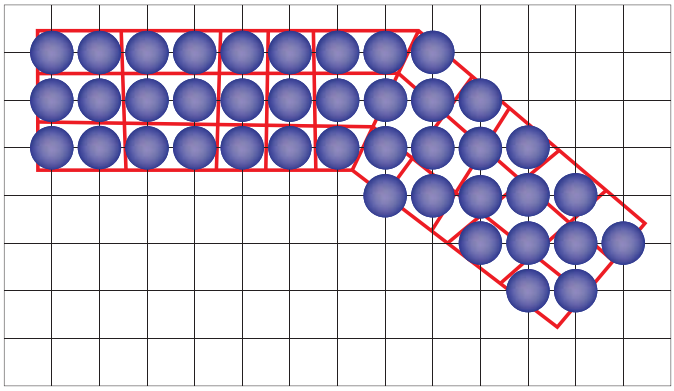
You can specify the thickness of the generated particles to be either variable or uniform as discussed in "Using section controls to convert continuum elements to particles" in "Section controls," Section 27.1.4.

Automatically generated sets and surfaces

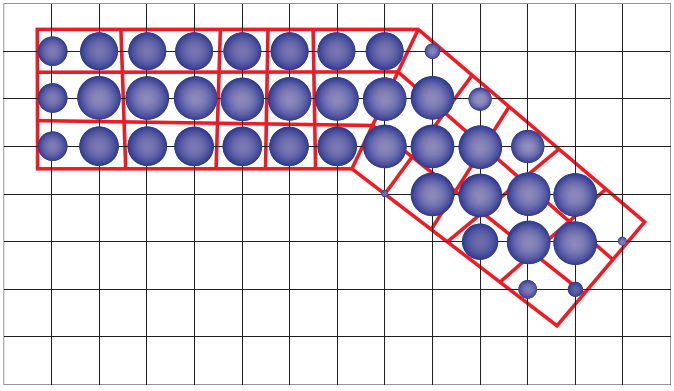
Since the particles are generated internally, you do not have the ability to define element sets, node sets, or surfaces associated with these particles. Consequently, a number of sets and surfaces are created internally for convenience. You can visualize these internal sets and surfaces via the usual techniques. Table 15.2.2–1, Table 15.2.2–2, and Table 15.2.2–3 describe the internally generated sets and surfaces.

Table 15.2.2–1 Internally generated element sets.

Internally generated element set	Description
ALL_GENERATED_ELEMENTS_SPH	All generated SPH particles in the entire model
ALL_PARENT_ELEMENTS_SPH	All parent elements in the entire model
*UserDefinedElsetName*_SECT_SPH	All generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefined_AElsetName*_SPH	All generated particles associated with the element set *UserDefined_AElsetName*



(a) particles with uniform thickness



(b) particles with variable thickness

Figure 15.2.2–2 Internally generated particles based on a uniform background.

Table 15.2.2–2 Internally generated node sets.

Internally generated node set	Description
ALL_PARENT_ELEMENT_NODES_E_SPH	All nodes of all parent elements in the entire model

Internally generated node set	Description
ALL_GENERATED_NODES_SPH	All nodes of all generated particles in the entire model
*UserDefinedElsetName*_SECT_E_SPH	All nodes of generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefined_ANsetName*_SPH	Nodes of generated particles from parent elements touching nodes of the *UserDefined_ANsetName* node set

Table 15.2.2–3 Internally generated surfaces.

Internally generated surfaces	Description
*UserDefinedElsetName*_PARENT_EE_SPH	Element-based surface containing all facets of all elements associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefinedElsetName*_SECT_NE_SPH	Node-based surface with all nodes of all generated particles associated with the *UserDefinedElsetName* element set used in the section definition
*UserDefinedSurfaceName*_NS_SPH	Node-based surface containing all nodes of generated particles associated with the elements used in the definition of the *UserDefinedSurfaceName* element-based surface

These sets and surfaces are used by features that are automatically generated internally, such as loads, initial conditions, mass scaling, contact definitions, and output requests. These internally generated features extend the features that you have defined for the associated parent sets and surfaces to internally generated particles. In all cases the internally generated features preserve the attributes that you have defined.

Initial conditions

Initial conditions (see "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1) cannot be specified directly for the generated particles. However, a subset of the possible initial conditions (stresses, velocity and rotating velocity) is applied to the generated particles automatically. You specify these initial conditions on the original element or node set you have defined in the model, and they are applied appropriately to the associated generated particles. The initial conditions are

applied via the internally created sets described above; hence, you must use an element or node set rather than element or node numbers when applying initial conditions.

Initial stresses specified on parent elements are applied to the generated particles. This feature is leveraged in cases where parent elements convert to particles at the very beginning of the analysis (time zero). All other initial conditions associated with elements are taken into account for the generated particles as long as the parent elements convert to particles after the first increment in the analysis. The state transfer mechanism described above appropriately transfers the information to particles and, hence, initial conditions are accounted for correctly in the particles.

Boundary conditions

Boundary conditions (see "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1) cannot be applied directly to the generated particles. Boundary conditions applied to nodes of the parent elements are not transferred to the generated particles. However, you can use contact interactions to enforce boundary conditions as explained in "Interactions."

Temperature and field variables specified on node sets that include parent element nodes are extended to the generated particles. Abaqus/Explicit generates corresponding temperature and field variables definitions internally via the internal node sets described in "Automatically generated sets and surfaces." If all of the nodes of a particular parent element have the same value at a given time, the generated particles would have that same value as well. If different values are specified, no interpolation occurs. Instead, the value of the last definition is used.

Loads

The loading types available for an explicit dynamic analysis are explained in "Applying loads: overview," Section 34.4.1. Concentrated nodal loads cannot be applied to generated particles. Gravity loads specified on the parent elements are the only distributed loads that are transferred upon conversion to the generated particles.

Material options

Any of the material models in Abaqus/Explicit can be used with the conversion technique.

Elements

When using the conversion technique and C3D8R, C3D6, and/or C3D4 reduced-integration parent elements to define the part, PC3D elements are generated internally at the beginning of the analysis; the parent elements are active, and the PC3D elements are inactive. Upon conversion the active status switches. At no time are a parent element and the associated generated particles both active. By default, the Visualization module automatically displays only the elements that are active at any given time.

Particle mass (and volume) is computed automatically from the mass (volume) of the parent element. All particles associated with a specific parent element will have the same mass (volume). The SPH smoothing length and domain required for the SPH formalism are computed in the same fashion as in the case when you define PC3D elements directly (see "Smoothed particle hydrodynamics," Section 15.2.1).

If mass scaling is defined on element sets containing parent elements, Abaqus/Explicit internally generates mass scaling definitions associated with the corresponding internal element sets described in "Automatically generated sets and surfaces."

Constraints

Constraints such as couplings or ties cannot be applied directly to the generated particles. However, constraints can be defined on nodes and surfaces associated with the parent element nodes and faces. If such constraints are used to attach parent elements to other Lagrangian bodies or they are used to drive the motion of a part, care must be exercised when the parent element faces involved in such constraints convert to particles. The constraint may be nullified upon parent element conversion and, consequently, the connection to other parts (in the case of tie constraints) or to the driving feature (in the case of coupling constraints) would no longer be realized. Hence, in certain cases you may need to place these constraints far enough from the parent elements that can convert for the constraints to be active throughout the analysis.

Element sets that are marked for possible conversion to particles but that are also part of the rigid body definition will never convert because the rigid body constraint is always enforced on the parent elements.

Interactions

Bodies modeled with elements that may convert to particles can interact with other finite element–meshed or analytical bodies via contact. Upon conversion the internally generated particles may also interact via contact with these bodies but only via the general contact functionality.

By default, if general contact interactions are included in your model, contact inclusions and exclusions involving internal node-based surfaces associated with the internal particles are generated. User-specified contact inclusions and exclusions referencing element-based surfaces that include convertible elements will also be reflected in internally generated requests. Table 15.2.2–4 and Table 15.2.2–5 show all correspondences. The naming convention used for the internally generated surfaces is explained in "Automatically generated sets and surfaces" above.

Table 15.2.2–4 Internally generated contact inclusions.

User-defined contact inclusion	Internally generated contact inclusions
*CONTACT INCLUSIONS, ALL EXTERIOR	*blank*, *AllUserElsets*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	None
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

Table 15.2.2–5 Internally generated contact exclusions.

User-defined contact exclusion	Internally generated contact exclusions
Always, regardless of user definitions	*UserElemBased*_PARENT_EE_SPH, *UserElemBased*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	None
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

As shown in the second row of Table 15.2.2–5, contact between the generated particles and the faces of the associated parent elements is always excluded from the general contact domain. The activated internal particles will interact with the neighboring yet inactive particles still attached to parent elements with exposed faces via the SPH formalism.

The contact interaction for the generated particles is the same as any contact interaction between a node-based surface (associated with the internal particles) and an element-based or analytical surface. All interaction types and formulations available for contact involving a node-based surface are allowed, including cohesive behavior. Different contact properties can be assigned via the usual options. The contact control and property assignment options used for pairs of surfaces that involve parent elements that can convert to particles will be reflected in internally generated assignments for the internal particle-based surfaces. Table 15.2.2–6 shows the internally generated assignments associated with user-defined requests.

Table 15.2.2–6 Internally generated contact control and property assignments.

User-defined contact inclusion	Internally generated contact inclusions
blank, *blank*	*blank*, *AllUserElsets*_SECT_NE_SPH
blank, *UserElemBased*	*blank*, *UserElemBased*_NS_SPH
UserElemBased,	*UserElemBased*, *UserElemBased*_NS_SPH
UserElemBased1, *UserElemBased2*	*UserElemBased1*, *UserElemBased2*_NS_SPH and *UserElemBased2*, *UserElemBased1*_NS_SPH

The generated particles may have different contact thicknesses since they are computed automatically at the beginning of the analysis. If one or two particles per isoparametric direction are requested to be generated upon conversion, all generated particles will have a contact thickness such that they are barely touching the closest face of the parent element. If three or more particles per direction are requested, some of the particles will not be touching the faces of the parent element. For these particles, the contact thickness will be the minimum thickness of all of the particles that are touching the parent element faces on that parent element.

You can specify the contact thickness of the generated particles by using the surface property assignment option for an element-based surface that includes the faces of the parent elements. This modeling choice affects contact interactions on parent elements before they convert.

Output

Output requests associated with parent elements, nodes of parent elements, or contact involving faces of parent elements trigger the creation of output requests associated with the corresponding internally generated particles. For example, if you request element output for an element set that contains parent elements, Abaqus/Explicit automatically creates an additional element output request using the corresponding internal element set containing generated particles, as described in "Automatically generated sets and surfaces."

A field output request for the STATUS output variable is created automatically for all parent elements and generated particles. The value of the STATUS output variable is toggled automatically between a value of zero and one upon conversion for both parent and generated particles. By default, only the active elements are displayed in the Visualization module. In addition, contour and vector plots are displayed appropriately on the elements that are currently active.

History output requests are also replicated for the generated particles. Since the actual element or node numbers of generated particles are defined internally, you can query the actual number of a particle in the Visualization module before identifying which output curve to display. For example, assume that you requested equivalent plastic strain history output for a small element set containing three C3D8R parent elements and that you requested that two particles per isoparametric direction (eight particles per parent element) are to be generated upon conversion. Before conversion you would have 3 curves to analyze; but after the three elements are converted, there are 24 curves from which to choose. You can query the element number of a particle and then select that curve from the 24 available history curves. Before conversion the curves associated with the particles have a value of zero. Upon conversion there will be a jump to the equivalent plastic strain value at the current time.

Limitations

Analyses involving finite element conversion to SPH particles are subject to the following limitations:

- Only reduced-integration continuum elements C3D8R, C3D6, and C3D4 are available for conversion.
- Surface loads specified on the faces of parent elements that convert during the analysis are not applied after conversion to particles. However, distributed loads, such as pressure, can be applied to other finite element surfaces that do not convert (e.g., on a piston surface) that can apply a pressure onto the particle elements (e.g., the fluid pushed by the piston) via contact interactions.
- Bodies modeled with elements that may convert to particles that were not defined using the same section definition will not interact with each other between the converted portions of the bodies. For example, body A and body B allow elements to convert to particles, but these elements are associated with different section definitions. After conversion, the particles will not interact.

- Within a given body (part) defined via one solid section definition, gravity loads and mass scaling cannot be specified selectively on a subset of elements referenced by this definition. Instead, the two features must be applied to all the elements in the element set associated with the solid section definition.
- Progressive conversion of finite elements into SPH particles during an analysis (based on strain, stress, or user-defined criterion) should be used only in applications that are inertia dominated and for which at any point during the analysis the strain energy is a small percentage of the total energy in the system. Specifically, progressive conversion should be used only in applications involving severe deformations, such as hypervelocity impact, blast, and crushing.

Input file template

The following example illustrates a smoothed particle hydrodynamic analysis of a bottle filled with fluid being dropped on the floor using the conversion technique. The plastic bottle and the floor are modeled with conventional shell elements. The fluid is modeled with C3D4 elements that will convert to two particles per isoparametric direction (four particles per element) at the beginning of the analysis based on a time-based criterion. Material property definitions are defined as usual for both the fluid and the bottle. Contact interaction is defined using the default options. Output is requested for stresses (pressure) and density in the fluid.

```
*HEADING
…
*ELEMENT, TYPE=C3D4, ELSET=Fluid_Inside_The_Bottle
…
*SOLID SECTION, ELSET=Fluid_Inside_The_Bottle, MATERIAL=Water,
CONTROLS=Time_Based_Conversion
*SECTION CONTROLS, ELEMENT CONVERSION=YES,
CONVERSION CRITERION=TIME, NAME=Time_Based_Conversion
First data line
Second data line
Third data line
2, 0.0
*MATERIAL, NAME=Water
Material definition for water, such as an EOS material
*ELEMENT, TYPE=S4R, ELSET=Plastic_Bottle
Element definitions for the shells
**
*INITIAL CONDITIONS, TYPE=VELOCITY
Data lines to define velocity initial conditions
**
*STEP
*DYNAMIC, EXPLICIT
*DLOAD
```

Data lines to define gravity load

```
**
*CONTACT
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=Fluid_Inside_The_Bottle
S, DENSITY
*END STEP
```

15.3 Particle generator

- “Particle generator,” Section 15.3.1

15.3.1 PARTICLE GENERATOR

Products: Abaqus/Explicit Abaqus/Viewer

References

- "Discrete element method," Section 15.1.1
- "Discrete particle elements," Section 33.1.1
- *PARTICLE GENERATOR
- *PARTICLE GENERATOR INLET
- *PARTICLE GENERATOR MIXTURE
- *PARTICLE GENERATOR FLOW
- *PROBABILITY DENSITY FUNCTION

Overview

The particle generator:

- is a tool for automatically introducing discrete particles (PD3D) into the problem domain during the course of the analysis;
- can be used during an initial step to prepare a discrete element method (DEM) model prior to the actual analysis; and
- can be used to continuously introduce discrete particles into the model while the analysis is in progress.

Introduction

The particle generator capability enables a modeling approach in which the discrete element method considers particles only after entering a region of interest. This approach can simplify model set-up and improve computational efficiency. Characteristics of this capability include:

- particles are generated at an inlet that can vary in shape over time,
- generated particles can have random sizes based on a user-specified probability density function,
- multiple species of particles can be generated simultaneously by a single generator, and
- the net mass flow rate and particle speeds at the inlet can be specified with time dependence.

Applications

How you use the particle generator will depend on the process being analyzed. Two broad usage categories are as follows:

- **Model creation:** In many applications the particle generator can be used to fill part of a domain with particles to establish initial conditions for subsequent studies. Figure 15.3.1–1 shows an

example where two particle generators are filling a drum mixer with two different batches of particles prior to performing a mixing simulation. The sequence of images show the start and end of the particle generation phase.

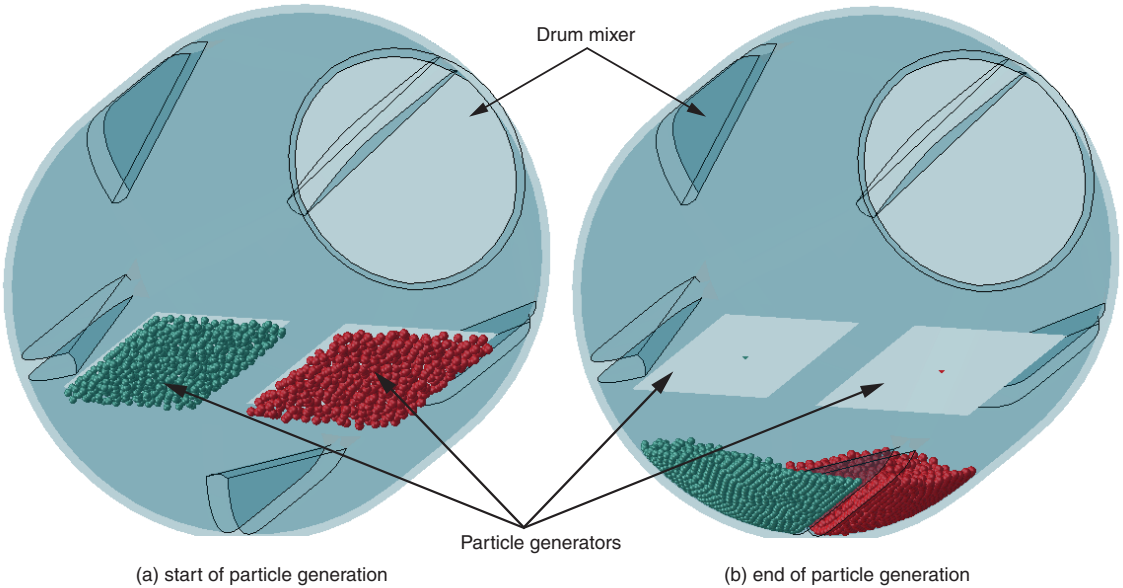


Figure 15.3.1–1 Model creation using particle generator.

- **Continuous generation:** In another type of application a particle generator can be used to inject a continuous stream of particles into the domain while an event of interest involving particles that have been introduced is underway. Figure 15.3.1–2 shows the use of a particle generator in a sorting analysis. In this application a particle generator is continuously feeding random sized particles to a screen sorter assembly while the analysis is in progress.

Defining a particle generator

To define a particle generator, you specify a name for the generator, the type of particle to be generated (only PD3D elements are supported), and the maximum number of particles that may be generated during the analysis. You must define an inlet surface, the list of particle species that will be generated, and the size distribution for each species.

In many processes the number and speed of the particles being introduced into the domain are not constant during the analysis. You must define the time-dependent mass flow rate per unit inlet area and the particle speed at the generator inlet to control the rate of injection of particles of each species through the inlet. These specifications allow the possibility of turning the particle generator on or off during the analysis.

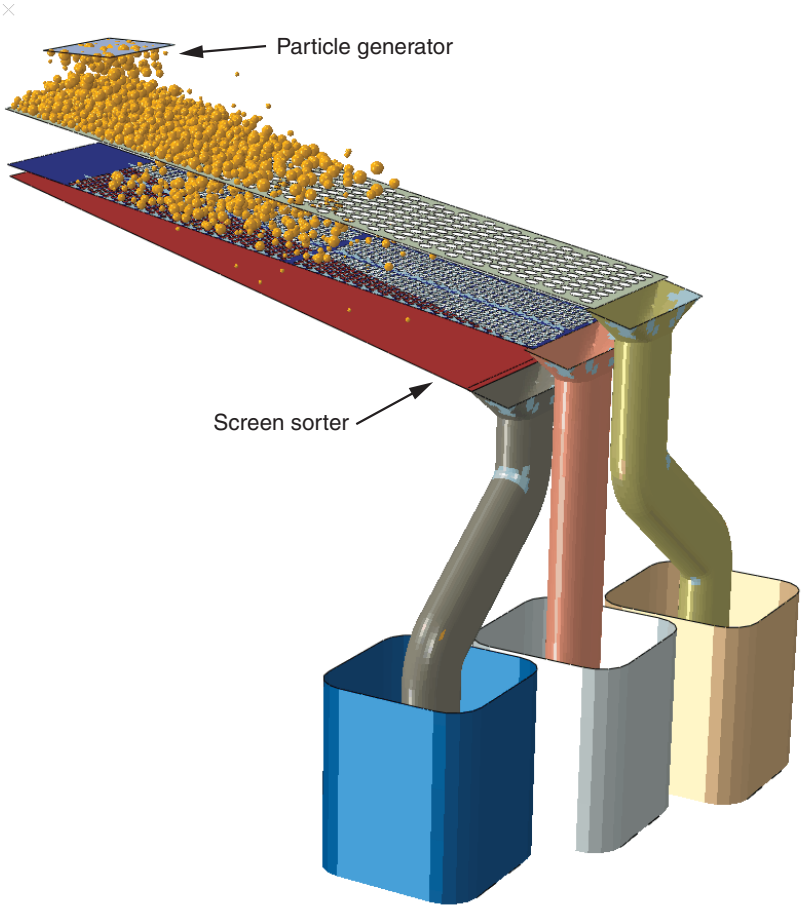


Figure 15.3.1–2 Continuous particle generation example.

Input File Usage: *PARTICLE GENERATOR, NAME=*generator_name*, TYPE=PD3D, MAXIMUM NUMBER OF PARTICLES=*number*

Inlet surface

The inlet surface can be thought of as an opening through which the particles are injected into the domain. It is important that no other model feature is positioned in front of the generator inlet to obstruct the flow of particles. An element-based surface must be defined for the inlet surface. Using surface elements is appropriate in most applications. Care should be taken with respect to the positive surface normal direction at the inlet because the initial particle velocity will be in this direction. A complex inlet

geometry as shown in Figure 15.3.1–3 may have several facets. In this example the normal direction to the inlet surface is out of the page.

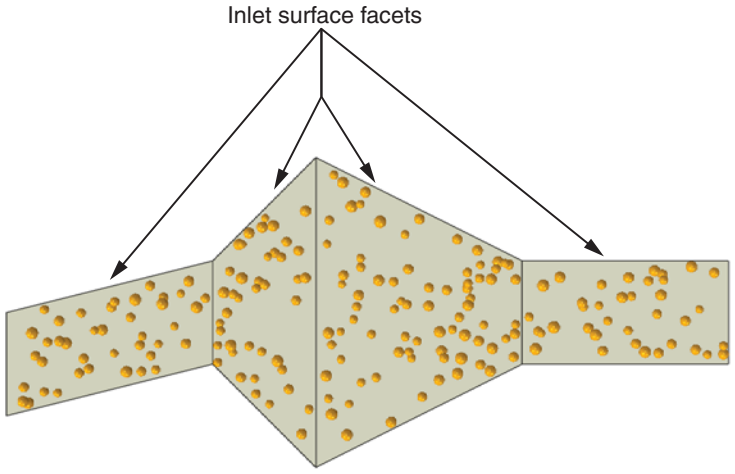


Figure 15.3.1–3 Complex inlet geometry with multiple facets.

The size and shape of the inlet surface can be modified during an analysis by adjusting the nodal positions of the inlet surface. The inlet surface can expand, collapse, and translate as well as undergo rigid body motions. Figure 15.3.1–4 shows the inlet surface of four different generators being subjected to different deformation and motion.

Input File Usage: ∗PARTICLE GENERATOR INLET, SURFACE=*inlet_surface_name*

Particle species

In many applications several different types of particles interact with each other. For example, sand, cement, and aggregate particles are mixed together to prepare a batch of concrete. A particle generator can be used to generate one or more species of particles as shown in Figure 15.3.1–5, where the two colors represent the two species of particles that have been simultaneously generated by a particle generator. An element set name is associated with each species in the mixture. This is analogous to a regular DEM analysis where the particles are grouped into element sets.

Input File Usage: ∗PARTICLE GENERATOR MIXTURE
element set or species 1, *element set for species 2*, *etc.*

Particle size distribution

Statistical distributions are commonly used to specify the particle sizes. Statistical distributions are defined by analytical or user-specified piecewise linear tabular probability density functions (PDF). The particle generator supports the following different types of probability density functions: uniform, normal, log-normal, and piecewise linear. The normal and log-normal PDFs have well-known analytical

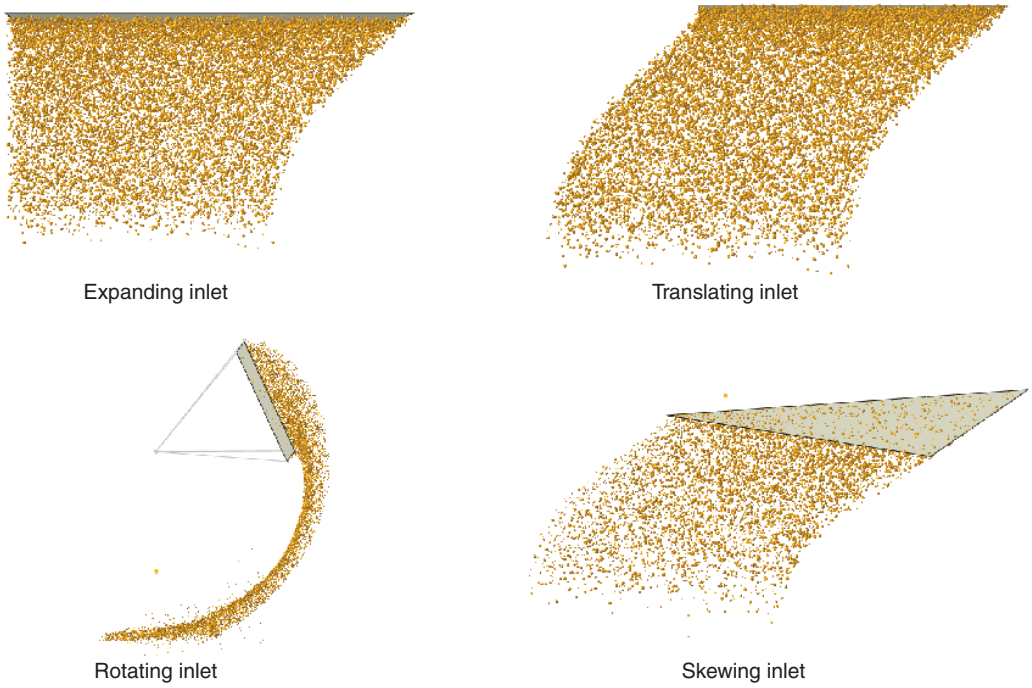


Figure 15.3.1–4 Expanding, translating, rotating, and skewing an inlet surface.

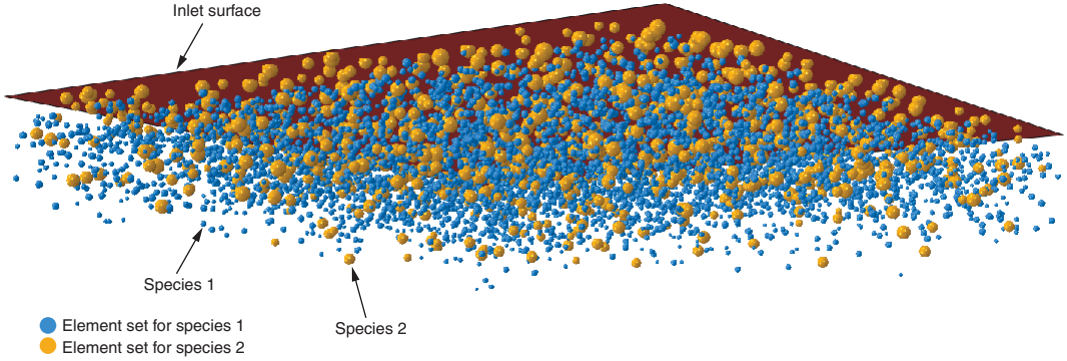


Figure 15.3.1–5 Generation of two species of particles.

forms that depend on the mean and standard deviation of the random deviate. In some cases the particle size distribution may be obtained from sieving particle samples in the field. These data are usually in the form of a table of size ranges and the percentages of particles that fall within those size ranges (a

histogram). The piecewise linear PDF can be used for such data. See "Probability density function," Section 2.12.1, for further details on the usage of probability density functions.

Mass flow rate

The rate of generation of particles is governed by the specified mass flow rate per unit area at the inlet, the area of the inlet, and the ability of the generator to insert a particle of a given size. The time varying mass flow rate per unit inlet area, $\dot{M}_s(t)$, has the units $\frac{mass}{time*length^2}$. For a multiple species particle generator the total mass flow rate per unit inlet area is the sum of the mass flow rates per unit inlet area of the individual species:

$$(\dot{M}_s(t) = \dot{M}_s^1(t) + \dot{M}_s^2(t) + ... + \dot{M}_s^n(t)).$$

The time varying mass flow rate per unit inlet area for each species must be provided in the form of an amplitude table ("Amplitude curves," Section 34.1.2).

Input File Usage: ∗PARTICLE GENERATOR FLOW, GENERATOR=*generator_name* , *mass flow rate amplitude for species 1*, *mass flow rate amplitude for species 2*, *etc.*

Particle speed

For a given particle generator, particles of all species are introduced into the domain through the inlet with a specified flow speed. The flow direction of the particles is along the normal to the inlet facet. Particles generated from different facets in the inlet surface will all have the same flow speed; however, depending on the facet normal direction, they may have different velocities. The time varying particle flow speed must be provided in the form of an amplitude table ("Amplitude curves," Section 34.1.2).

Input File Usage: ∗PARTICLE GENERATOR FLOW, GENERATOR=*generator_name*
flow speed amplitude curve,

Local coordinate system for inlet faces

The current coordinates of the nodes of each facet in the inlet surface are used to define a plane (a generation plane) where particles will be generated. Figure 15.3.1–6 shows the local directions of the generation plane. The first local direction, $\vec{t_1}$, is defined by the longest facet edge, while $\vec{t_3}$, the third local direction, is normal to the facet. The normal to the facet is obtained from the cross-product of the diagonal vectors (the vector joining nodes 2 and 4 and the vector joining nodes 1 and 3; see Figure 15.3.1–6 for local node numbers). The second in-plane direction is $\vec{t_2} = \vec{t_3} \times \vec{t_1}$. The plane of generation is positioned R_{min} in front of the inlet facet, where R_{min} is the smallest particle radius for the generator.The in-plane particle coordinates of a particle are generated randomly according to a uniform random distribution. The generated in-plane coordinates are acceptable if the particle can be placed at that location ensuring that it does not overlap with previously generated particles and also does not intersect the boundaries defined by the facet edges.

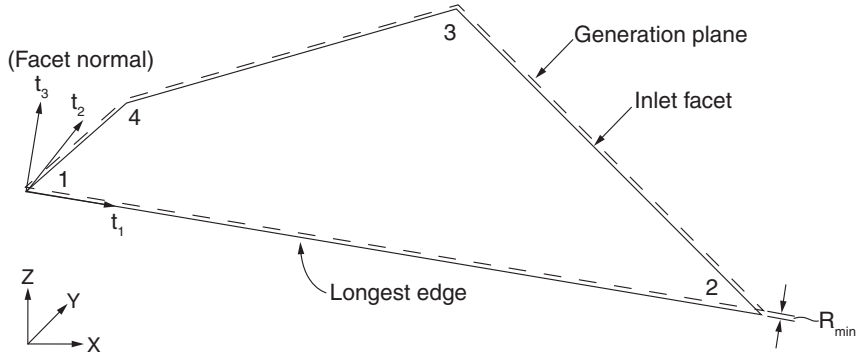


Figure 15.3.1–6 Particle generation plane.

Assigning properties to particles

Particles of different species will have different size distribution, density, and contact properties. To assign a density and size distribution for a particle species, the corresponding element set name is specified on a section definition for the PD3D elements. The element set name is also used to define a surface that is referred to under the contact specifications. This way of assigning particle properties and contact properties for generated particles is no different from what is done for user-defined sets of discrete particles. See "Discrete element method," Section 15.1.1, for details on how to assign properties to PD3D elements and how to set up contact definitions for a DEM analysis.

Particle size rejection

Abaqus/Explicit discards particle sizes that are outside the specified minimum and maximum range for the PDF. For normal and log-normal type PDFs this leads to a truncated distribution, as explained in "Probability density function," Section 2.12.1. Keep this in mind when comparing the generated particle size distribution to the target PDF.

A generated particle size is discarded if it is too large to fit through the current inlet geometry. It is possible that the generator may fail to place a particle in the randomly generated location in the generation plane if there is not sufficient room to accommodate the newly generated particle size. If this happens, the generator will generate another random location and attempt to place the particle there. By default, 15 such attempts will be made, after which the particle size is discarded and another particle size is randomly generated and the process is repeated. You can modify the default number of attempts to place a generated particle size.

If particle sizes are rejected because of either of the above reasons, the generated size distribution cannot be guaranteed to match the specified size distribution. Figure 15.3.1–7 shows a comparison of generated and specified piecewise linear particle size distributions for a specific problem where 14608 particles were generated at a constant mass flow rate and with constant particle speed through an inlet

of fixed geometry. The overall agreement of the generated particle size distribution is good even though the generator was unable to insert some particles into the domain.

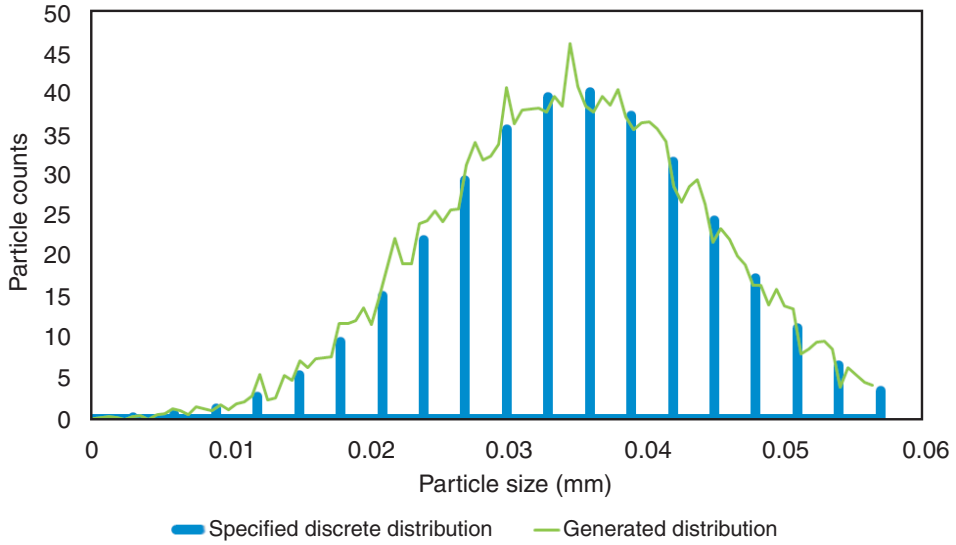


Figure 15.3.1–7 Comparison of specified and generated discrete particle size distributions.

Input File Usage: ∗PARTICLE GENERATOR, MAX INSERTION ATTEMPTS=*number*

Mass balance for continuous generation

At any increment, k, corresponding to the end of increment time, t_k, and time increment size, Δt_k, the specified mass of particles of species, i, to be generated is

$$m_{specified_k}{}^i = \dot{M}_s^i(t_k)A(t_k)\Delta t_k,$$

where $A(t_k)$ is the current inlet facet area. Due to the discrete nature of the time increment and particle sizes, the particle generator will not be able to match the specified mass exactly for the species for the given increment. The particle generator, therefore, maintains a mass balance for each species. For a given species a negative mass balance indicates that for that species the mass of particles generated was higher than what was required. The excess mass balance for species, i, from the previous increment $m_{balance_{k-1}}{}^i$ is taken into account while computing the target mass to be generated in the current increment:

$$m_{target_k}{}^i = m_{specified_k}{}^i + m_{balance_{k-1}}{}^i.$$

The mass balance for species i for the current increment is

$$m_{balance_k}{}^i = m_{target_k}{}^i - m_{actual}{}^i,$$

where $m_{actual}{}^i$ is the actual mass of the particles generated in increment k. The particle generator tries to match the specified mass flow rate per unit inlet area as closely as possible. If the target mass for a species is negative because of a high negative mass balance, no particles will be generated for that species until the target mass becomes positive again. Figure 15.3.1–8 shows a comparison of generated and specified mass flow rate for a specific problem where the specified mass flow rate per unit inlet area does not vary with time and the inlet geometry is fixed.

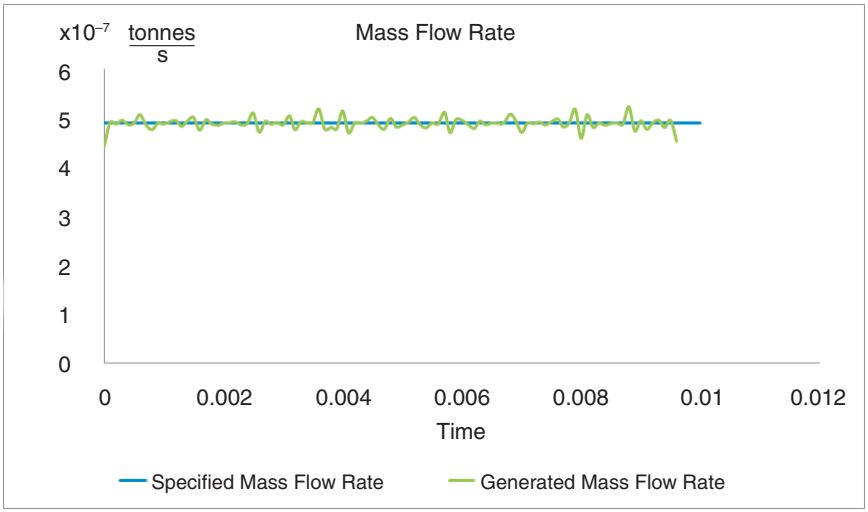


Figure 15.3.1–8 Comparison of specified and generated mass flow rate.

Inlet blocking

The flow speed, $Q(t)$, and mass flow rate per unit inlet area, $\dot{M}_s(t)$, are related as follows:

$$\dot{M}_s(t) = Q(t)\rho\epsilon(t),$$

where ρ is the particle density and $\epsilon(t)$ is the solid fraction. The solid fraction is the ratio of the volume of particles to the volume of the region they occupy. This relation suggests that the packing of particles ahead of the inlet increases for a given mass flow rate per unit inlet area as the flow speed is reduced. If this happens, the particle generator will take many more attempts to insert a particle. Therefore, a meaningful specification of flow speed for a given mass flow rate per unit inlet area is crucial for the efficiency of the particle generator. It is also important that no other model feature is positioned in front of the generator inlet to obstruct the flow of particles.

Automatic halting of particle generator

A particle generator will permanently halt after generating the maximum number of particles. Since it may be difficult to make an accurate estimation of the total number of particles that need to be generated during an analysis, overestimating the maximum number of particles will ensure that a particle generator will not run out of particles. However, grossly overestimating the maximum number of particles will lead to larger memory requirements for the analysis.

The particle generator will temporarily halt for an increment if the region near the inlet has a high volume fraction of particles. This helps avoid performance degradation due to an excessive number of failed attempts to insert a new particle when the inlet is becoming clogged with existing particles. The default threshold for a temporary halt to the generator is a particle volume fraction of 0.4 over a volume that extends $2.0R_{max}$ in front of the inlet surface, where R_{max} is the maximum particle radius for the generator. You can modify the default value of the solid fraction used for halting the generator.

It is possible that the current inlet geometry and the particle size distribution is such that the particle generator repeatedly exhausts the 15 attempts to insert a particle of a given size. If 10 such cycles of 15 attempts each have been unsuccessful in a given increment, the generator will temporarily halt generation. You can modify the default value of 10 cycles prior to halting particle generation.

Input File Usage: ∗PARTICLE GENERATOR, SOLID FRACTION=*threshold solid fraction*,
MAX ATTEMPTS PER INCREMENT=*number of insertion attempts cycles*

Initial conditions

Initial conditions should not be applied to nodes of particle species.

Boundary conditions

Boundary conditions can be applied to the nodes of the inlet facet to expand, shrink, or move the inlet facets of a generator.

Loads

The element set name for a species can be used to apply loads such as gravity or body loads.

Elements

The particle generator generates PD3D elements only.

Constraints

Constraints such as multi-point constraint type BEAM can be applied to the nodes of inlet facets to facilitate the application of rigid body motion to the inlet facet of a generator.

Interactions

Contact interactions between particle species are specified similar to user-defined PD3D elements.

Output

Output is available for the element sets and node sets generated by the particle generator.

Limitations

The following limitations apply to analyses involving the particle generator:

- Multi-domain analysis is not allowed.
- Only PD3D elements can be generated.
- It is not possible to generate clusters of PD3D elements.
- Although supported in Abaqus/Viewer, the functionality is not supported in Abaqus/CAE.

Input file template

The following example illustrates a particle generator with two particle species:

```
*HEADING
…
*PARTICLE GENERATOR, NAME=generator_name, TYPE=PD3D,
MAXIMUM NUMBER OF PARTICLES=number, SOLID FRACTION=number,
MAX INSERTION ATTEMPTS=number, MAX ATTEMPTS PER INCREMENT=number
**
*PARTICLE GENERATOR INLET, SURFACE=inlet_surf
**
*PARTICLE GENERATOR MIXTURE
gen_SET1, gen_SET2
**
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET1, TYPE=NORMAL
Data line to define PDF
*PROBABILITY DENSITY FUNCTION, NAME=PDF_gen_SET2, TYPE=LOGNORMAL
Data line to define PDF
**
*DISCRETE SECTION, ELSET=gen_SET1
PDF_gen_SET1
*DISCRETE SECTION, ELSET=gen_SET2
PDF_gen_SET2
…
*SURFACE, TYPE=ELEMENT, NAME=inlet_surf
element set name
*SURFACE, TYPE=ELEMENT, NAME=PG_SURF1
gen_SET1
*SURFACE, TYPE=ELEMENT, NAME=PG_SURF2
```

gen_SET2
**
SURFACE INTERACTION,NAME=prop_gen_SET1*
SURFACE INTERACTION,NAME=prop_gen_SET2*
SURFACE INTERACTION,NAME=prop_gen_SET1_and_gen_SET2*
**
AMPLITUDE, name=flow_speed*
AMPLITUDE, name=mass_flow_rate_gen_SET1*
AMPLITUDE, name=mass_flow_rate_gen_SET2*
**
***STEP*
***DYNAMIC, EXPLICIT*
**
PARTICLE GENERATOR FLOW, GENERATOR=generator_name*
flow_speed, mass_flow_rate_gen_SET1, mass_flow_rate_gen_SET2
**
***DLOAD*
Data lines to define gravity load for gen_SET1
Data lines to define gravity load for gen_SET2
**
***CONTACT*
***CONTACT INCLUSIONS*
PG_SURF1, PG_SURF1
PG_SURF2, PG_SURF2
PG_SURF1, PG_SURF2
***CONTACT PROPERTY ASSIGNMENT*
PG_SURF1, PG_SURF1, prop_gen_SET1
PG_SURF2, PG_SURF2, prop_gen_SET2
PG_SURF1, PG_SURF2, prop_gen_SET1_and_gen_SET2
**
***END STEP*

16. Sequentially Coupled Multiphysics Analyses

Sequentially coupled multiphysics analyses 16.1

16.1 Sequentially coupled multiphysics analyses

- "Predefined fields for sequential coupling," Section 16.1.1
- "Sequentially coupled thermal-stress analysis," Section 16.1.2
- "Predefined loads for sequential coupling," Section 16.1.3

16.1.1 PREDEFINED FIELDS FOR SEQUENTIAL COUPLING

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Sequentially coupled thermal-stress analysis," Section 16.1.2
- "Predefined fields," Section 34.6.1
- "Creating and modifying output requests," Section 14.4.5 of the Abaqus/CAE User's Guide
- "Defining a temperature field," Section 16.11.9 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

The time history of the following nodal output quantities, generated in an Abaqus/Standard analysis, can be read into subsequent Abaqus/Standard analyses as predefined fields for sequentially coupled multiphysics workflows:

- Temperature
- Normalized concentration
- Electric potential

A sequentially coupled multiphysics analysis can be used when the coupling between one or more of the physical fields in a model is only important in one direction—a special common case is a sequential thermal-stress analysis ("Sequentially coupled thermal-stress analysis," Section 16.1.2). While the uncoupled thermal-stress analysis is the most common sequential multiphysics workflow, the predefined field capability in Abaqus/Standard directly supports similar sequential workflows involving normalized concentrations ("Mass diffusion analysis," Section 6.9.1) and electric potentials ("Coupled thermal-electrical analysis," Section 6.7.3). As with temperatures, normalized concentrations and electric potentials can be read from the output database (**.odb**) file into subsequent analyses as predefined fields.

When defined by results from a previous analysis, predefined fields typically vary with position and are time dependent—they are predefined because they are not changed by the current analysis. When predefined fields are read from a previous analysis, they are read in at the nodes. They are then interpolated within elements as needed (see "Interpolating data between meshes" in "Predefined fields," Section 34.6.1). Any number of predefined fields can be read in, and material properties can be defined to depend on them. In addition, volumetric strain will arise in a stress analysis if thermal expansion ("Thermal expansion," Section 26.1.2) or field expansion ("Field expansion," Section 26.1.3) are included in the material property definition.

Predefined fields may affect the system response through:

- the constitutive behavior, such as the yield stress defined as a function of temperature or field variables; or
- volumetric strains when thermal or field expansion behaviors ("Thermal expansion," Section 26.1.2, and "Field expansion," Section 26.1.3) are included in the material definition in a stress/displacement analysis.

Saving temperatures, normalized concentrations, and electric potentials for predefined fields in subsequent analyses

Nodal temperatures, normalized concentrations, and electrical potentials can be stored as functions of time for use in subsequent analyses. Temperatures can be stored in either the results (`.fil`) file or the output database (`.odb`) file, but normalized concentrations and electrical potentials can be used only if they are stored in the output database file. Saved values must be read into the new analyses as predefined fields. See "Node output" in "Output to the data and results files," Section 4.1.2, and "Node output" in "Output to the output database," Section 4.1.3.

Saving temperatures for predefined fields in subsequent analyses

To be read as a predefined field, nodal temperatures must be stored as functions of time in the results (`.fil`) file or output database (`.odb`) file. You can request nodal temperature output (NT) in an uncoupled heat transfer analysis or in a coupled thermal-electrical analysis.

Saving normalized concentrations for predefined fields in subsequent analyses

To be read as predefined fields, normalized concentrations must be stored as functions of time in the output database (`.odb`) file—unlike nodal temperatures they cannot be read directly from a results file. You can request nodal normalized concentrations output (NNC) in a mass diffusion analysis.

Saving electric potentials for predefined fields in subsequent analyses

To be read as predefined fields, electrical potentials must be stored as functions of time in the output database (`.odb`) file—unlike nodal temperatures they cannot be read directly from a results file. You can request nodal electric potential output (EPOT) in a coupled thermal-electrical analysis or a piezoelectric analysis.

Transferring temperatures as temperature fields

To define the temperature field at different times in the current analysis, you read the nodal temperatures stored as a function of time in the heat transfer results or output database file. Nodes can be removed for the current problem; for example, in a sequential thermal-stress analysis elements that represent nonstructural parts of the heat transfer mesh (such as insulation or cooling fluid) can be omitted in the stress analysis. When the heat transfer results file or output database file is read, temperatures at nodes that are not present in the mesh for the current analysis are ignored.

You must specify the name of the thermal analysis results file or output database file that contains the required nodal temperatures. The file extension is optional. If the heat transfer model and the current analysis model share the same mesh, the default is the results file. If the heat transfer model and the current analysis model have dissimilar meshes, the output database file must be used. See "Reading the values of a field from a user-specified file" in "Predefined fields," Section 34.6.1, for more information.

If both models contain part and assembly definitions, the part (`.prt`) files from both analyses are required to transfer temperatures from the thermal analysis to the current analysis. If the thermal model is defined in terms of an assembly of part instances, the current analysis must be as well. The part instance names and local node numbers must be the same in both analyses for the nodes at which temperatures are transferred.

Transferring temperatures, normalized concentrations, and electric potentials from the output database to predefined fields

To define predefined fields at different times in the current analysis, you can read nodal temperatures, normalized concentrations, or electric potentials stored as a function of time in the output database file. Nodes can be removed for the current problem. When the nodal output variables on the output database file are on nodes that are not present in the mesh for the current analysis, they are ignored.

You must specify the name of the output database file that contains the required nodal output variables as well as the nodal output label (NT, NNC, or EPOT) to identify the field that is being read. See "Defining fields using nodal scalar output values from a user-specified output database file" in "Predefined fields," Section 34.6.1.

If both models contain part and assembly definitions, the part (`.prt`) files from both analyses are required to transfer nodal results from the original analysis to the current analysis. If the original model is defined in terms of an assembly of part instances, the current analysis must be as well. The part instance names and local node numbers must be the same in both analyses for the nodes at which nodal results are transferred.

Initial conditions

Appropriate initial conditions for Abaqus/Standard procedures are discussed in Chapter 6, "Analysis Procedures." You can read the nodal temperatures, normalized concentrations, or electric potentials from previous analyses to initialize predefined fields. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, for details.

Boundary conditions

Appropriate boundary conditions for Abaqus/Standard procedures are discussed in Chapter 6, "Analysis Procedures." See also "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Loads

Appropriate loadings for Abaqus/Standard procedures are discussed in Chapter 6, "Analysis Procedures." See also "Applying loads: overview," Section 34.4.1.

Predefined fields

See "Predefined fields," Section 34.6.1, for additional details on predefined temperatures and fields.

Material options

See Part V, "Materials," for details on the material models available in Abaqus/Standard.

Volumetric strain will arise in a stress analysis if thermal expansion ("Thermal expansion," Section 26.1.2) or field expansion ("Field expansion," Section 26.1.3) is included in the material property definition.

Elements

Continuum and structural elements available in Abaqus/Standard are discussed in Chapter 28, "Continuum Elements," and Chapter 29, "Structural Elements." Details on how results from a previous analysis can be transferred to a current analysis are discussed in "Predefined fields," Section 34.6.1.

Output

Appropriate output variables for Abaqus/Standard are described in Part V, "Materials." All of the output variables are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

A moisture-stress analysis is an example of a sequentially coupled multiphysics analysis. A typical sequentially coupled moisture-stress analysis consists of two Abaqus/Standard runs: a mass diffusion analysis and a subsequent stress analysis. Normalized concentrations are stored in the output database file for the mass diffusion analysis and read into the subsequent stress analysis as a predefined field.

The following template shows the input for the mass diffusion analysis **massdiffusion.inp**:

```
*HEADING
...
*ELEMENT, TYPE=DC2D4
(Choose the mass diffusion element type)
...
*STEP
*MASS DIFFUSION
...
Apply loads and boundary conditions
...
** Write all normalized concentrations to the output
** database file, massdiffusion.odb
*OUTPUT, FIELD
*NODE OUTPUT, NSET=NALL
```

```
 NNC
*END STEP
```

The following template shows the input for the subsequent static structural analysis:

```
*HEADING
…
*ELEMENT, TYPE=CPE4R
```
(Choose the continuum element type compatible with the mass diffusion element type used)

```
*MATERIAL
*EXPANSION, FIELD=1
```
(Define field expansion for field 1 so that the normalized concentration causes volumetric strain in the stress analysis)

```
…
*STEP
*STATIC
…
```
Apply structural loads and boundary conditions

```
…
*FIELD, FILE=massdiffusion.odb, OUTPUT VARIABLE=NNC, FIELD=1
```
Read in all normalized concentrations from the output database file into field variable 1

```
…
*END STEP
```

16.1.2 SEQUENTIALLY COUPLED THERMAL-STRESS ANALYSIS

Products: Abaqus/Standard Abaqus/CAE

References

- "Defining an analysis," Section 6.1.2
- "Heat transfer analysis procedures: overview," Section 6.5.1
- "Predefined fields for sequential coupling," Section 16.1.1
- "Creating and modifying output requests," Section 14.4.5 of the Abaqus/CAE User's Guide
- "Defining a temperature field," Section 16.11.9 of the Abaqus/CAE User's Guide, in the HTML version of this guide

Overview

A sequentially coupled heat transfer analysis:

- is used when the stress/deformation field in a structure depends on the temperature field in that structure, but the temperature field can be found without knowledge of the stress/deformation response; and
- is usually performed by first conducting an uncoupled heat transfer analysis and then a stress/deformation analysis.

A thermal-stress analysis in which the temperature field does not depend on the stress field is a common example of a sequential multiphysics workflow and is one case of the more general workflow described in "Predefined fields for sequential coupling," Section 16.1.1. In such thermal-stress analyses, temperature is calculated in an uncoupled heat transfer analysis ("Uncoupled heat transfer analysis," Section 6.5.2) or in a coupled thermal-electrical analysis ("Coupled thermal-electrical analysis," Section 6.7.3).

Saving the nodal temperatures

Nodal temperatures are stored as a function of time in the heat transfer results (`.fil`) file or output database (`.odb`) file by requesting output variable NT as nodal output to the results or output database file. See "Node output" in "Output to the data and results files," Section 4.1.2, and "Node output" in "Output to the output database," Section 4.1.3.

Transferring the heat transfer results to the stress analysis

The temperatures are read into the stress analysis as a predefined field; the temperature varies with position and is usually time dependent. It is predefined because it is not changed by the stress analysis solution. Such predefined fields are always read into Abaqus/Standard at the nodes. They are then interpolated to the calculation points within elements as needed (see "Interpolating data between meshes" in "Predefined fields," Section 34.6.1). The temperature interpolation in the stress elements is usually

approximate and one order lower than the displacement interpolation to obtain a compatible variation of thermal and mechanical strain. Any number of predefined fields can be read in, and material properties can be defined to depend on them.

For more information, see "Transferring temperatures as temperature fields" in "Predefined fields for sequential coupling," Section 16.1.1.

Initial conditions

Appropriate initial conditions for the thermal and stress analysis problems are described in the heat transfer and stress analysis sections—for example, see "Heat transfer analysis procedures: overview," Section 6.5.1; "Coupled thermal-electrical analysis," Section 6.7.3; "Static stress analysis procedures: overview," Section 6.2.1; and "Dynamic analysis procedures: overview," Section 6.3.1. See also "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1.

Boundary conditions

Appropriate boundary conditions for the thermal and stress analysis problems are described in the heat transfer and stress analysis sections—for example, see "Heat transfer analysis procedures: overview," Section 6.5.1; "Coupled thermal-electrical analysis," Section 6.7.3; "Static stress analysis procedures: overview," Section 6.2.1; and "Dynamic analysis procedures: overview," Section 6.3.1. See also "Boundary conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.3.1.

Loads

Appropriate loading for the thermal and stress analysis problems is described in the heat transfer and stress analysis sections—for example, see "Heat transfer analysis procedures: overview," Section 6.5.1; "Coupled thermal-electrical analysis," Section 6.7.3; "Static stress analysis procedures: overview," Section 6.2.1; and "Dynamic analysis procedures: overview," Section 6.3.1. See also "Applying loads: overview," Section 34.4.1.

Predefined fields

In addition to the temperatures read in from the heat transfer analysis, user-defined field variables can be specified; these values only affect field-variable-dependent material properties, if any. See "Predefined fields," Section 34.6.1.

Material options

The materials in the thermal analysis must have thermal properties such as conductivity defined (see "Thermal properties: overview," Section 26.2.1). Any mechanical properties such as elasticity will be ignored in the thermal analysis, but they must be defined for the stress analysis procedure. See Part V, "Materials," for details on the material models available in Abaqus/Standard.

Thermal strain will arise in the stress analysis if thermal expansion ("Thermal expansion," Section 26.1.2) is included in the material property definition.

Elements

Any of the heat transfer elements in Abaqus/Standard can be used in the thermal analysis. In the stress analysis the corresponding continuum or structural elements must be chosen. For example, if heat transfer shell element type DS4 is defined by nodes 100, 101, 102, and 103 in the heat transfer analysis, three-dimensional shell element type S4R or S4R5 must be defined by these nodes in the stress analysis procedure. For continuum elements heat transfer results from a mesh using first-order elements can be transferred to a stress analysis with a mesh using second-order elements (see "Using second-order stress elements with first-order heat transfer elements (the midside node capability)" in "Predefined fields," Section 34.6.1).

Output

The nodal temperatures must be written to the heat transfer analysis results or output database file by requesting the output variable NT (see "Output to the data and results files," Section 4.1.2). These temperatures will be read into the stress analysis procedure.

Appropriate output variables are described in the heat transfer and stress analysis sections. All of the output variables are outlined in "Abaqus/Standard output variable identifiers," Section 4.2.1.

Input file template

A typical sequentially coupled thermal-stress analysis consists of two Abaqus/Standard runs: a heat transfer analysis and a subsequent stress analysis.

The following template shows the input for the heat transfer analysis **heat.inp**:

```
*HEADING
…
*ELEMENT, TYPE=DC2D4
(Choose the heat transfer element type)
…
*STEP
*HEAT TRANSFER
…
Apply thermal loads and boundary conditions
…
** Write all nodal temperatures to the results or
** output database file, heat.fil/heat.odb
*NODE FILE, NSET=NALL
 NT
*OUTPUT, FIELD
*NODE OUTPUT, NSET=NALL
 NT
*END STEP
```

The following template shows the input for the subsequent static structural analysis:

```
*HEADING
…
*ELEMENT, TYPE=CPE4R
(Choose the continuum element type compatible with the heat transfer element type used)
…
*STEP
*STATIC
…
Apply structural loads and boundary conditions
…
*TEMPERATURE, FILE=heat
Read in all nodal temperatures from the results or output database file, heat.fil/heat.odb
…
*END STEP
```

16.1.3 PREDEFINED LOADS FOR SEQUENTIAL COUPLING

Product: Abaqus/Standard

References

- "Mapping thermal and magnetic loads," Section 3.2.27
- "Defining an analysis," Section 6.1.2
- "Eddy current analysis," Section 6.7.5
- "Concentrated loads," Section 34.4.2

Overview

The values of the following whole element output quantities, generated in an Abaqus/Standard time-harmonic eddy current analysis, can be read into subsequent Abaqus/Standard analyses as point loads for sequentially coupled multiphysics workflows:

- Rate of Joule heat dissipation
- Magnetic body force intensity

A sequentially coupled multiphysics analysis can be used to apply electromagnetically generated loads (from a time-harmonic eddy current analysis) in a heat transfer, coupled temperature-displacement, or stress/displacement analysis. In many cases coupling is important only from the time-harmonic eddy current analysis; the impact of loading on the structure's mechanical or thermal response is not great enough to affect the validity of the original time-harmonic eddy current analysis.

Saving Joule heat dissipation or magnetic body force intensity for use in subsequent analyses

You can request Joule heat dissipation output (EMJH) or magnetic body force intensity output (EMBF) in a time-harmonic eddy current analysis. Only values stored in the output database (`.odb`) file are available for use with sequential coupling.

Converting results for subsequent use

The whole element quantities are converted to nodal load quantities using the **abaqus emloads** utility. The utility converts Joule heat dissipation output to concentrated heat flux and magnetic body force intensity output to point loads. This utility also enables conversion of results between dissimilar meshes. For more information, see "Mapping thermal and magnetic loads," Section 3.2.27.

Conversion limitations

When converting results values between dissimilar meshes, global conservation of the net flux is ensured provided that the model domain in the heat transfer, coupled temperature-displacement, or stress/displacement analysis matches the model domain in the time-harmonic eddy current analysis.

The conservative mapping algorithm used in the **abaqus emloads** utility also provides a locally smooth distribution of point flux values (either body force or concentrated heat flux) in cases where the mesh in the time-harmonic eddy current analysis is finer than the "target" representative mesh. In situations where this is not the case and the "target" representative mesh is finer or of similar size to the mesh in the time-harmonic eddy current analysis, you may observe nodal locations with zero converted flux values. In these cases you will still observe global conservation of the flux, but your solution may be adversely affected locally. You can correct for these situations by always performing the time-harmonic eddy current analysis with a finer mesh.

Transferring nodal loads from the output database to concentrated loads

To define loads in a heat transfer, coupled temperature-displacement, or stress/displacement analysis, you can read nodal concentrated heat fluxes and point loads from the output database (**.odb**) file created by the **abaqus emloads** utility.

Input file template

In this example heat flux values are stored in the output database from a time-harmonic eddy current analysis. These values, after conversion to point heat fluxes, are read into a subsequent analysis as a concentrated flux.

The following template shows the input for the time-harmonic eddy current analysis **electromagnetic.inp**:

```
*HEADING
…
*ELEMENT, TYPE=EMC3D8
(Choose the electromagnetic element type)
…
*STEP
*ELECTROMAGNETIC, LOW FREQUENCY, TIME HARMONIC
…
Apply loads and boundary conditions
…
** Write element Joule heat dissipation results to the output
** database file, electromagnetic.odb
*OUTPUT, FIELD
*ELEMENT OUTPUT, ELSET=CONDUCTOR
 EMJH
*END STEP
```

The following template shows the input for the heat transfer analysis, **heattransfer.inp**, which refers to an output database, **pointflux.odb**, created using the **abaqus emloads** utility, and which has mapped quantities from the results of the time-harmonic eddy current analysis, stored in **electromagnetic.odb**:

```
*HEADING
…
*ELEMENT, TYPE=DC3D8
(Choose the heat transfer continuum element type)
…
*STEP
*HEAT TRANSFER, STEADY STATE
…
Apply heat transfer loads and boundary conditions
…
*CFLUX, FILE=pointflux.odb
Read in all nodal heat flux values from the output database and apply as concentrated nodal fluxes
…
*END STEP
```

17. Co-simulation

Co-simulation 17.1
Preparing an Abaqus analysis for co-simulation 17.2
Co-simulation between Abaqus solvers 17.3
Co-simulation using Abaqus and discrete models 17.4

17.1 Co-simulation

- “Co-simulation: overview,” Section 17.1.1

17.1.1 CO-SIMULATION: OVERVIEW

The co-simulation technique is a capability for run-time coupling of Abaqus and other analysis programs. An Abaqus analysis can be coupled to another Abaqus analysis or to a third-party analysis program to perform multiphysics simulations and multidomain (multimodel) coupling.

Abaqus provides built-in procedures to solve multiphysics simulations as described in "Multiphysics analyses" in "Solving analysis problems: overview," Section 6.1.1. For multiphysics problems for which Abaqus does not provide a built-in solution procedure or where the solution procedure is limited in functionality, you can use the co-simulation technique to couple Abaqus with a third-party analysis program; for example, fluid-structure interaction (FSI) simulation in conjunction with computational fluid dynamics (CFD) analysis programs.

Co-simulation between Abaqus/Standard and Abaqus/Explicit illustrates a multiple domain analysis approach, where each Abaqus analysis operates on a complementary section of the model domain where it is expected to provide the more computationally efficient solution. For example, Abaqus/Standard provides a more efficient solution for light and stiff components, while Abaqus/Explicit is more efficient for solving complex contact interactions.

Features of the Abaqus co-simulation technique

The Abaqus co-simulation technique:

- can be used to solve complex fluid-structure interactions by coupling Abaqus with CFD analysis programs, including Abaqus/CFD transient analysis (co-simulation with the Abaqus/CFD steady-state solver is not supported);
- can be used to solve conjugate heat transfer problems by coupling Abaqus/Standard with CFD analysis programs, including Abaqus/CFD transient analyses;
- can be used to solve problems involving electromagnetic-thermal or electromagnetic-mechanical interactions by coupling Abaqus with an electromagnetic analysis program, including electromagnetic analysis procedures in Abaqus/Standard;
- can be used for multiphysics simulations by coupling Abaqus with third-party analysis programs;
- can be used to solve complex multidomain analyses more effectively by coupling Abaqus/Standard to Abaqus/Explicit;
- can be used to solve structural-to-logical simulations by coupling Abaqus/Standard or Abaqus/Explicit with Dymola;
- can be used to couple Abaqus with in-house codes using the SIMULIA Co-Simulation Engine;
- is intended for advanced users with in-depth knowledge of Abaqus and the third-party analysis program;
- allows for both unidirectional and bidirectional transfer of data;
- can be used with Abaqus models having linear or nonlinear structural response; and
- supports steady-state, transient, and, for electromagnetic procedures, time-harmonic simulations.

Interaction between domains modeled with different analysis programs

In a co-simulation the interaction between the domains is through a common physical interface region over which data are exchanged in a synchronized manner between Abaqus and the coupled analysis program.

One domain may affect the response of another domain through one or more of the following:

- the constitutive behavior, such as the yield stress defined as a function of temperature or stress defined as a function of other solution fields, such as thermal strains or the piezoelectric effect;
- surface tractions/fluxes, such as a fluid exerting pressure on a structure;
- body forces/fluxes, such as heat generation due to flow of current in a coupled thermal-electrical simulation;
- contact forces, such as the forces due to contact between a vehicle and an occupant/pedestrian modeled as separate domains;
- kinematics, such as fluid in contact with a compliant structure where the interface motion affects the fluid flow; and
- discrete coupling, such as sensor and actuation information.

Coupling Abaqus using the SIMULIA Co-Simulation Engine

The SIMULIA Co-Simulation Engine provides coupling between Abaqus analyses or between Abaqus and third-party analysis programs. This coupling method is used for fluid-structure, conjugate heat transfer, electromagnetic-structural, electromagnetic-thermal, and structural-logical simulations, and when coupling Abaqus/Standard to Abaqus/Explicit for interaction between implicit dynamic and explicit dynamic domains.

Fluid-structure interaction

You can solve complex fluid-structure interaction (FSI) problems by coupling Abaqus/Standard or Abaqus/Explicit to a computational fluid dynamics (CFD) analysis program. Abaqus/Standard and Abaqus/Explicit solve the structural domain, and the CFD analysis program solves the fluid domain. Abaqus/Standard and Abaqus/Explicit can be coupled with Abaqus/CFD as well as with several third-party CFD analysis programs.

For detailed information on coupling Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, and "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2. For a complete list of qualified partner products, see the Co-simulation page at www.3ds.com/simulia.

Conjugate heat transfer

You can solve conjugate heat transfer problems involving fluids and structures by coupling Abaqus/Standard to a computational fluid dynamics (CFD) analysis program. Abaqus/Standard models heat transfer within the structure (see "Uncoupled heat transfer analysis," Section 6.5.2, and "Fully coupled thermal-stress analysis," Section 6.5.3), and the CFD analysis program solves the

energy equation for the fluid flow surrounding the structure. Abaqus/Standard can be coupled with Abaqus/CFD as well as with several third-party CFD analysis programs.

For an example of Abaqus/CFD to Abaqus/Standard co-simulation, refer to "Conjugate heat transfer analysis of a component-mounted electronic circuit board," Section 6.1.1 of the Abaqus Example Problems Guide. For detailed information on coupling Abaqus/CFD to Abaqus/Standard, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, and "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2. For a complete list of qualified partner products, see the Co-simulation page at www.3ds.com/simulia.

Electromagnetic-thermal or electromagnetic-mechanical coupling

Applications such as induction heating require interaction between electromagnetic and thermal fields. You can solve this class of problems by coupling two Abaqus/Standard analyses, where one analysis solves for the fields in the electromagnetic domain, while the other solves for the fields in the thermal domain. Abaqus/Standard can be coupled with itself, as well as with several third-party electromagnetic analysis programs.

For detailed information on coupling Abaqus/Standard to Abaqus/Standard, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, and "Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation," Section 17.3.3. For a complete list of qualified partner products, see the Co-simulation page at www.3ds.com/simulia.

System-level modeling via logical-physical interaction

System-level modeling refers to modeling of systems that may include both physical (structural, thermal, acoustics, etc.) and logical components. The distinction between the two modeling abstractions is as follows:

- Logical modeling refers to a large class of modeling abstractions often encountered in the engineering practice. Generally speaking, you can designate a part of a system as using a logical modeling abstraction when most (if not all) of the geometry of the part is removed. Examples include electronic control modules, electric motors, and pneumatic or hydraulic subsystems, which in many cases can be modeled from a functional perspective without attempting to model the flow of electrons, the variation of magnetic fluxes, or the air/fluid type of flow in ducts and pipes. Dymola offers a variety of logical modeling options.
- Physical modeling is the complementary modeling abstraction to logical modeling. Abaqus uses a physical modeling abstraction most of the time; as elements deform, they know precisely about their geometry, thus trying to mimic the real world at a fine-grain level.

In many engineering systems the interaction between logical and physical components is paramount, and you cannot fully analyze one without the other. Co-simulation using Abaqus and Dymola provides the capability to analyze this type of system.

Consider the example of a rolling mill: the incoming slab, which may not have a constant thickness, can be modeled in Abaqus as being deformed by the rolling cylinders. Because of the nonconstant incoming thickness, a pressure that adapts as a function of deformation needs to be exerted on the cylinders to compensate such that the exit thickness is as constant as possible. Abaqus sensors can

export the information about the mechanical status of the system to Dymola, which in turn could use this information to model the necessary compensators to calculate the needed actuation load at any given time. Abaqus can import the actuation load and apply it to the cylinders.

For detailed information on coupling Abaqus/Standard to Dymola, see "Structural-to-logical co-simulation," Section 17.4.1. For a complete list of qualified partner products, see the Co-simulation page at www.3ds.com/simulia.

Interaction between an implicit transient analysis and an explicit dynamics analysis

In certain cases you can realize significant computational cost savings by partitioning a model and combining the Abaqus/Standard and Abaqus/Explicit solutions, such as

- when the simulation is principally a candidate for Abaqus/Explicit, but where certain parts of the model can be idealized using substructures in Abaqus/Standard, or
- when the simulation is principally a candidate for Abaqus/Standard, but where complex contact conditions would be handled more effectively by Abaqus/Explicit.

For an example of Abaqus/Standard to Abaqus/Explicit co-simulation, refer to "Dynamic impact of a scooter with a bump," Section 2.4.1 of the Abaqus Example Problems Guide. For detailed information on coupling Abaqus/Standard and Abaqus/Explicit, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, and "Structural-to-structural co-simulation," Section 17.3.1.

Coupling using the MpCCI interface

MpCCI, the multiphysics code coupling interface developed and distributed by the Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI), provides an open system approach for general multidisciplinary simulations between Abaqus and any third-party analysis program that supports MpCCI. MpCCI provides a scalable communication infrastructure and mapping algorithms for multiple physics domains. In a co-simulation using MpCCI, Abaqus communicates in real time with the MpCCI coupling server to exchange fields with the third-party analysis program while each analysis advances its simulation time.

Coupling through MpCCI may occur between Abaqus and any third-party analysis program that supports the MpCCI interface. This includes in-house codes that have the MpCCI adapter embedded. SIMULIA actively supports and qualifies a link between Abaqus and FLUENT for fluid-structure interaction. For further information on coupling using the MpCCI interface, contact Fraunhofer SCAI (www.mpcci.de).

Additional applications

There are many other applications for which co-simulation can be employed with partner products. For example, vehicle ride comfort and durability simulation using FTire from Cosin Scientific Software (www.cosin.eu). For a complete list of qualified partner products, see the Co-simulation page at www.3ds.com/simulia.

Strength of physics coupling

You will typically apply co-simulation techniques to problems where the most complex physics occurs within domains that are handled exclusively within an analysis program (e.g., Abaqus or a CFD analysis program). Due to the comparative numerical simplicity of the numerical techniques applied at the co-simulation interface, the physics controlling the interaction at the interface of the separate analysis domains (the strength of the physics coupling) must be relatively weak for the co-simulation technique to be applied effectively.

Coupling to third-party analysis programs

Analysis domains are coupled in a staggered approach either using a globally explicit manner or an implicit iterative manner; that is, the equations for each domain are solved separately, and loads and boundary conditions are exchanged at the common interface.

In cases where the coupling is sufficiently weak, the coupling may be required only in one direction (such as when an electromagnetic force field contributes to the structural response, but a reverse coupling provides no significant impact on the electromagnetic field).

In an explicit staggered approach, such as the Gauss-Seidel coupling scheme, fields are exchanged only once per coupling step. This coupling strategy is applicable to problems that exhibit weak to moderate physics coupling (for example, aeroelasticity problems where you have air interacting with a relatively stiff structure). The explicit staggered approach requires a small coupling step size.

In an implicit iterative approach, the fields are exchanged multiple times per coupling step until an overall equilibrium is achieved prior to advancing to the next coupling step. Implicit coupling is computationally more expensive per coupling step and generally can be employed to problems exhibiting moderate to strong physics coupling. In general, a larger coupling step size can be employed than for explicit schemes.

Figure 17.1.1–1 illustrates the coupling strength with an analogy in the frequency domain. Consider a lumped parameter dynamic system with a coupling impedance directly related to a response frequency ω. In a staggered solution approach each domain is solved by temporarily ignoring the coupling terms represented by the gray spring and dashpot in Figure 17.1.1–1.

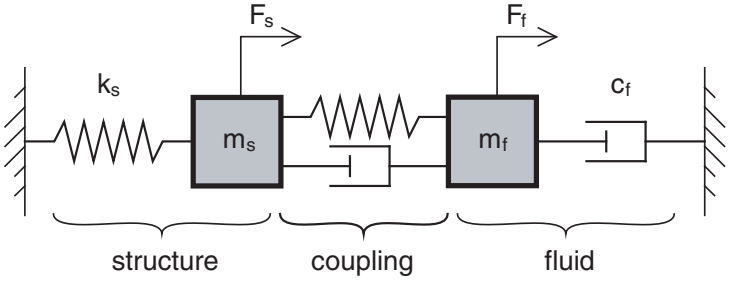


Figure 17.1.1–1 Mechanical impedance analogy.

When the response frequency and coupling impedance are low, a staggered approach will likely provide adequate solution accuracy and performance. However, when the response frequency is high, such that the coupling impedance is relatively large compared to the structure or fluid, you may encounter solution stability issues with the staggered approach.

Coupling in Abaqus/Standard to Abaqus/Explicit co-simulation

The strength of the physics coupling can generally be greater in the coupling of Abaqus/Standard to Abaqus/Explicit using the co-simulation technique. Through communication of "right-hand-side" and "left-hand-side" terms, Abaqus/Standard to Abaqus/Explicit co-simulation provides a robust interface solution across a wide range of problem parameters. In many cases you can choose to have Abaqus/Standard and Abaqus/Explicit each advance their solutions according to their own automatic time incrementation scheme without adversely affecting the interface solution stability.

References

For the latest support information and tips on running FSI simulations and crash safety simulations, see the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base.

17.2 Preparing an Abaqus analysis for co-simulation

- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1

17.2.1 PREPARING AN Abaqus ANALYSIS FOR CO-SIMULATION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CFD

References

- "Co-simulation: overview," Section 17.1.1
- "Structural-to-structural co-simulation," Section 17.3.1
- "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2
- "Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation," Section 17.3.3
- *CO-SIMULATION
- *CO-SIMULATION REGION
- *CO-SIMULATION CONTROLS

Overview

Preparing an Abaqus analysis for co-simulation involves the following:

- identifying an Abaqus analysis step for co-simulation analysis;
- identifying the co-simulation interface regions in the Abaqus model; and
- identifying the fields exchanged during the co-simulation.

This section provides an overview of preparing an Abaqus analysis for a co-simulation. The discussion in this section is general and may not apply to every product pairing. "Co-simulation between Abaqus solvers," Section 17.3, provides setup, execution, and limitation details for co-simulation between Abaqus solvers. For co-simulation between Abaqus and third-party analysis programs, consult the appropriate User's Guide.

Identifying an Abaqus step for co-simulation analysis

The co-simulation event need not begin at the start of the first step in an Abaqus analysis. However, it does need to start with the beginning of an analysis step and end within that analysis step. Hence, you need to define the step durations in Abaqus such that the start of the co-simulation event falls at the beginning of an Abaqus analysis step and to define that particular step so that the co-simulation event ends by the end of that step. Regular loads and boundary conditions for the Abaqus model are specified as usual.

Communication with the coupled analysis is initiated as the co-simulation event begins and is terminated when the co-simulation event time is reached. Abaqus may terminate the co-simulation event when the end of the analysis step is reached prior to the co-simulation event time or when the analysis cannot proceed any further; for example, due to convergence problems. In such a case, a warning message is issued to all clients, and the co-simulation is terminated.

Co-simulation is supported by the following Abaqus procedures:

- "Static stress analysis," Section 6.2.2
- "Quasi-static analysis," Section 6.2.5
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Explicit dynamic analysis," Section 6.3.3
- "Uncoupled heat transfer analysis," Section 6.5.2
- "Fully coupled thermal-stress analysis," Section 6.5.3
- "Incompressible fluid dynamic analysis," Section 6.6.2
- "Piezoelectric analysis," Section 6.7.2
- "Coupled thermal-electrical analysis," Section 6.7.3
- "Eddy current analysis," Section 6.7.5
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1

Input File Usage: Use the following option within a step definition to indicate the beginning of a co-simulation event:

∗CO-SIMULATION, NAME=*name*

Identifying the analysis program communicating with Abaqus during the co-simulation

You can couple Abaqus with another Abaqus analysis or Abaqus with certain third-party analysis programs using the SIMULIA Co-Simulation Engine. For details on coupling with third-party analysis programs, see the respective User's Guides.

Input File Usage: Use the following option to couple Abaqus analyses (except Abaqus/Standard to Abaqus/Explicit) and Abaqus to third-party analysis programs:

∗CO-SIMULATION, NAME=*name*, PROGRAM=MULTIPHYSICS

Use the following option to couple Abaqus/Standard to Abaqus/Explicit:

∗CO-SIMULATION, NAME=*name*, PROGRAM=ABAQUS

Identifying the co-simulation interface region

Interaction between two Abaqus models or between an Abaqus model and a third-party analysis model takes place through a common interface region referred to as the co-simulation interface region. The co-simulation interface region may be a set of discrete points, a surface region, or a volume region. You must be consistent in your interface region definition; if you define a surface co-simulation region in one analysis, then you must define a surface co-simulation region in the other analysis. Furthermore, these co-simulation regions need to be co-located and have the same region boundaries.

Interacting through discrete points

Interaction can occur through a set of discrete points where only nodal position information without element topology information (e.g., tributary area) defines the co-simulation interface region. In this case the spatial mapping is limited to point-to-point mapping, and you must ensure that there are matching nodes between the models.

In Abaqus you can use a node set or a node-based surface to define a co-simulation interface region consisting of discrete points.

Input File Usage: Use the following option to define a node set as a co-simulation region in an Abaqus model:

∗CO-SIMULATION REGION, TYPE=NODE
nodeset_A

Use the following options to define a node-based surface as a co-simulation region in an Abaqus model:

∗SURFACE, TYPE=NODE
nodeset_A
∗CO-SIMULATION REGION, TYPE=SURFACE
node-based surface name

Interacting through a surface

Interaction between distinct domains occurs through a common interface surface. For example, when a fluid interacts with a solid without penetrating it, the fluid-solid interface is defined through a surface. In this case both nodal position and element topology information define the co-simulation interface, and appropriate spatial mapping between dissimilar surface meshes is performed to conservatively map fields.

Input File Usage: Use the following option to define an element-based surface as a co-simulation region in an Abaqus model:

∗CO-SIMULATION REGION, TYPE=SURFACE (default)
element-based surface name

Interacting through a volume

Interaction between overlapping domains occurs through a volume. In this case both nodal position and element topology information define the co-simulation region, and appropriate spatial mapping between dissimilar volume meshes is performed to conservatively map fields.

The interface region is defined by an element set.

Input File Usage: Use the following option to define a volume as a co-simulation region in an Abaqus model:

∗CO-SIMULATION REGION, TYPE=VOLUME
elset_A

Identifying the fields exchanged across a co-simulation interface

The coupling of the domain models can be through loads and/or boundary conditions prescribed at the co-simulation interface. In addition, mass, rotary inertia, and heat capacitance terms can also be exchanged. Based on the physics and the interaction type and its enforcement, you must specify the fields that are imported and/or exported in an Abaqus analysis during the co-simulation.

The co-simulation interface can consist of a group of discrete points (nodes), a surface region, or a volume region. Not all fields can be exchanged across all region types.

This section provides a general overview of all fields available in Abaqus. For detailed information on the fields exchanged between two Abaqus solvers, see "Structural-to-structural co-simulation," Section 17.3.1, and "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2. For detailed information on fields exchanged by Abaqus and a third-party analysis program, see the respective User's Guides.

Input File Usage: Use the following option to import field data over a region into Abaqus:

```
*CO-SIMULATION REGION, IMPORT
region_A, import_field_1
region_A, import_field_2
```

Use the following option to export data from Abaqus:

```
*CO-SIMULATION REGION, EXPORT
region_A, export_field_1
region_A, export_field_2
```

Procedures involving mechanical degrees of freedom

Table 17.2.1–1 lists the fields that can be exchanged for procedures supporting mechanical degrees of freedom (degrees of freedom 1–6), their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 17.2.1–1 Exchanging fields for procedures supporting mechanical degrees of freedom.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
UT or U	Displacement	P, S, V	S, E, C	S, E	L
VT or V	Velocity (transient procedures)	P, S, V	S, E, C	S, E	LT^{-1}
AT or A	Acceleration (transient procedures)	P, S, V	S	S, E	LT^{-2}
UR	Rotations	P, S	S, E	S, E	radians
VR	Angular velocity (transient procedures)	P, S	S, E	S, E	radians T^{-1}
AR	Angular acceleration (transient procedures)	P, S	S	S, E	radians T^{-2}
COORD	Current coordinates	P, S, V		S, E	
CF	Concentrated forces	P, S, V	S, E	S, E	F

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
CM	Concentrated moments	P, S	S, E	S, E	FL
TRSHR	Traction vector	S		C	FL^{-2}
PRESS	Pressure normal to element surface	S	S		FL^{-2}
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

The following procedures support co-simulation using mechanical degrees of freedom:

- "Static stress analysis," Section 6.2.2
- "Quasi-static analysis," Section 6.2.5
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Explicit dynamic analysis," Section 6.3.3
- "Fully coupled thermal-stress analysis," Section 6.5.3
- "Incompressible fluid dynamic analysis," Section 6.6.2
- "Piezoelectric analysis," Section 6.7.2
- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1

Displacements

Displacements (field ID UT or U) for the translational degrees of freedom can be exported by Abaqus/Standard and Abaqus/Explicit. Displacements can be imported by Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD. When imported, displacements are ramped from the values of the previous exchange time point to those of the next target time point. In an implicit dynamic analysis, velocity and acceleration must be imported when importing displacement. The displacements are in the global coordinate system.

Displacements are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit and for surface regions in Abaqus/CFD.

Displacements can be viewed in the Visualization module of Abaqus/CAE.

Velocity and acceleration

Velocity (field ID VT or V) and acceleration (field ID AT or A) for the translational degrees of freedom can be imported and exported by Abaqus/Standard for transient procedures and by Abaqus/Explicit. In an implicit dynamic analysis, when importing velocity or acceleration, all three fields—displacement, velocity, and acceleration—must be imported. Velocity can be imported by Abaqus/CFD. Velocity and acceleration are in the global coordinate system.

Velocity and acceleration are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit and for surface regions in Abaqus/CFD.

Rotations

Rotations (field ID UR) can be imported and exported by Abaqus/Standard and Abaqus/Explicit. In an implicit dynamic analysis, rotational velocity and rotational acceleration must be imported when importing rotations. Rotations are in the global coordinate system.

Rotations are available for points and surface regions.

Rotations can be viewed in the Visualization module of Abaqus/CAE.

Rotational velocity and rotational acceleration

Rotational velocity (field ID VR) and rotational acceleration (field ID AR) can be imported and exported by Abaqus/Standard for transient procedures and by Abaqus/Explicit. In an implicit dynamic analysis, when importing rotational velocity or rotational acceleration, all three fields—rotation, rotational velocity, and rotational acceleration—must be imported. Rotational velocity and rotational acceleration are in the global coordinate system.

Rotational velocity and rotational acceleration are available for points and surface regions.

Current coordinates

Current nodal coordinates (field ID COORD) can be exported by Abaqus/Standard and Abaqus/Explicit. The coordinates are the current coordinates of the deformed structure whether small- or large-displacement analysis is performed. In general, it is preferred to export displacements (field ID UT or U) rather than current coordinates when results are mapped between dissimilar interface regions. In cases where the partner client does not retain the original coordinates, it may be necessary to send current coordinate values rather than displacements.

Current coordinates are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated forces

Concentrated forces (field ID CF), if imported, are ramped from the values of the previous exchange time point to those of the next target time point in Abaqus/Standard and are kept constant over the exchange interval in Abaqus/Explicit. The concentrated forces are in the global coordinate system.

When exporting concentrated forces, Abaqus/Standard transfers reaction forces at interface nodes that have prescribed displacements. The reaction forces are exported in the global coordinate system.

Concentrated forces are available for points, surface regions, and volume regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated normal forces can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CF.

Concentrated moments

Concentrated moments (field ID CM), if imported, are ramped from the values of the previous exchange time point to those of the next target time point in Abaqus/Standard and are kept constant over the exchange interval in Abaqus/Explicit. The concentrated moments are in the global coordinate system.

Concentrated moments are available for points and surface regions in Abaqus/Standard and Abaqus/Explicit.

Concentrated moments can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CM.

Traction vector

The traction vector (field ID TRSHR), supported by Abaqus/CFD, exports the fluid total traction (normal and shear components) on the interface surface. Usually, the exported traction vector is integrated to concentrated forces (field ID CF) when imported into Abaqus/Standard or Abaqus/Explicit in a fluid-structure simulation.

The traction vector is a force vector in the global Cartesian coordinate system.

The traction vector is available for surface regions in Abaqus/CFD.

Normal pressure

Normal pressure (field ID PRESS), supported for import by Abaqus/Standard, is the traction normal component to the surface. Pressure values are ramped from the values of the previous exchange time point to those of the next target time point when imported into Abaqus/Standard. In most cases it is preferred to import concentrated forces (field ID CF) since these contain both the normal and shear traction components. For membrane-like structures it might be preferable to import pressures.

Normal pressure can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable P.

Procedures involving thermal degrees of freedom

Table 17.2.1–2 lists the thermal fields available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.The following procedures support co-simulation using thermal degrees of freedom:

- “Uncoupled heat transfer analysis,” Section 6.5.2
- “Fully coupled thermal-stress analysis,” Section 6.5.3
- “Incompressible fluid dynamic analysis,” Section 6.6.2
- “Coupled thermal-electrical analysis,” Section 6.7.3

Table 17.2.1–2 Exchanging fields for procedures supporting thermal degrees of freedom.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
NT	Temperature as a nodal degree of freedom	P, S, V	S, E	S, E	θ
CFL	Concentrated heat flux at a node	P, S, V	S, E		JT^{-1}
HFL	Heat flux normal to element surface	S	S	C	$JT^{-1}L^{-2}$
CFILM	Film properties	S	S		$JT^{-1}\theta^{-1}$
FILM	Film properties (MpCCI only)	S	S		$JT^{-1}L^{-2}\theta^{-1}$
TEMP	Temperature as a nodal degree of freedom	P, S, V	C		θ
LUMPEDHEATCAPACITANCE	Lumped heat capacitance	P, S, V	S, E	C	$JM^{-1}\theta^{-1}$
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

Nodal temperature

Nodal temperature (field ID NT) can be imported and exported by Abaqus/Standard and Abaqus/Explicit and imported by Abaqus/CFD (as field ID TEMP). Temperature values are ramped from the values of the previous exchange time point to those of the next target time point when imported into Abaqus/Standard and Abaqus/CFD.

Temperature values can be exchanged either on the top surface (SPOS) or the bottom surface (SNEG) of structural elements. Temperatures cannot be exchanged on double-sided surfaces. When exchanging temperatures on both the top and bottom faces, define two different regions; one to exchange temperature on the top face and the other to exchange temperature on the bottom face. For volume regions, only degree of freedom NT11 is used, and it should not be used for exchanging temperature values over volumes discretized by nonthermal element types.

Nodal temperature values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable NT.

Heat flux

Use concentrated heat flux (field ID CFL) for heat entering at a node in Abaqus/Standard and Abaqus/Explicit. Concentrated heat flux is available for points, surface regions, and volume regions.

Heat flux values can be exchanged either on the top surface (SPOS) or the bottom surface (SNEG) of structural elements. Heat flux cannot be exchanged on double-sided surfaces. When exchanging heat flux on both the top and bottom faces, define two different regions; one to exchange heat flux on the top face and the other to exchange heat flux on the bottom face.

Concentrated heat flux values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CFL.

Use surface heat flux (field ID HFL) for a distributed heat flux entering the surface in Abaqus/Standard or distributed heat flux leaving a surface in Abaqus/CFD. Distributed heat flux is available only for surface regions.

Film properties

Use surface film properties (field ID FILM) or concentrated (nodal) film properties (field ID CFILM) to model convection governed by

$$q = -h(\theta_{wall} - \theta_{fluid}),$$

where $\boldsymbol{q}$ is the heat flux entering the surface, $\boldsymbol{h}$ is a film coefficient, θ_{wall} is the wall temperature, and θ_{fluid} is the fluid or ambient temperature. The film coefficient is computed from the heat flux and fluid temperature obtained from the computational fluid dynamics analysis and the wall temperature from the Abaqus/Standard analysis computed during the previous exchange interval, according to

$$h = q/(\theta_{fluid} - \theta_{wall}).$$

Both the film coefficient and fluid temperature are passed into Abaqus/Standard and are kept constant over the subsequent exchange interval. When the fluid and wall temperatures coincide, an arbitrary small value for the heat transfer coefficient is passed into Abaqus. To obtain reasonable film properties for the first exchange interval, you should ensure that the wall temperatures are initialized properly in Abaqus and that you provide a good estimate for the initial fluid temperature.

Film properties are available only for surface regions in Abaqus/Standard.

Heat capacitance

Nodal (lumped) heat capacitance (field ID LUMPEDHEATCAPACITANCE) can be exported by Abaqus/CFD in models in which heat capacitance is defined. Nodal heat capacitance can be imported into Abaqus/Standard and Abaqus/Explicit.

Procedures involving pore fluid pressure

Table 17.2.1–3 lists additional fields that can be exchanged for a coupled pore fluid diffusion/stress analysis, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 17.2.1–3 Exchanging fields for a coupled pore fluid diffusion/stress analysis.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
POR	Pore fluid pressure at a node	P, S, V	S	S	FL^{-2}
CFF	Concentrated fluid flow at a node	P, S, V	S		L^3T^{-1}
RVF	Reaction fluid volume flux due to prescribed pressure	P, S, V		S	L^3T^{-1}
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

The following procedure involving pore fluid pressure supports co-simulation:

- "Coupled pore fluid diffusion and stress analysis," Section 6.8.1

Pore pressure

Nodal pore pressure (field ID POR) can be imported and exported by Abaqus/Standard for points, surface regions, and volume regions.

Nodal pore pressure values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable POR.

Concentrated fluid flow

Fluid flow (field ID CFF) defines the seepage flow at a node. Concentrated fluid flow can be imported by Abaqus/Standard for points, surface regions, and volume regions.

Concentrated fluid flow values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable CFF.

Reaction fluid volume flow

Reaction fluid volume flux (field ID RVF) defines the rate at which fluid volume is entering or leaving the model through the node to maintain the prescribed pore pressure. Reaction fluid volume flux can be exported by Abaqus/Standard for points, surface regions, and volume regions.

Procedures involving electromagnetic response

Table 17.2.1–4 lists additional fields that can be exchanged for an electromagnetic analysis, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 17.2.1–4 Exchanging fields for a electromagnetic analysis.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
EMJH	Joule heating flux due to flow of current	V		S	$JT^{-1}L^{-3}$
EMBF	Magnetic body force intensity vector due to flow of current	V		S	$FT^{-1}L^{-3}$
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

The following procedure involving electromagnetics supports co-simulation:

- "Eddy current analysis," Section 6.7.5

Joule heating flux

The Joule heating flux (field ID EMJH) can be exported by Abaqus/Standard for volume regions. It can be imported in a downstream heat transfer analysis as concentrated nodal heat flux (field ID CFL).

Values for the Joule heating flux can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable EMJH.

Magnetic body force intensity vector

The magnetic body force intensity vector (field ID EMBF) can be exported by Abaqus/Standard for volume regions. It can be imported in a downstream stress analysis as concentrated force (field ID CF).

Magnetic body force intensity vector values can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variable EMBF.

Temperature and independent field variables

Field variables are time-dependent, predefined fields that exist over the spatial domain of the model (see "Predefined fields," Section 34.6.1). Field variables in conjunction with the co-simulation technique extend the possibilities of multiphysics by allowing material point dependencies on an external field defined by another application.

Field variables must be numbered consecutively starting with one. Field variables can be defined:

- by entering the data directly,
- by reading an Abaqus results file or output database file,
- in an Abaqus/Standard user subroutine, and
- through the co-simulation interface.

If field variables are defined by multiple methods, Abaqus processes them in the order defined above. Care needs be taken when field variables are used with structural elements, such as membranes and shells. In this case only the top or bottom face forming the interface region receives a value.

Table 17.2.1–5 lists the temperature and independent field variables available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 17.2.1–5 Exchanging temperature and independent field variables.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
TEMP	Temperature as field variable	V	S		θ
FV1	Field variable 1	V	S		
FV2	Field variable 2	V	S		
FV3	Field variable 3	V	S		
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

The following Abaqus/Standard procedures support import of temperature and independent field variables:

- “Static stress analysis,” Section 6.2.2
- “Quasi-static analysis,” Section 6.2.5
- “Implicit dynamic analysis using direct integration,” Section 6.3.2
- “Fully coupled thermal-stress analysis,” Section 6.5.3
- “Piezoelectric analysis,” Section 6.7.2
- “Eddy current analysis,” Section 6.7.5
- “Coupled pore fluid diffusion and stress analysis,” Section 6.8.1

Temperature

Temperature (field ID TEMP) can be imported by Abaqus/Standard for procedures that allow material properties to be defined as a function of an external temperature field. When imported, temperature values are ramped from the values of the previous exchange time point to those of the next target time

point. Use field ID NT instead of field ID TEMP to import temperature values for thermal procedures (procedures using degrees of freedom 11, 12, etc.).

Temperature can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting element output variable TEMP.

Independent field variables

Independent field variables (field IDs FV1, FV2, and FV3) can be imported by Abaqus/Standard, allowing material properties to be defined as a function of the external fields. When imported, independent field variable values are ramped from the values of the previous exchange time point to those of the next target time point.

Field variables can be viewed in the Visualization module of Abaqus/CAE for an Abaqus/Standard simulation by requesting output variables FV1, FV2, and/or FV3.

Miscellaneous fields

Table 17.2.1–6 lists miscellaneous fields available for co-simulation exchange, their associated field identifiers, the supported co-simulation interface region types, and which Abaqus solvers support import and export of the field values.

Table 17.2.1–6 Exchanging miscellaneous fields.

Field ID	Fields	Interface Type[1]	Abaqus Solver[2]		Units
			Import	Export	
MASS or LUMPEDMASS	Mass	P, S	S, E	S, E, C	M
RI	Rotary inertia	P, S	S	E	ML^2
[1] P (points), S (surface region), V (volume region)					
[2] S (Abaqus/Standard), E (Abaqus/Explicit), C (Abaqus/CFD)					

Lumped mass

Lumped mass values (field ID MASS or LUMPEDMASS) at nodes can be exported by Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD and can be imported by Abaqus/Standard and Abaqus/Explicit.

Lumped mass is available for points and surface regions.

Rotary inertia

Nodal (lumped) rotary inertia (field ID RI) can be imported by Abaqus/Standard and exported by Abaqus/Explicit over points or surface regions for models using structural elements.

Defining the coupling and rendezvousing scheme

Different types of analyses have different time integration requirements that will influence or dictate the frequency of interaction between the analyses in a co-simulation to obtain an accurate and robust solution. For example, consider the difference in time integration between an implicit and an explicit dynamic analysis. Furthermore, Abaqus/Standard can adjust the increment sizes automatically to obtain an economical and accurate solution for transient problems (see "Incrementation" in "Defining an analysis," Section 6.1.2). For example, consider a transient heat transfer analysis modeling a diffusive process; here the analysis may use small time increments at the beginning of the analysis where there is a high gradient in the solution and large time increments toward the end of the analysis when steady state is reached.

The parameters that you use to control these co-simulation exchanges depend on the co-simulation interface that you are using.

- You define the co-simulation algorithm and related exchange parameters in a co-simulation configuration file.
- For structural-to-structural co-simulation using Abaqus/Standard and Abaqus/Explicit, you must also provide co-simulation controls parameters in the input file.

Using the SIMULIA Co-Simulation Engine configuration file

The SIMULIA Co-Simulation Engine employs an independent software component, termed the "director," which defines all aspects of the interaction for co-simulation between analysis programs and provides the necessary instructions to implement the coupling and rendezvousing schemes. You provide the director with relevant parameters for your scheme choices through the co-simulation configuration file. When you use Abaqus/CAE to execute the co-simulation, the configuration file is created for you automatically.

The configuration file must be in Extensible Markup Language (XML) format, which uses the file extension **`xml`**. You can define a configuration file through a predefined template, or you can create a fully elaborated form of the configuration file.

Using predefined configuration templates

For the co-simulation analysis cases described in "Co-simulation between Abaqus solvers," Section 17.3, predefined templates that define common coupling and rendezvousing schemes are available. To use one of the predefined templates, you must create a configuration file with the structure shown below.

```
<?xml version="1.0" encoding="utf-8"?>
Required XML declaration line
<CoupledMultiphysicsSimulation>
Required XML root element; identifies file as describing a multiphysics simulation
    <template_name>
        <template_parameter_1>parameter_1_name</template_parameter_1>
        <template_parameter_2>parameter_2_name</template_parameter_2>
```

```
        <template_parameter_3>parameter_3_name</template_parameter_3>
    </template_name>
    Closure of the template element
</CoupledMultiphysicsSimulation>
Closure of the XML root element
```

You enter the template name and a short list of parameter settings, such as the names of the two analysis jobs and the duration of the analysis. Details of the predefined templates are provided in "Structural-to-structural co-simulation," Section 17.3.1; "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2; and "Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation," Section 17.3.3; as well as information on how to obtain an example configuration file for each template, such as the example shown below for a fluid-to-structural co-simulation.

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
   <template_std_cfd_fsi>
      <Standard_Job>StandardJobName</Standard_Job>
      <Cfd_Job>CfdJobName</Cfd_Job>
      <duration>duration</duration>
   </template_std_cfd_fsi>
</CoupledMultiphysicsSimulation>
```

Using elaborated configuration files

At run time, the SIMULIA Co-Simulation Engine director applies your parameter settings to the template, creating an elaborated configuration file that is then used in the co-simulation analysis. An elaborated configuration file is defined as a configuration file that provides all details of the configuration explicitly without referring to a template.

In cases where predefined templates are not available (such as coupling with an in-house or third-party code) or are insufficient (for example, you want to exchange more variables at the co-simulation interface region or adjust mapping tolerances), you must create an elaborated configuration file. For tips on working with elaborated configuration files, see "Advanced Uses of the SIMULIA Co-Simulation Engine Configuration File" in the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base. For detailed information about the elaborated configuration file, see the SIMULIA Co-Simulation Engine Application Programing Interface (API) documentation.

Coupling and rendezvousing schemes for elaborated configuration files

You define the co-simulation coupling and rendezvousing schemes in an elaborated configuration file.

Coupling scheme

The coupling scheme defines the sequence of exchanges between analysis programs and whether a coupled simulation can be run in a serial, parallel, or implicit iterative manner. When deciding on the

coupling scheme, you should consider solution stability issues as well as the utilization impact on your computing resources

Parallel explicit coupling scheme (Jacobi)

In a parallel explicit coupling scheme, both simulations are executed concurrently, exchanging fields to update the respective solutions at the next target time. The parallel coupling scheme may make more efficient use of computing resources; however, it is considered less stable than the sequential scheme and should be employed only for weakly coupled physics simulations using small coupling steps. The co-simulation partner analysis must also specify a Jacobi coupling algorithm.

Sequential explicit coupling scheme (Gauss-Seidel)

In a sequential explicit coupling scheme, the simulations are executed in sequential order. One analysis leads while the other analysis lags the co-simulation. The co-simulation partner analysis must also specify a Gauss-Seidel coupling algorithm.

The sequential explicit coupling scheme should be employed only for weakly coupled physics simulations using small coupling steps.

Iterative coupling scheme

In an iterative coupling scheme, the simulations are executed in sequential order. One analysis leads while the other analysis lags the co-simulation. Multiple exchanges per coupling step are performed until termination criteria are met.

The termination criteria depend on the analyses in the co-simulation; for co-simulation between Abaqus and third-party analysis products, consult the appropriate User's Guide.

Coupling step size

The coupling step is the period between two consecutive exchanges and consequently defines the frequency of exchange between the analyses in a co-simulation. The coupling step size is established at the beginning of each coupling step and is used to compute the target time (the time when the next synchronized exchange occurs).

The methods available in Abaqus for computing the coupling step size are described in the sections below. To determine the methods available for a co-simulation partner analysis, consult the appropriate third-party program documentation.

Constant coupling step size

A constant user-defined coupling step size is the most basic method of defining a coupling step size. Both analyses advance while exchanging data at target points according to

$$t_{i+1} = t_i + \Delta t_c,$$

where Δt_c is a value that defines the coupling step size to be used throughout the coupled simulation, t_{i+1} is the target time, and t_i is the time at the start of the coupling step.

Minimum coupling step size

This method selects the minimum of the coupling step sizes suggested by each analysis. Abaqus always uses the next increment suggested by its automatic incrementation as its suggested coupling step size.

Maximum coupling step size

This method selects the maximum of the coupling step sizes suggested by each analysis. Abaqus always uses the next increment suggested by its automatic incrementation as its suggested coupling step size.

Importing the coupling step size

Abaqus can import a coupling step size suggested by the co-simulation partner analysis.

Exporting the coupling step size

Abaqus can export a suggested coupling step size to the co-simulation partner analysis.

Time incrementation scheme

Abaqus may take multiple increments per coupling step, or you can force Abaqus to use a single increment per coupling step.

Typically, Abaqus may perform several increments (referred to as "subcycling") during the coupling step. During subcycling, Abaqus/Standard ramps the loads and boundary conditions (with the exception of film properties) from the values at the end of the previous coupling step to the values at the target time, while in Abaqus/Explicit the loads are applied at the start of the coupling step and kept constant over the coupling step.

Subcycling allows Abaqus to use its own time incrementation to reach the target coupling time; specifically, it allows Abaqus to cut back the increment size if there are nonlinear events that require the increment size to be reduced.

In certain cases you may force Abaqus to use a time increment size dictated by the coupling step size (i.e., no subcycling). This allows both solvers to use the same time incrementation and avoid interpolation of quantities during the coupling step. When proceeding in this "lockstep" manner, Abaqus will not be able to reduce the time increment to resolve nonlinear events and, consequently, will terminate the simulation in cases where the nonlinear events require that the increment size be reduced.

Model dimension and coordinate systems

Two-dimensional and three-dimensional Abaqus models are fully supported. Axisymmetric Abaqus models are supported only for Abaqus/Standard to Abaqus/Explicit co-simulation. For co-simulations that do not support two-dimensional and axisymmetric models, you can represent these models as a three-dimensional slice of unit thickness (or wedge element) with the appropriate boundary conditions applied.

Vector quantities are defined according to Abaqus conventions; the first component represents the quantity along the x-axis, the second quantity represents the quantity along the y-axis, and the third quantity represents the quantity along the z-axis (for three-dimensional models). For axisymmetric models in Abaqus the axis of revolution is about the y-axis. These conventions apply to both the exported and the imported vector quantities.

All exported vector quantities are expressed in the global coordinate system of the Abaqus model, ignoring any transformation definitions. Similarly, the third-party program must provide vector quantities that are imported into Abaqus in the global coordinate system of the Abaqus model.

The third-party analysis program may use different conventions, please refer to the appropriate third-party program documentation for further modeling details and/or limitations.

Unit system

Abaqus does not require that the analysis be run with a particular unit system. In general, the unit system used in creating the Abaqus model may not be the same as that used with the third-party program model. When the two unit systems differ, the fields exchanged between the two programs must go through a transformation of units. Refer to the appropriate third-party program documentation for further modeling details.

Restarting a co-simulation

Field imported into Abaqus/Standard, Abaqus/Explicit, or Abaqus/CFD are not saved to the Abaqus restart database. Thus, to restart a co-simulation, the coupled analysis must send the fields at the start of the restart analysis. These fields must balance the conjugate fields computed by the Abaqus analysis such that equilibrium is maintained. You must synchronize the restart information written between the analyses to ensure that the simulation is restarted at the same solution (step) time. For more information, see "Synchronizing restart information written in a co-simulation" in "Restarting an analysis," Section 9.1.1. Specifically, the solution time for the particular step/increment number from which Abaqus is restarted must correspond to the coupled analysis solution.

Limitations

The following limitations apply:

- The steps in the Abaqus model must be defined such that the co-simulation fits entirely within a single Abaqus step. Further, there can be only one co-simulation in the Abaqus job. You can use the restart capability to perform multiple co-simulations for an analysis (see "Restarting an analysis," Section 9.1.1).
- A co-simulation surface or volume defined over beam, pipe, and truss elements or defined over the edges of three-dimensional elements cannot be used as an interface region. You should use discrete points to transfer loads and boundary conditions.
- A co-simulation surface or volume defined over modified triangular elements or modified tetrahedral elements cannot be used as an interface region.
- Quadratic coupled temperature-displacement elements cannot be used as an interface region in a co-simulation using the coupled temperature-displacement procedure.
- When performing a co-simulation, output at specified time points may not be satisfied at the requested times, depending on the synchronization parameters.

There may be further limitations depending on the third-party analysis program being used. For more information, refer to the appropriate third-party program documentation.

17.3 Co-simulation between Abaqus solvers

- "Structural-to-structural co-simulation," Section 17.3.1
- "Fluid-to-structural and conjugate heat transfer co-simulation," Section 17.3.2
- "Electromagnetic-to-structural and electromagnetic-to-thermal co-simulation," Section 17.3.3
- "Executing a co-simulation," Section 17.3.4

17.3.1 STRUCTURAL-TO-STRUCTURAL CO-SIMULATION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References

- "Co-simulation: overview," Section 17.1.1
- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1
- *CO-SIMULATION
- *CO-SIMULATION CONTROLS
- "Defining a Standard-Explicit co-simulation interaction," Section 15.13.14 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- Chapter 26, "Co-simulation," of the Abaqus/CAE User's Guide

Overview

Co-simulation between two structural solvers (solvers exchanging displacements and/or rotations and the conjugate fields' force and/or moments) represents a very strong physics coupling and requires special treatment at the co-simulation interface. Abaqus supports co-simulation between Abaqus/Standard and Abaqus/Explicit by providing specialized interface handling. Although you can perform a structure-to-structure co-simulation between two Abaqus/Standard analyses or between two Abaqus/Explicit analyses, it is not recommended due to the lack of proper handling at the interface.

This section discusses analysis setup, execution, and limitation details specific to Abaqus/Standard to Abaqus/Explicit co-simulation.

Refer to "Dynamic impact of a scooter with a bump," Section 2.4.1 of the Abaqus Example Problems Guide, for an example of Abaqus/Standard to Abaqus/Explicit co-simulation.

Identifying the Abaqus step for the co-simulation analysis

The following Abaqus/Standard analysis procedures can be used for an Abaqus/Standard to Abaqus/Explicit co-simulation:

- "Static stress analysis," Section 6.2.2
- "Implicit dynamic analysis using direct integration," Section 6.3.2

The following Abaqus/Explicit analysis procedure can be used for an Abaqus/Standard to Abaqus/Explicit co-simulation:

- "Explicit dynamic analysis," Section 6.3.3

Input File Usage: Use the following option within a step definition for an Abaqus/Standard to Abaqus/Explicit co-simulation:

*CO-SIMULATION, PROGRAM=ABAQUS

Abaqus/CAE Usage: Use the following option for an Abaqus/Standard to Abaqus/Explicit co-simulation:

Interaction module: **Create Interaction**: **Standard-Explicit co-simulation**

Identifying the co-simulation interface region

Interaction between the Abaqus/Standard and Abaqus/Explicit models takes place through a common interface region.

You can specify an interface region using either node sets or surfaces when coupling Abaqus/Standard to Abaqus/Explicit. You must, however, be consistent in your region definition in Abaqus/Standard and Abaqus/Explicit; if you define a co-simulation region with a node set or node-based surface in one analysis, you must use the same type of co-simulation region definition in the other analysis. For node-based surfaces the nodes have to be co-incident since no topology information is provided to conservatively map fields between the Abaqus/Standard and Abaqus/Explicit models. Likewise, if you define a co-simulation region with an element-based surface in one analysis, you must define your co-simulation region with an element-based surface in the other analysis.

You may have dissimilar meshes in regions shared in the Abaqus/Standard and Abaqus/Explicit model definitions. In some cases, however, you can improve solution stability and accuracy by ensuring that you have matching nodes at the interface (see “Dissimilar mesh-related limitations). In these cases you can use the modeling practice described in “Ensuring matching nodes at the interface regions,” Section 26.4 of the Abaqus/CAE User’s Guide, to ensure these matching nodes.

Input File Usage: Use the following option to define an element-based or node-based surface as a co-simulation region in an Abaqus model:

∗CO-SIMULATION REGION, TYPE=SURFACE (default)
surface_A

Use the following option to define a node set as a co-simulation region in an Abaqus model:

∗CO-SIMULATION REGION, TYPE=NODE
nodeset_A

Only one ∗CO-SIMULATION REGION option can be defined in each Abaqus analysis. In addition, only one node set or surface can be defined.

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Standard-Explicit co-simulation**: **Surface** or **Node Region**: select region

Identifying the fields exchanged across a co-simulation interface

For Abaqus/Standard to Abaqus/Explicit co-simulation, you do not define the fields exchanged; they are determined automatically according to the procedures and co-simulation parameters used.

Defining the rendezvousing scheme

For structural-to-structural co-simulation, you must specify co-simulation controls for each analysis and create a configuration file; the consistency of the parameters is confirmed during execution. The SIMULIA Co-Simulation Engine configuration file is used to define the time incrementation process and the frequency of exchange between the two Abaqus analyses. Abaqus/CAE automatically creates and uses this configuration file. If you are not using Abaqus/CAE to perform the co-simulation, you must create the configuration file manually.

Predefined templates are available for the following co-simulation schemes:

- You can force Abaqus/Standard to use the same increment size as Abaqus/Explicit, or
- you can allow the increment sizes in Abaqus/Standard to differ from those in Abaqus/Explicit (subcycling).

You refer to these predefined templates when you create your configuration files.

Input File Usage: Use both of the following options to specify co-simulation controls:

∗CO-SIMULATION, PROGRAM=ABAQUS, CONTROLS=*name*
∗CO-SIMULATION CONTROLS, NAME=*name*

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Standard-Explicit co-simulation**

Time incrementation scheme

You can force Abaqus/Standard to use the same increment size as Abaqus/Explicit, or you can allow the increment sizes in Abaqus/Standard to differ from those in Abaqus/Explicit (subcycling). The time incrementation scheme that you choose for coupling affects the solution computational cost and accuracy but not the solution stability.

The subcycling scheme is frequently the most cost effective since Abaqus/Standard time increments, free of any forced co-simulation time incrementation constraints, are commonly much longer than Abaqus/Explicit time increments. The subcycling scheme, however, may be less cost effective when a large portion of the nodes in the model are at the co-simulation interface. This is because Abaqus/Standard performs a set of stabilization operations at the interface (a “free solve”) for each increment in the Abaqus/Explicit analysis. These free-solve operations require an implicit solution of a dense system of equations that scale with the number of interface nodes. In cases of a large number of interface nodes the computational cost of this interface solve can exceed any cost savings seen due to subcycling. Hence, for a model where a significant share of the nodes are at the co-simulation interface performance may be poorer with the subcycling scheme.

Forcing Abaqus/Standard to use the same increment size as Abaqus/Explicit

You can force Abaqus/Standard to match the increment size of Abaqus/Explicit, and fields will be exchanged at each of the shared increments.

Input File Usage: Use the following option in the Abaqus/Standard analysis and in the Abaqus/Explicit analysis:

∗CO-SIMULATION CONTROLS, TIME INCREMENTATION=LOCKSTEP

Abaqus/CAE Usage: Use the following input in the Abaqus/Standard analysis and in the Abaqus/Explicit analysis:

Interaction module: **Create Interaction**: **Standard-Explicit co-simulation**: **Incrementation control**: **Lock time steps**

Allowing the increment sizes in Abaqus/Standard to differ from those in Abaqus/Explicit

You can allow the Abaqus/Standard increment size to differ from those in Abaqus/Explicit (subcycling). In this case fields will be exchanged as needed.

Input File Usage: Use the following option in the Abaqus/Standard analysis and in the Abaqus/Explicit analysis:

```
*CO-SIMULATION CONTROLS,
TIME INCREMENTATION=SUBCYCLE
```

Abaqus/CAE Usage: Use the following input in the Abaqus/Standard analysis and in the Abaqus/Explicit analysis:

Interaction module: **Create Interaction**: **Standard-Explicit co-simulation**: **Incrementation control**: **Allow subcycling**

Controlling interface matrix factorization frequency

For the subcycling time incrementation scheme an interface solve is performed, by default, in Abaqus/Standard for every Abaqus/Explicit increment. This solve can be significantly costly for two reasons. First, the interface matrix used for the interface solve is dense and its size scales with the number of interface nodes. Second, the interface matrix changes every Abaqus/Explicit increment, requiring factorization in Abaqus/Standard for every Abaqus/Explicit increment. You can reduce the impact of this cost by approximating the interface matrix and factorizing it typically once for the duration of an Abaqus/Standard increment, rather than for each Abaqus/Explicit increment. However, if the Abaqus/Explicit stable time increment changes significantly, the interface matrix is refactored for stability reasons.

Allowing Abaqus/Standard to factorize the interface matrix every Abaqus/Explicit increment

Factorizing the interface matrix every Abaqus/Explicit increment is the default approach.

Input File Usage: Use the following option in the Abaqus/Standard analysis:

```
*CO-SIMULATION CONTROLS,
FACTORIZATION FREQUENCY=EXPLICIT INCREMENT
```

Abaqus/CAE Usage: Factorizing the interface matrix every Abaqus/Explicit increment is used by default in Abaqus/CAE.

Forcing Abaqus/Standard to factorize the interface matrix once per Abaqus/Standard increment

When the number of interface nodes is large, the cost of the interface factorization can be significantly reduced by using this approach. Only the interface matrix factorization is performed once per

Abaqus/Standard increment; the interface solve is performed every Abaqus/Explicit increment using this factorized interface matrix. Since this approach approximates the interface matrix, it may slightly increase the drift in the displacement solution at the co-simulation interface. The performance gain with this method depends on the number of interface nodes, the subcycling ratio (which is the ratio between Abaqus/Standard and Abaqus/Explicit increments), and the size of the models. For models with greater than 100 interface nodes and a subcycling ratio greater than 50, this method typically reduces the analysis time by a factor between 1.2 and 3.0. The performance gain increases for larger subcycling ratios and decreases for larger models.

Input File Usage: Use the following option in the Abaqus/Standard analysis:

```
*CO-SIMULATION CONTROLS,
FACTORIZATION FREQUENCY=STANDARD INCREMENT
```

Abaqus/CAE Usage: Factorizing the interface matrix once per Abaqus/Standard increment is not supported in Abaqus/CAE.

Coupling step size

The coupling step size is the period between two consecutive co-simulation data exchanges between Abaqus/Standard and Abaqus/Explicit and always equals the current Abaqus/Explicit increment size.

When using the subcycling method, this data exchange does not represent a constraint on Abaqus/Standard incrementation; the Abaqus/Standard analysis advances in time using its normal time incrementation logic.

Creating a configuration file

You can use predefined templates to create a configuration file for the coupling schemes described above. Table 17.3.1–1 describes the two predefined templates available for Abaqus/Standard to Abaqus/Explicit co-simulation and lists example configuration files that you can review.

Table 17.3.1–1 Templates for structural-to-structural co-simulation.

Structural-to-structural co-simulation: subcycling	Coupling scheme: Allow Abaqus/Explicit to subcycle
	`template_std_xpl_subcycle`
	Example file: **`exa_std_xpl_subcycle.xml`**
Structural-to-structural co-simulation: lockstep	Coupling scheme: Abaqus/Standard and Abaqus/Explicit use a single increment per coupling step (lockstep)
	`template_std_xpl_lockstep`
	Example file: **`exa_std_xpl_lockstep.xml`**

To obtain an example configuration file, you can use the **abaqus fetch** utility. For example, to obtain the example for Abaqus/Standard to Abaqus/Explicit subcycling, use the following command:

```
abaqus fetch job=exa_std_xpl_lockstep
```

The example file **exa_std_xpl_lockstep.xml** is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
   <template_std_xpl_subcycle>
      <Standard_Job>standard_job_name</Standard_Job>
      <Explicit_Job>explicit_job_name</Explicit_Job>
      <duration>duration_value</duration>
   </template_std_xpl_subcycle>
</CoupledMultiphysicsSimulation>
```

In certain cases you may need to use co-simulation configuration features that are not described in the predefined templates. For example, you may wish to change the dissimilar mesh mapping search tolerances; these tolerances are available generally in the configuration file but are not described in the predefined templates. For these cases, you must create an elaborated configuration file; for more information, see "Using elaborated configuration files" in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

Executing the coupled analysis

You execute the co-simulation interactively in Abaqus/CAE or from the command line, as described in "Executing a co-simulation," Section 17.3.4.

Diagnostics information

The Abaqus/Standard job provides detailed descriptions of co-simulation operations in the message (**.msg**) file. For the subcycling scheme the status (**.sta**) file provides summary information indicating when the interface calculations followed by re-solve of the increment are made, as shown in the following example status file. The **E** suffix in the attempt-count entry (column 3) indicates an increment performing interface calculations. An increment without the **E** suffix indicates re-solve of the increment.

```
SUMMARY OF JOB INFORMATION:
STEP   INC ATT SEVERE EQUIL TOTAL  TOTAL      STEP       INC OF       DOF    IF
               DISCON ITERS ITERS  TIME/    TIME/LPF    TIME/LPF    MONITOR RIKS
               ITERS               FREQ
  1     1   1E    0     1     1  0.000       0.000      0.001000
  1     1   1     0     3     3  0.00100     0.00100    0.001000
  1     2   1E    0     1     1  0.00100     0.00100    0.001000
  1     2   1     0     3     3  0.00200     0.00200    0.001000
  1     3   1E    0     1     1  0.00200     0.00200    0.001000
  1     3   1     0     2     2  0.00300     0.00300    0.001000
  1     4   1E    0     1     1  0.00300     0.00300    0.001000
  1     4   1     0     3     3  0.00400     0.00400    0.001000
```

The Abaqus/Explicit job provides summary descriptions of co-simulation operations in the status (**.sta**) file.

Limitations

The following limitations apply to Abaqus/Standard to Abaqus/Explicit co-simulation in addition to the limitations discussed in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

General limitations

- Displacement compatibility at the co-simulation interface is not maintained when you allow the Abaqus/Standard increment size to differ from that in Abaqus/Explicit (i.e., when you specify subcycling as a co-simulation time incrementation control). In this case velocity compatibility is maintained, but you may see small amounts of displacement mismatch between Abaqus/Standard and Abaqus/Explicit as the simulation advances in time. This "drift" is more pronounced if severe nonlinearity such as plastic deformation occurs at the co-simulation interface. You can control this drift by adjusting Abaqus/Standard solution parameters so that the Abaqus/Standard increment size is reduced (e.g., by limiting the maximum time increment size or specifying a smaller half-increment residual tolerance for implicit dynamic analyses).
- Nodal transformations are not permitted on the co-simulation region nodes.
- The ALE technique may not be used in elements attached to co-simulation region nodes.
- Fully coupled temperature-displacement elements can be used, but no temperature quantities are exchanged.
- An Abaqus/Standard static stress analysis cannot be used with the lockstep time incrementation scheme in Abaqus/Standard to Abaqus/Explicit co-simulation.
- Only points and surface regions are allowed; coupling volume regions is not supported.
- Only a single region can be defined as the interface region; multiple interface regions are not supported.

Dissimilar mesh-related limitations

When your Abaqus/Standard and Abaqus/Explicit co-simulation region meshes differ, the following limitations apply:

- Solution accuracy may be affected when your co-simulation region meshes are not uniform in the presence or absence of rotational degrees of freedom; for example, if a continuum element mesh is locally reinforced with beam or shell elements at the co-simulation region interface.
- In cases where the stress state near the co-simulation interface is significant (approaching 1% or more) relative to the material stiffness, you may observe appreciable irregular mesh distortion if the mesh density adjacent to the co-simulation region differs greatly between the Abaqus/Explicit and Abaqus/Standard models. For example, this effect is common with large deformation of hyperelastic materials. You can minimize this effect by choosing a similar or finer mesh at the Abaqus/Standard co-simulation region when using the subcycling time integration scheme or by choosing a similar or finer mesh at the Abaqus/Explicit co-simulation region when using the lockstep time integration scheme.

Abaqus/Standard analysis limitations

Abaqus/Standard elements that have no equivalent degree-of-freedom counterpart in Abaqus/Explicit cannot be connected to co-simulation region nodes. These elements include

- Axisymmetric elements with twist degrees of freedom (the CGAX element family)
- Axisymmetric solid elements with asymmetric deformation (the CAXA element family)
- Generalized plane strain elements (the CPEG element family)
- Coupled pore pressure-displacement elements
- Heat transfer and thermal-electrical elements
- Acoustic elements
- Piezoelectric elements

The following specific limitations also apply:

- A co-simulation region node cannot be a slave node in a tie constraint, an MPC constraint, or a kinematic coupling constraint.

Abaqus/Explicit analysis limitations

Stability and accuracy of the co-simulation solution may be adversely affected when the following model features are defined at or near the co-simulation region:

- Connector elements connected to co-simulation region nodes.
- Co-simulation region nodes that participate in a tie constraint, an MPC constraint, or a kinematic coupling constraint.

When using these features, you should compare the Abaqus/Standard and Abaqus/Explicit solutions (e.g., compatibility of the displacement history) at the co-simulation interface as an indicator of solution accuracy.

17.3.2 FLUID-TO-STRUCTURAL AND CONJUGATE HEAT TRANSFER CO-SIMULATION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CFD Abaqus/CAE

References

- "Co-simulation: overview," Section 17.1.1
- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1
- *CO-SIMULATION
- *CO-SIMULATION CONTROLS
- "Defining a fluid-structure co-simulation interaction," Section 15.13.15 of the Abaqus/CAE User's Guide, in the HTML version of this guide
- Chapter 26, "Co-simulation," of the Abaqus/CAE User's Guide

Overview

This section discusses analysis setup, execution, and limitation details specific to fluid-to-structural co-simulation and conjugate heat transfer using Abaqus/CFD and Abaqus/Standard or Abaqus/Explicit.

Refer to "Conjugate heat transfer analysis of a component-mounted electronic circuit board," Section 6.1.1 of the Abaqus Example Problems Guide, for an example of conjugate heat transfer.

Identifying the Abaqus step for the co-simulation analysis

The following Abaqus/CFD analysis procedure can be used for co-simulation with Abaqus/Standard or Abaqus/Explicit:

- "Incompressible fluid dynamic analysis," Section 6.6.2

The following Abaqus/Standard analysis procedures can be used for co-simulation with Abaqus/CFD:

- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Uncoupled heat transfer analysis," Section 6.5.2

The following Abaqus/Explicit analysis procedures can be used for co-simulation with Abaqus/CFD:

- "Explicit dynamic analysis," Section 6.3.3
- "Fully coupled thermal-stress analysis in Abaqus/Explicit" in "Fully coupled thermal-stress analysis," Section 6.5.3

Input File Usage: Use the following option within a step definition for an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation:

*CO-SIMULATION, PROGRAM=MULTIPHYSICS

Abaqus/CAE Usage: Use the following option for an Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation:

Interaction module: **Create Interaction**: **Fluid-Structure Co-simulation boundary**

Identifying the co-simulation interface region

You specify an interface region using surfaces when coupling Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit. You must define an element-based surface, and you can specify only one surface to be used as the interface region in the analysis. You may have dissimilar meshes in regions shared in the model definitions.

Input File Usage: Use the following option to define an element-based surface as a co-simulation region:

*CO-SIMULATION REGION, TYPE=SURFACE
surface_A

Abaqus/CAE Usage: Interaction module: **Create Interaction**: **Fluid-Structure Co-simulation boundary**: select surface region

Identifying the fields exchanged across a co-simulation interface

For Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation, see the tables in "Identifying the fields exchanged across a co-simulation interface" in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, for lists of fields that are available for co-simulation exchange. When using Abaqus/CAE, the fields exchanged are determined automatically by Abaqus/CAE.

Defining the rendezvousing scheme

The SIMULIA Co-Simulation Engine configuration file is used to define the time incrementation process and the frequency of exchange between the two Abaqus analyses. Abaqus/CAE automatically creates and uses this configuration file. If you are not using Abaqus/CAE to perform the co-simulation, you must create the configuration file manually.

Predefined templates are available for commonly used coupling schemes. You refer to these templates when you create configuration files. This section describes the rendezvous scheme settings and predefined configuration file templates.

Defining the coupling scheme

The sequential explicit coupling scheme (also referred to as the Gauss-Seidel coupling algorithm) is the only coupling scheme available for Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation. In all of the predefined templates, the Abaqus/CFD analysis lags the co-simulation and the Abaqus/Standard or Abaqus/Explicit analysis leads the co-simulation. For conjugate heat transfer, the Abaqus/CFD analysis can either lag or lead the co-simulation. For fluid-structure interaction, the Abaqus/CFD analysis must lag the co-simulation and the Abaqus/Standard or Abaqus/Explicit analysis must lead the co-simulation.

Coupling step size

The coupling step size is the period between two consecutive co-simulation data exchanges. The coupling step size is determined automatically based on the type of analysis and is used to obtain time-accurate solutions for the coupled physics problem. For fluid-structure interaction (FSI) and conjugate heat transfer (CHT) analyses that couple Abaqus/CFD and Abaqus/Standard, the coupling step size is the minimum of the time step sizes determined by the automatic time incrementation schemes of the individual analyses. For FSI problems that couple Abaqus/CFD and Abaqus/Explicit, Abaqus/Explicit imports the coupling step size from Abaqus/CFD; consequently, Abaqus/CFD exports the coupling step size to Abaqus/Explicit.

Time incrementation scheme

Depending on the type of analysis, Abaqus may either perform one increment (referred to as "lockstep") or several increments (referred to as "subcycling") per coupling step. For FSI analyses that couple Abaqus/CFD and Abaqus/Explicit, Abaqus/Explicit typically uses subcycling while Abaqus/CFD uses lockstep behavior.

Creating a configuration file

You can use predefined templates to create a configuration file for the coupling schemes described above. Table 17.3.2–1 describes the predefined templates available for fluid-to-structural co-simulation and conjugate heat transfer and lists example configuration files that you can review.

To obtain an example configuration file, you can use the **abaqus fetch** utility. For example, to obtain the example for Abaqus/Standard to Abaqus/CFD conjugate heat transfer, use the following command:

```
abaqus fetch job=exa_std_cfd_cht
```

The example file **exa_std_cfd_cht** is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
   <template_std_cfd_cht>
      <Standard_Job>standard_job_name</Standard_Job>
      <Cfd_Job>cfd_job_name</Cfd_Job>
      <duration>duration_value</duration>
   </template_std_cfd_cht>
</CoupledMultiphysicsSimulation>
```

In certain cases you may need to use co-simulation configuration features that are not described in the predefined templates. For example, you may wish to change the dissimilar mesh mapping search tolerances; these tolerances are available generally in the configuration file but are not described in the predefined templates. For these cases, you must create an elaborated configuration file; for more information, see "Using elaborated configuration files" in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

Table 17.3.2–1 Templates for fluid-to-structural co-simulation and conjugate heat transfer.

Fluid-to-structural co-simulation: Abaqus/Standard and Abaqus/CFD	Coupling scheme: • Abaqus/Standard analysis leads • Coupling step size is based on the minimum of both analyses • Allow Abaqus/Standard to subcycle; Abaqus/CFD uses a single increment per coupling step (lockstep)
	`template_std_cfd_fsi`
	Example file: **`exa_std_cfd_fsi`**
Fluid-to-structural co-simulation: Abaqus/Explicit and Abaqus/CFD	Coupling scheme: • Abaqus/Explicit analysis leads • Abaqus/CFD defines the coupling step size • Allow Abaqus/Explicit to subcycle; Abaqus/CFD uses a single increment per coupling step (lockstep)
	`template_xpl_cfd_fsi`
	Example file: **`exa_xpl_cfd_fsi`**
Conjugate heat transfer: Abaqus/Standard and Abaqus/CFD	Coupling scheme: • Abaqus/Standard analysis leads • Coupling step size is based on the minimum of both analyses • Allow Abaqus/Standard to subcycle; Abaqus/CFD uses a single increment per coupling step (lockstep)
	`template_std_cfd_cht`
	Example file: **`exa_std_cfd_cht`**
Conjugate heat transfer: Abaqus/Explicit and Abaqus/CFD	Coupling scheme: • Abaqus/Explicit analysis leads • Abaqus/CFD defines the coupling step size • Allow Abaqus/Explicit to subcycle; Abaqus/CFD uses a single increment per coupling step (lockstep)
	`template_xpl_cfd_cht`
	Example file: **`exa_xpl_cfd_cht`**

Executing the coupled analysis

You execute the co-simulation interactively in Abaqus/CAE or from the command line, as described in "Executing a co-simulation," Section 17.3.4. By default, when coupling Abaqus/CFD to Abaqus/Explicit, the Abaqus/Explicit packager and analysis are both run in single precision.

Limitations

The following limitations apply to Abaqus/CFD to Abaqus/Standard or to Abaqus/Explicit co-simulation in addition to the limitations discussed in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

General limitation

Only a single region can be defined as the interface region; multiple interface regions are not supported.

Abaqus/Standard and Abaqus/Explicit analysis limitations

The following Abaqus/Standard and Abaqus/Explicit elements cannot be used in a co-simulation with Abaqus/CFD:

- Axisymmetric elements with twist degrees of freedom (the CGAX element family)
- Axisymmetric solid elements with asymmetric deformation (the CAXA element family)
- Generalized plane strain elements (the CPEG element family)
- Coupled pore pressure-displacement elements
- Acoustic elements
- Piezoelectric elements

17.3.3 ELECTROMAGNETIC-TO-STRUCTURAL AND ELECTROMAGNETIC-TO-THERMAL CO-SIMULATION

Product: Abaqus/Standard

References

- "Co-simulation: overview," Section 17.1.1
- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1
- *CO-SIMULATION

Overview

This section discusses analysis setup and execution details specific to electromagnetic-to-structural and electromagnetic-to-thermal co-simulation using Abaqus/Standard procedures. Co-simulation between a time-harmonic or transient electromagnetic analysis and a static, transient implicit dynamic, coupled temperature-displacement, or transient heat transfer analysis is supported.

An electromagnetic to a heat transfer co-simulation analysis is useful for applications such as induction heating, which involves two-way coupling: the Joule heat production from the electromagnetic analysis drives a heat transfer analysis and determines the temperature distribution, while the temperature distribution, in turn, affects the electromagnetic fields through temperature-dependent material properties (such as electrical conductivity and magnetic permeability). For electromagnetic-to-thermal coupling, co-simulation between a time-harmonic or transient electromagnetic analysis and a transient heat transfer analysis is supported.

An electromagnetic to transient implicit dynamic analysis is useful for applications such as electromagnetic forming, where the Lorentz body forces from an electromagnetic analysis drive a transient dynamic analysis. Co-simulation between a transient electromagnetic analysis and a static or transient implicit dynamic analysis is supported. However, the coupling is only one way; that is, the effects of deformation of parts of the domain (metal work piece, in this case) on the electromagnetic fields is not accounted for. Hence, such analysis should be used only when the effects of deformation on the electromagnetic fields are relatively small.

Identifying the Abaqus step for the co-simulation analysis

The following Abaqus/Standard analysis procedures can be used for an electromagnetic-to-structural co-simulation:

- "Static stress analysis," Section 6.2.2
- "Implicit dynamic analysis using direct integration," Section 6.3.2
- "Uncoupled heat transfer analysis," Section 6.5.2
- "Eddy current analysis," Section 6.7.5

The following Abaqus/Standard analysis procedures can be used for electromagnetic-to-thermal co-simulation:

- "Uncoupled heat transfer analysis," Section 6.5.2
- "Eddy current analysis," Section 6.7.5

Input File Usage: Use the following option within a step definition for an Abaqus/Standard to Abaqus/Standard co-simulation:

∗CO-SIMULATION, PROGRAM=MULTIPHYSICS

Identifying the co-simulation interface region

Interaction between the electromagnetic and structural models occurs through a common volume interface region.

You must specify the volume interface region using element sets between the Abaqus/Standard analyses. You must be consistent in your region definition in both the Abaqus/Standard simulations; in other words, you must define the same interface region in both the analyses.

You can have dissimilar meshes in regions shared in the two model definitions.

Input File Usage: Use the following option to define an element-based co-simulation region in an Abaqus/Standard model:

∗CO-SIMULATION REGION, TYPE=VOLUME
elset_A

Only one ∗CO-SIMULATION REGION option can be defined in each Abaqus/Standard analysis. In addition, only one element set can be defined.

Identifying the fields exchanged across a co-simulation interface

For Abaqus/Standard to Abaqus/Standard co-simulation, see the tables in "Identifying the fields exchanged across a co-simulation interface" in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, for lists of fields that are available for co-simulation exchange.

Defining the rendezvousing scheme

The SIMULIA Co-Simulation Engine configuration file is used to define the time incrementation process and the frequency of exchange between the two Abaqus/Standard analyses.

Predefined templates are available for commonly used coupling schemes. You can refer to these templates when you create your configuration files. This section describes the rendezvous scheme settings and the predefined configuration file templates.

Time incrementation scheme

You can force the two transient Abaqus/Standard analyses to use the same increment size, or you can allow the increment sizes to differ (subcycling). The time incrementation scheme that you choose for coupling affects the solution computational cost and accuracy but not the solution stability. When using the subcycling method, this data exchange does not represent a constraint on Abaqus/Standard

incrementation; the Abaqus/Standard analysis advances in time using its normal time incrementation logic but performs data exchanges as needed at the coupling step size intervals.

A time-harmonic electromagnetic procedure is defined in the frequency domain and does not have a solution time-scale associated with it in the sense that a transient analysis does. It is convenient to introduce a pseudo-solution time scale that is associated with the time-harmonic electromagnetic procedure involved in a co-simulation analysis, thereby facilitating coupling with a transient analysis at certain solution time intervals of the latter analysis. The pseudo-time scale of the time-harmonic electromagnetic analysis follows the solution time scale in the transient heat transfer analysis and is reset in every coupling step in a manner described below.

Defining the coupling scheme

The sequential explicit coupling scheme (also referred to as the Gauss-Seidel coupling algorithm) and the iterative coupling scheme are available for electromagnetic-to-structural and electromagnetic-to-thermal co-simulation. The electromagnetic analysis must always lead the co-simulation, while the heat transfer or the stress analysis always lags the co-simulation. All the predefined templates are set up with the above lead-lag sequence.

Coupling step size

The coupling step size is the period between two consecutive co-simulation data exchanges between the two Abaqus/Standard analyses. For transient electromagnetic to transient heat transfer or transient implicit dynamic co-simulation, the coupling step size can be specified to be equal to the minimum of the time step sizes determined by the automatic time incrementation schemes of the individual analyses or to a constant user-defined value.

When the leading electromagnetic analysis is time harmonic, the coupling step size can be specified to be equal to the time step size of the lagging transient heat transfer or implicit dynamic analysis or to a constant user-defined value. In the latter case, the time-harmonic electromagnetic analysis would solve for the fields at the end of each successive constant user-defined coupling step size, while the lagging heat transfer or stress analysis would typically subcycle until the target coupling step time is reached.

For iterative coupling, the two analyses must be coupled at the end of each time increment, and subcycling should not be used. If subcycling is used in this situation, the exchanged updated solutions during the iterations will be utilized only for the very last increment and the cumulative effect of the updates over the previous increments (between the last coupling and the current coupling) will be lost.

Creating a configuration file

You can use predefined templates to create the configuration file for the coupling schemes described above. Table 17.3.3–1 describes the predefined templates available for electromagnetic to transient heat transfer analyses and for electromagnetic to stress-displacement analyses and lists example configuration files that you can review.

Table 17.3.3–1 Templates for electromagnetic co-simulation.

Electromagnetic to transient heat transfer co-simulation	Coupling scheme: • Electromagnetic analysis leads • Heat transfer analysis defines the coupling step size
	template_em_std_export
	Example file: **exa_em_std_export**
	Coupling scheme: • Electromagnetic analysis leads • Allow heat transfer analysis to subcycle
	template_em_std_fixed
	Example file: **exa_em_std_fixed**
	Coupling scheme: • Electromagnetic analysis leads • Heat transfer analysis defines the coupling step size • Iterative coupling
	template_em_std_iterative
	Example file: **exa_em_std_iterative**
Electromagnetic to Abaqus/Standard stress/displacement co-simulation	Coupling scheme: • Electromagnetic analysis leads • Step size is determined based on the minimum suggested step size of the electromagnetic and Abaqus/Standard stress/displacement analyses. • Either analysis can subcycle • Body forces are transferred from the electromagnetic to Abaqus/Standard stress/displacement analysis. No other co-simulation transfer occurs.
	template_em_std_force_oneway
	Example file: **exa_em_std_force_oneway**

To obtain an example configuration file, you can use the **abaqus fetch** utility. For example, to obtain the example for which the heat transfer analysis serves as the master in determining the coupling step size, use the following command:

```
abaqus fetch job=exa_em_std_export
```

The example file **exa_em_std_export.xml** is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<CoupledMultiphysicsSimulation>
   <template_em_std_export>
      <EM_Job>em_job_name</EM_Job>
      <HeatTransfer_Job>ht_job_name</HeatTransfer_Job>
      <duration>duration_value</duration>
   </template_em_std_export>
</CoupledMultiphysicsSimulation>
```

In certain cases you may need to use co-simulation configuration features that are not described in the predefined templates. For example, you may wish to change the dissimilar mesh mapping search tolerances; these tolerances are available generally in the configuration file but are not described in the predefined templates. For these cases, you must create an elaborated configuration file; for more information, see "Using elaborated configuration files" in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

Executing the coupled analysis

You execute the co-simulation from the command line, as described in "Executing a co-simulation," Section 17.3.4.

Limitations

The limitations discussed in "Preparing an Abaqus analysis for co-simulation," Section 17.2.1, apply to electromagnetic-to-structural and electromagnetic-to-thermal co-simulation.

17.3.4 EXECUTING A CO-SIMULATION

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CFD Abaqus/CAE

References

- "Abaqus/Standard, Abaqus/Explicit, Abaqus/CFD, AND FMU co-simulation execution," Section 3.2.4
- "Understanding co-executions," Section 19.4 of the Abaqus/CAE User's Guide

Overview

You can execute the following types of co-simulations from the command line:

- Structural-to-structural
- Fluid-to-structural
- Conjugate heat transfer
- Electromagnetic-to-structural
- Electromagnetic-to-thermal

You can also execute structural-to-structural, fluid-to-structural, and conjugate heat transfer co-simulations in Abaqus/CAE.

Executing a co-simulation from Abaqus/CAE

You can execute the coupled analysis interactively in Abaqus/CAE as described in "Understanding co-executions," Section 19.4 of the Abaqus/CAE User's Guide. You are not required to create a configuration file; Abaqus/CAE creates the file automatically.

Abaqus/CAE Usage: Job module:
Co-execution→Create: select the Abaqus/Standard model and the Abaqus/Explicit model; **Communication time out**: *timeout-value*
Co-execution→Manager: **Submit**

Executing a co-simulation from the command line

You execute the Abaqus jobs as described in "Abaqus/Standard, Abaqus/Explicit, Abaqus/CFD, AND FMU co-simulation execution," Section 3.2.4.

Command usage example

Use the following command to submit a co-simulation between two Abaqus analyses, "job-1" and "job-2":

```
abaqus cosimulation cosimjob=beam job=job-1,job-2
configure=config
```

Considerations for using the timeout parameter

The **timeout** execution parameter specifies the amount of time in seconds that each analysis waits to receive the co-simulation message expected from the other analysis that is running. The default timeout value is 60 minutes when submitting jobs using the command line options and 10 minutes when executing the jobs in Abaqus/CAE. When the timeout period is large compared to typical analysis increment wallclock times, you have greater flexibility in starting jobs and performing operations that precede the co-simulation analysis step. Examples where this flexibility is needed include: job submission using queues, analyses where steps that precede the co-simulation step have long run times, and cases where one job is resubmitted because of an input error. However, a large timeout period can cause problems when one of the co-simulation jobs fails (for reasons such as convergence issues or availability of computer resources) before the initial co-simulation communication is established. In these cases you may prefer to terminate the job left running rather than have it wait the entire timeout period.

Limitations

Electromagnetic-to-structural and electromagnetic-to-thermal co-simulations are not supported in Abaqus/CAE.

17.4 Co-simulation using Abaqus and discrete models

- "Structural-to-logical co-simulation," Section 17.4.1

17.4.1 STRUCTURAL-TO-LOGICAL CO-SIMULATION

Products: Abaqus/Standard Abaqus/Explicit

References

- "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2
- "Dymola model execution," Section 3.2.5
- "Co-simulation: overview," Section 17.1.1
- "Preparing an Abaqus analysis for co-simulation," Section 17.2.1
- *CO-SIMULATION
- *CO-SIMULATION CONTROLS
- *CO-SIMULATION REGION

Overview

This section discusses analysis setup, execution, and limitation details for structural-to-logical co-simulation using Abaqus and Dymola. To use co-simulation as described in this section, you will need Dymola version 7.2 or later, with the **`.DLL`** option. For more information, see "User's and Installation Guides for Logical-Physical modeling using Abaqus and Dymola" in the Dassault Systèmes Knowledge Base at www.3ds.com/support/knowledge-base.

Preparing an Abaqus model for a structural-to-logical co-simulation

To prepare an Abaqus model for a structural-to-logical co-simulation, you need to:

- identify the Abaqus analysis step involved,
- identify the interface sensors and actuators to be exchanged, and
- define the rendezvousing scheme.

The following sections briefly describe these steps. For more detailed information on co-simulation analysis model creation, see "Preparing an Abaqus analysis for co-simulation," Section 17.2.1.

Identifying the Abaqus analysis step

The period of time of co-simulation interaction is defined as a co-simulation event. This event begins at the start of an Abaqus analysis step (the co-simulation step) and ends by the end of that step. Any procedure type in the analysis that supports the co-simulation technique can be a co-simulation step; however, only one co-simulation step per analysis job is allowed. For multiple co-simulation steps, you can use the restart capability to define new co-simulation steps in subsequent analysis jobs.

You can define a co-simulation step in which Abaqus exchanges fields with Dymola at run time. You specify co-simulation controls that define the rendezvousing scheme.

Input File Usage: Use the following options within a step definition:

∗CO-SIMULATION, NAME=*co-simulation name*,
PROGRAM=MULTIPHYSICS,
CONTROLS=*name*
∗CO-SIMULATION CONTROLS, NAME=*name*

Defining sensors and actuators

You must define sensors and actuators. Any nodal or connector element history output variable can be defined as a named sensor. In addition, some of the whole surface contact and contact pair output history variables can be defined as a named sensor. See "Sensor definition in Abaqus/Standard and Abaqus/Explicit" in "Output to the output database," Section 4.1.3, for more details.

Input File Usage: Use the following options to define a sensor:

∗OUTPUT, HISTORY, FREQUENCY, SENSOR, NAME=*sensor_name*
∗NODE OUTPUT, NSET=*NSET_Name*
nodal output variable

Use the following option to define an actuator:

∗AMPLITUDE, NAME=*name*, DEFINITION=ACTUATOR

There are no data lines associated with this amplitude definition to define the amplitude value at a given simulation time. Instead, the amplitude value at a given simulation time will be imported from the Dymola output. The user-specified name can be referenced in any of the Abaqus options that can refer to an amplitude (such as ∗CLOAD, ∗BOUNDARY, ∗FIELD, and ∗CHANGE FRICTION).

For example,

∗CLOAD, AMPLITUDE=*name*
101, 3, 1.0

Identifying the sensors and actuators to be exchanged

You must specify the fields that are imported and/or exported in the Abaqus analysis during the co-simulation.

The import definition ensures that the values of the Dymola outputs as computed at the time of the last data exchange are the Abaqus inputs (actuators) and are given as the current values of the specified amplitude names. Equally important, the export definition ensures that the current values of the Abaqus outputs (sensors) are given as the values of the Dymola inputs at the time of the data exchange.

You can specify up to 5000 actuators/sensors.

Input File Usage: Use the following options to identify the sensors and actuators to be exchanged:

∗CO-SIMULATION REGION, TYPE=DISCRETE, IMPORT
Actuator_name1, Actuator_name2, etc.

∗CO-SIMULATION REGION, TYPE=DISCRETE, EXPORT
Sensor_name1, Sensor_name2, etc.

The user-defined names on the data lines are comma-separated with eight entries per line. Each name must be less than 80 characters long. Use more than one data line if necessary.

Defining the rendezvousing scheme: target time

The coupling step is the period between two consecutive exchanges and, consequently, defines the exchange frequency between Abaqus and Dymola. The coupling step size is established at the beginning of each coupling step and is used to compute the target time (the time when the next synchronized exchange occurs). In most cases, exchanging data every Abaqus increment is sufficient.

Determining the target time

During a coupled simulation between Abaqus and Dymola, the current coupling step size is negotiated by both solvers to establish a suitable coupling step size. Each solver "suggests" a coupling step size. The smaller of these suggested step sizes is then selected. Abaqus can suggest either a constant or a variable coupling step size. For a variable coupling step size, Abaqus sets the suggested coupling step size equal to the next suggested increment size computed by the automatic time incrementation scheme. Using a variable coupling step size typically leads to the most accurate solution.

Abaqus and Dymola exchange the coupling step size and establish a "negotiated" coupling step size based on the smaller of the solvers' suggested values. This coupling step size is then used to advance the target time for the next coupling rendezvous.

Input File Usage: Use the following option to have Abaqus suggest a constant coupling step size:

∗CO-SIMULATION CONTROLS, STEP SIZE=*coupling_step_size*

Use the following option to have Abaqus suggest a variable coupling step size:

∗CO-SIMULATION CONTROLS (omit the STEP SIZE parameter)

How Abaqus meets the target time

In addition to defining the coupling step size, you define co-simulation controls to describe how Abaqus advances to its next target time. By default, Abaqus may perform several time increments during the coupling step period, referred to as "subcycling." Subcycling allows Abaqus to cut back the current increment size and take smaller increments to resolve nonlinear events.

Alternatively, you can force Abaqus to use an increment size that is equal to the negotiated coupling step size, referred to as "lockstep." When proceeding in a lockstep manner, Abaqus will not be able to reduce the time increment to resolve nonlinear events. Consequently, Abaqus terminates the simulation in cases where the nonlinear events require that the increment size be reduced.

Input File Usage: Use the following option to have Abaqus suggest a constant coupling step size:

∗CO-SIMULATION CONTROLS, TIME INCREMENTATION=LOCKSTEP

Use the following option to have Abaqus suggest a variable coupling step size:

*CO-SIMULATION CONTROLS,
TIME INCREMENTATION=SUBCYCLE

Reaching target times

You can control how accurately Abaqus meets the target time. The Abaqus target times can be reached in an "exact" or "loose" manner. By default, Abaqus reaches the target times in an exact manner; that is, Abaqus adjusts the time increment as necessary to exactly meet the target time. Alternatively, you can direct Abaqus to meet the target times in an approximate or loose manner, performing a co-simulation exchange only after a target time is passed, with time incrementation in Abaqus advancing without regard for the target time. In certain cases, such as when Abaqus takes many more increments than Dymola, the loose approach may be more cost effective.

Input File Usage: Use the following option to reach target times in an exact manner:

*CO-SIMULATION CONTROLS, TIME MARKS=YES (default)

Use the following option to reach target times in a loose manner:

*CO-SIMULATION CONTROLS, TIME MARKS=NO

Preparing the Dymola model for the coupled simulation

To prepare your Dymola model for coupled simulation, you need to:

- define named inputs and outputs,
- translate the model with the **`.DLL`** option to produce **`dymosim.dll`**,
- define co-simulation parameters, and
- map the signal (sensors and actuators) names in Abaqus to signal names in Dymola.

The following sections briefly describe these steps; they are not intended to describe how to use Dymola. For further information on Dymola and its user interface, refer to the relevant user's manuals. After these steps are complete, you must have the following files in the directory from which the Dymola analysis will be run:

- **`libdsdll.dll`** corresponding to the Dymola installation,
- **`dymosim.dll`** corresponding to the desired Dymola model, and
- a map (**`.sgn`**) file whose contents are described below.

Defining named inputs and outputs

The Dymola graphical editor provides the most convenient way to define inputs and outputs. From the Modelica library palette, drag **Modelica.Blocks.Interfaces.RealInput** blocks onto the canvas to define inputs (blue-filled triangles). Use **Modelica.Blocks.Interfaces.RealOutput** blocks to define outputs (hollow triangles). You can change the names assigned by default via popup dialog boxes. It is important that you use all uppercase letters when specifying the names of the input and output blocks.

Translating the model with the `.DLL` option

In the **Simulation** panel, select **Simulation→Setup** and then click the **Compiler** tab. Toggle on **Export model as DLL with API**, and click **OK**. In the **Simulation** panel, select **Simulation→Translate**. A file named **`dymosim.dll`** is created in the current working folder. You should pay particular attention to the warning/error messages issued during translation. A successful co-simulation analysis can be conducted only if the translation was successful. The **`dymosim.dll`** file includes information related to this specific model. If co-simulation with other Dymola models is necessary, make sure to change the current working directory so that the **`dymosim.dll`** file is not overwritten.

Defining co-simulation parameters

While the **`dymosim.dll`** file includes information about the Dymola model itself, it does not include information about the duration of the analysis, the frequency of data exchanges, the required output frequency, and the solver type to be used for the time integration in Dymola. You need to provide this information in a separate text file, *mapFileName*`.sgn`, using the following format:

> **`Analysis:`** *analysisTime couplingStepSize OutputPointsPerCouplingStep DymolaSolverType*

Guidelines for providing the information are as follows:

- **`Analysis`** must be followed immediately by a colon (no spaces). A single blank space follows the colon after which the four quantities are given separated by single blank spaces.
- *analysisTime* must match the analysis time specified in the Abaqus co-simulation step.
- *couplingStepSize* must match the coupling step size specified in the Abaqus model, if it was specified. The most convenient way to set up the co-simulation controls is to use the default subcycling method and to have Abaqus suggest a variable coupling step size. Then you can specify *couplingStepSize* as being equal to *analysisTime* to allow the co-simulation to exchange data every Abaqus increment.
- *OutputPointsPerCouplingStep* sets the frequency of output generation in Dymola. A value of 1 is normally sufficient.
- Refer to the Dymola User's Manual when choosing a solver type. Typically, solverType 8 performs the best, although it is slightly more expensive than other solvers. The CPU cost of the co-simulation analysis is typically dominated by the Abaqus analysis and, hence, the performance of the Dymola solver is of little consequence to the performance of the co-simulation analysis.

In most cases, specifying the following information in the map (**`.sgn`**) file is sufficient:

```
Analysis: analysisTime analysisTime 1 8
```

Mapping signal names

It is quite likely that in common practice the Dymola and the Abaqus models are prepared by different engineers in the organization as the finite element analyst may not have any controls expertise and vice versa. It is reasonable to assume that the names used for inputs in Dymola would not match the names

used for Abaqus sensors and/or the names for outputs in Dymola would not match the names for Abaqus actuators. In such cases, you can modify the names used in one of the analysis programs. If that is not possible, the map (**.sgn**) file can be used to map signal names to be exchanged from Abaqus to Dymola. You can add the following optional lines immediately after the **Analysis** line described above:

Actuators:
AbaqusActuatorName1 DymolaOutputName1
AbaqusActuatorName2 DymolaOutputName2
Etc.
Sensors:
AbaqusSensorsName1 DymolaInputName1
AbaqusSensorsName1 DymolaInputName1
Etc.

Guidelines for providing the information are as follows:

- **Actuators** is followed by a list of name pairs to match Abaqus actuators to Dymola outputs. Use as many lines as needed (up to 5000).
- Similarly, **Sensors** is followed by a list of name pairs to match Abaqus sensors to Dymola inputs. Use as many lines as needed (up to 5000).

Executing the coupled analysis

You execute the Abaqus job as described in "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2. You execute the Dymola job as described in "Dymola model execution," Section 3.2.5. Abaqus and Dymola can be run on heterogeneous and remote platforms that are located on the same network domain. For co-simulation purposes, Dymola has to be run on a Windows 32-bit or 64-bit system while Abaqus can be run on any supported platform available. The communication between Abaqus and Dymola is via Transmission Control Protocol (TCP) sockets. To start a coupled simulation between Abaqus and Dymola, one of the solvers must initiate the communication process, while the other solver must connect to the initiated communication process. Either Abaqus or Dymola can initiate the communication process; however, the starting method differs for each solver. Each method is discussed in the following sections.

You will need to know the network host ID for the machine on which the communication process was started. This ID can either be a host name or an Internet Protocol (IP) address. The port designation allows different applications to maintain their own channels of communication. Set the port number to any available port on your system. Port numbers in the range of 0 to 65535 can be assigned. In general, port numbers under 1024 are reserved by the operating system and should be avoided. You can run multiple analysis jobs simultaneously on a given system by assigning each analysis a unique port number.

The Abaqus and Dymola-related input files do not have to be located in the same directory. If the two co-simulation analyses are run on different platforms that are not binary compatible (such as a Linux 64-bit system for the Abaqus job and a 32-bit Windows system for the Dymola job), then ASCII data

exchange must be enabled. To accomplish that, add the following two lines in your local environment file for machines used for the co-simulation analysis:

```
import os
os.environ['ABAQUS_DCI_FORMAT']='ASCII'
```

See "Using the Abaqus environment files," Section 4.1 of the Abaqus Installation and Licensing Guide, for details about the environment files.

Abaqus initiates the communication process

To have Abaqus initiate the communication process, you supply only the port number (and not the host name) on the command line; for example, the Abaqus job can be run using the following command:

abaqus job=*job_name* **port=***portNumber*

When Abaqus initiates the communication process, Dymola must connect to the host and the specified port where the Abaqus analysis was started. The following command can be used for the Dymola job:

abaqus dymola input=*mapFileName* **host=***machineName* **port=***portNumber*

Dymola initiates the communication process

Alternatively, the following two commands can be used to have Dymola initiate the communication process:

abaqus job=*job_name* **host=***machineName* **port=***portNumber*

abaqus dymola input=*mapFileName* **port=***portNumber*

Limitations

Structural-to-logical co-simulation using Abaqus and Dymola is not intended for applications in which the mechanical/thermal system itself is split between the two analysis programs. For example, results obtained for a coupled analysis in which the tires and the road are modeled in Abaqus while the suspension and the body are modeled in Dymola will likely lead to unreliable results.

18. Extending Abaqus Analysis Functionality

User subroutines and utilities 18.1

18.1 User subroutines and utilities

- "User subroutines: overview," Section 18.1.1
- "Available user subroutines," Section 18.1.2
- "Available utility routines," Section 18.1.3

18.1.1 USER SUBROUTINES: OVERVIEW

References

- "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2
- Abaqus User Subroutines Reference Guide

Overview

User subroutines:

- are provided to increase the functionality of several Abaqus capabilities for which the usual data input methods alone may be too restrictive;
- provide an extremely powerful and flexible tool for analysis;
- are written as C, C++, or Fortran code and must be included in a model when you execute the analysis, as discussed below;
- must be included and, if desired, can be revised in a restarted run, since they are not saved to the restart files (see "Restarting an analysis," Section 9.1.1);
- cannot be called one from another; and
- can in some cases call utility routines that are also available in Abaqus (see "Available utility routines," Section 18.1.3).

Including user subroutines in a model

You can include one or more user subroutines in a model by specifying the name of a C, C++, or Fortran source or precompiled object file that contains the subroutines. Details are provided in "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2.

Input File Usage: Enter the following input on the command line:

abaqus job=*job-name* **user**={*source-file* | *object-file*}

Abaqus/CAE Usage: Job module: job editor: **General**: **User subroutine file**

Managing external databases in Abaqus and exchanging information with other software

In Abaqus it is sometimes desirable to set up the runtime environment and manage interactions with external data files or parallel processes that are used in conjunction with user subroutines. For example, there may be history-dependent quantities to be computed externally, once per increment, for use during the analysis; or output quantities that are accumulated over multiple elements in COMMON block variables within user subroutines may need to be written to external files at the end of a converged increment for postprocessing. Such operations can be performed with user subroutine **`UEXTERNALDB`** in Abaqus/Standard and **`VEXTERNALDB`** in Abaqus/Explicit. This user interface can potentially be used to exchange data with another code, allowing for "stagger" between Abaqus and another code.

Writing a user subroutine

User subroutines should be written with great care. To ensure their successful implementation, the rules and guidelines below should be followed. For a detailed discussion of the individual subroutines, including coding interfaces and requirements, refer to the Abaqus User Subroutines Reference Guide.

Required `INCLUDEs`

Every user subroutine written in Fortran must include one of the following statements as the first statement after the argument list:

- Abaqus/Standard:

```
include 'aba_param.inc'
```

- Abaqus/Explicit:

```
include 'vaba_param.inc'
```

If variables are exchanged between the main user subroutine and subsequent subroutines, you should specify the above include statement in all the subroutines to preserve precision.

Every C and C++, user subroutine must include the statement

```
#include <aba_for_c.h>
```

This file contains macros for the Fortran-to-C interface interoperability.

The files **aba_param.inc**, **vaba_param.inc**, and **aba_for_c.h** are installed on the system by the Abaqus installation procedure and contain important installation parameters. These statements tell the Abaqus execution procedure, which compiles and links the user subroutine with the rest of Abaqus, to include the **aba_param.inc** or **vaba_param.inc** file automatically. It is not necessary to find the file and copy it to any particular directory; Abaqus will know where to find it.

Naming convention

If user subroutines call other subroutines or use COMMON blocks to pass information, such subroutines or COMMON blocks should begin with the letter K since this letter is never used to start the name of any subroutine or COMMON block in Abaqus.

User subroutines written in C or C++ will be called from Fortran; therefore, they must conform to the Fortran calling conventions: the name of a C or C++ subroutine must be wrapped in a **FOR_NAME** macro; for example,

```
extern "C" void FOR_NAME(film)(...) {...  }
```

and the arguments must be passed by reference.

Redefining variables

User subroutines must perform their intended function without overwriting other parts of Abaqus. In particular, you should redefine only those variables identified in this chapter as "variables to be defined." Redefining "variables passed in for information" will have unpredictable effects.

Compilation and linking problems

If problems are encountered during compilation or linking of the subroutine, make sure that the Abaqus environment file (the default location for this file is the **`site`** subdirectory of the Abaqus installation) contains the correct compile and link commands as specified in the Abaqus Installation and Licensing Guide. These commands should have been set up by the Abaqus site manager during installation. The number and type of arguments must correspond to what is specified in the documentation. Mismatches in type or number of arguments may lead to platform-dependent linking or runtime errors.

Memory allocation considerations

Your user subroutine will share memory resources with Abaqus. When you need to use large arrays or other large data structures, you should allocate their memory dynamically, so that memory is allocated from the heap and not the stack. Failure to dynamically allocate large arrays may result in stack overflow errors and an abort of your Abaqus analysis. For an example of dynamic allocation using native Fortran allocatable arrays, refer to "Creation of a data file to facilitate the postprocessing of elbow element results: FELBOW," Section 15.1.6 of the Abaqus Example Problems Guide. Abaqus also provides another, more convenient way for users to allocate their own storage (see "Allocatable arrays," Section 2.1.23 of the Abaqus User Subroutines Reference Guide).

Testing and debugging

When developing user subroutines, test them thoroughly on smaller examples in which the user subroutine is the only complicated aspect of the model before attempting to use them in production analysis work.

If needed, debug output can be written to the Abaqus/Standard message (**`.msg`**) file using Fortran unit 7 or to the Abaqus/Standard data (**`.dat`**) file or the Abaqus/Explicit log (**`.log`**) file using Fortran unit 6; these units should not be opened by your routines since they are already opened by Abaqus.

Fortran units 15 through 18 or units greater than 100 can be used to read or write other user-specified information. The use of other Fortran units may interfere with Abaqus file operations; see "Fortran unit numbers used by Abaqus," Section 3.7.1. You must open these Fortran units; and because of the use of scratch directories, the full pathname for the file must be used in the OPEN statement.

Environment variable **`ABA_PARALLEL_DEBUG`** can be set to turn on verbosity, to display compilation and linking commands, and to enable debugging of user subroutines.

Terminating an analysis

Utility routine **`XIT`** (Abaqus/Standard) or **`XPLB_EXIT`** (Abaqus/Explicit) should be used instead of **`STOP`** when terminating an analysis from within a user subroutine. This will ensure that all files

associated with the analysis are closed properly (“Terminating an analysis,” Section 2.1.15 of the Abaqus User Subroutines Reference Guide).

Models defined in terms of an assembly of part instances

An Abaqus model can be defined in terms of an assembly of part instances (see “Defining an assembly,” Section 2.10.1).

Reference coordinate system

Although a local coordinate system can be defined for each part instance, all variables (such as current coordinates) are passed to a user subroutine in the global coordinate system, not in a part-local coordinate system. The only exception to this rule is when the user subroutine interface specifically indicates that a variable is in a user-defined local coordinate system (“Orientations,” Section 2.2.5, or “Transformed coordinate systems,” Section 2.1.5). The local coordinate system originally may have been defined relative to a part coordinate system, but it was transformed according to the positioning data given for the part instance. As a result, a new local coordinate system was created relative to the assembly (global) coordinate system. This new coordinate system definition is the one used for local orientations in user subroutines.

Node and element numbers

The node and element numbers passed to a user subroutine are internal numbers generated by Abaqus. These numbers are global in nature; all internal node and element numbers are unique. If the original number and the part instance name are required, call the utility subroutine **GETPARTINFO** (Abaqus/Standard) or **VGETPARTINFO** (Abaqus/Explicit) from within your user subroutine (see “Obtaining part information,” Section 2.1.5 of the Abaqus User Subroutines Reference Guide). The expense of calling these routines is not trivial, so minimal use of them is recommended.

Another utility subroutine, **GETINTERNAL** (Abaqus/Standard) or **VGETINTERNAL** (Abaqus/Explicit), can be used to retrieve the internal node or element number corresponding to a given part instance name and local number.

Set and surface names

Set and surface names passed to user subroutines are always prefixed by the assembly and part instance names, separated by underscores. For example, a surface named **surf1** belonging to part instance **Part1-1** in assembly **Assembly1** will be passed to a user subroutine as

```
Assembly1_Part1-1_surf1
```

Solution-dependent state variables

Solution-dependent state variables are values that can be defined to evolve with the solution of an analysis.

Defining and updating

Any number of solution-dependent state variables can be used in the following user subroutines:

- **CREEP**
- **FRIC**
- **HETVAL**
- **UANISOHYPER_INV**
- **UANISOHYPER_STRAIN**
- **UEL**
- **UEXPAN**
- **UGENS**
- **UHARD**
- **UHYPER**
- **UINTER**
- **UMAT**
- **UMATHT**
- **UMULLINS**
- **USDFLD**
- **UTRS**
- **VFABRIC**
- **VFRIC**
- **VFRICTION**
- **VUANISOHYPER_INV**
- **VUANISOHYPER_STRAIN**
- **VUFLUIDEXCH**
- **VUHARD**
- **VUINTER**
- **VUINTERACTION**
- **VUMAT**
- **VUMULLINS**
- **VUSDFLD**
- **VUTRS**

- **VUVISCOSITY**
- **VWAVE**

The state variables can be defined as a function of any other variables appearing in these subroutines and can be updated accordingly. Solution-dependent state variables should not be confused with field variables, which may also be needed in the constitutive routines and can vary with time; field variables are discussed in detail in "Predefined fields," Section 34.6.1.

Solution-dependent state variables used in **VFRIC**, **VUINTER**, **VFRICTION**, and **VUINTERACTION** are defined as state variables at slave nodes and are updated with other contact variables.

Allocating space

You must allocate space for each of the solution-dependent state variables at every applicable integration point or contact slave node.

Separate user subroutine groups have been identified that differ in the way the number of solution-dependent state variables is defined. These groups are described below. Solution-dependent state variables can be shared by subroutines within the same group; they cannot be shared between subroutines belonging to different groups.

Input File Usage: For most subroutines the number of such variables required at the points or nodes is entered as the only value on the data line of the *DEPVAR option, which should be included as part of the material definition for every material in which solution-dependent state variables are to be considered:

*DEPVAR

For subroutines that do not use the material behavior defined with the *MATERIAL option, the *DEPVAR option is not used.

For subroutine **UEL**:

*USER ELEMENT, VARIABLES=*number of variables*

For subroutine **UGENS**:

*SHELL GENERAL SECTION, USER, VARIABLES=*number of variables*

For subroutines **FRIC** and **VFRIC**:

*FRICTION, USER, DEPVAR=*number of variables*

For subroutines **UINTER** and **VUINTER**:

*SURFACE INTERACTION, USER, DEPVAR=*number of variables*

For subroutine **VFRICTION**:

*FRICTION, USER=FRICTION, DEPVAR=*number of variables*

For subroutine **VUFLUIDEXCH**:

*FLUID EXCHANGE PROPERTY, TYPE=USER,
DEPVAR=*number of variables*

For subroutine **VUINTERACTION**:

*SURFACE INTERACTION, USER=INTERACTION, DEPVAR=*number of variables*

For subroutine **VWAVE**:

*WAVE, TYPE=USER, DEPVAR=*number of variables*

Abaqus/CAE Usage: For most subroutines the number of such variables required at the points or nodes is entered as part of the material definition for every material in which solution-dependent state variables are to be considered:

Property module: material editor: **General→Depvar**: **Number of solution-dependent state variables**

Defining initial values

You can define the initial values of solution-dependent state variable fields directly or in Abaqus/Standard through a user subroutine. The initial values of solution-dependent state variables for contact or for user subroutine **VWAVE** in Abaqus/Explicit are assigned as zero internally.

Defining initial values directly

You can define the initial values in a tabular format for elements and/or element sets. See "Initial conditions in Abaqus/Standard and Abaqus/Explicit," Section 34.2.1, for additional details.

Input File Usage: *INITIAL CONDITIONS, TYPE=SOLUTION

Defining initial values in a user subroutine in Abaqus/Standard

For complicated cases in Abaqus/Standard you can call user subroutine **SDVINI** so that dependencies on coordinates, element numbers, etc. can be used in the definition of the variable field.

Input File Usage: *INITIAL CONDITIONS, TYPE=SOLUTION, USER

Element deletion controlled by solution-dependent state variables

If element deletion controlled by state variables is defined in an analysis (see "User-defined mechanical material behavior," Section 26.7.1), the value of the state variable that is controlling the deletion flag can be modified by any of the user subroutines in which state variables are used, provided that the user subroutine is called at a material point.

Output

User-defined, solution-dependent state variables can be written to the data (**.dat**) file, the output database (**.odb**) file, and the results (**.fil**) file; the output identifiers SDV and SDV*n* are available as element integration variables (see "Abaqus/Standard output variable identifiers," Section 4.2.1, and "Abaqus/Explicit output variable identifiers," Section 4.2.2). Output of these variables is not available for user subroutines **VFRIC**, **VUINTER**, **VFRICTION**, **VUINTERACTION**, and **VWAVE**.

Alphanumeric data

Alphanumeric data, such as labels (names) of surfaces or materials, are always passed into user subroutines in the upper case. As a result, direct comparison of these labels with corresponding lower-case characters will fail. Upper case must be used for all such comparisons. An example of such a comparison can be found in "UMAT," Section 1.1.44 of the Abaqus User Subroutines Reference Guide. It illustrates the code setup inside user subroutine **UMAT** when more than one user-defined material model needs to be defined. The variable **CMNAME** is compared against **MAT1** and **MAT2** (even in situations where the material names may have been defined as **mat1** and **mat2**, respectively.)

Precision in Abaqus/Explicit

Abaqus/Explicit is installed with both single precision and double precision executables. To use the double precision executable, you must specify double precision when you run the analysis (see "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2). All variables in the user subroutines that start with the letters a to h and o to z will automatically be defined in the precision of the executable that you run. The precision of the executable is defined in the **vaba_param.inc** file, and it is not necessary to define the precision of the variables explicitly.

Vectorization in Abaqus/Explicit

Abaqus/Explicit user subroutines are written with a vector interface, which means that blocks of data are passed to the user subroutines. For example, the vectorized user material routines (**VFABRIC** and **VUMAT**) are passed stresses, strains, state variables, etc. for **nblock** material points. One of the parameters defined by **vaba_param.inc** is **maxblk**, the maximum block size. If the user subroutine requires the dimensioning of temporary arrays, they can be dimensioned by **maxblk**.

Parallelization

User subroutines can be used when running jobs in parallel. In thread-parallel mode access to common blocks, common files, and other shared resources need to be guarded against race conditions. Special utility routines are provided for that purpose. For details, see "Ensuring thread safety," Section 2.1.22 of the Abaqus User Subroutines Reference Guide. An environment variable **ABA_PARALLEL_DEBUG** can be set to increase verbosity and help troubleshoot problems should they arise in parallel runs.

User subroutine calls

Most of the user subroutines available in Abaqus are called at least once for each increment during an analysis step. However, as discussed below, many subroutines are called more or less often.

Subroutines that define material, element, or interface behavior

Most user subroutines that are used to define material, element, or interface behavior are called twice per material point, element, or slave surface node in the first iteration of every increment such that the model's initial stiffness matrix can be formulated appropriately for the step procedure chosen. The subroutines

are called only once per material point, element, or slave surface node in each succeeding iteration within the increment.

By default, in transient implicit dynamic analyses ("Implicit dynamic analysis using direct integration," Section 6.3.2) Abaqus/Standard calculates accelerations at the beginning of each dynamic step. Abaqus/Standard must call user subroutines that are used to define material, element, or interface behavior two extra times for each material point, element, or slave surface node prior to the zero increment. The extra calls to the user subroutines are not made if the initial acceleration calculations are suppressed. If the half-increment residual tolerances are being checked in a transient implicit dynamic step, Abaqus/Standard must call these user subroutines (except **UVARM**) one extra time for each material point, element, or slave surface node at the end of each increment. If the calculation of the half-increment residual is suppressed, the extra call to the user subroutines is not made.

User subroutines **UHARD**, **UHYPEL**, **UHYPER**, and **UMULLINS**, when used in plane stress analyses, are called more often.

Subroutines that define initial conditions or orientations

User subroutines that are used to define initial conditions or orientations are called before the first iteration of the first step's initial increment within an analysis.

Subroutines that define predefined fields

User subroutines that are used to define predefined fields are called prior to the first iteration of the relevant step's first increment for all iterations of all increments whenever the current field variable is needed.

Verification of subroutine calls

If there is any doubt as to how often a user subroutine is called, this information can be obtained upon testing the subroutine on a small example, as suggested earlier. The current step and increment numbers are commonly passed into these subroutines, and they can be printed out as debug output (also discussed earlier). The iteration number for which the subroutine is called may not be passed into the user subroutine; however, if printed output is sent from the subroutine to the message (**.msg**) file ("Output," Section 4.1.1), the location of the output within this file will give the iteration number, provided that the output to the message file is written at every increment.

Utility routines

A variety of utility routines are available to assist in the coding of user subroutines. You include the utility routine inside a user subroutine. When called, the utility routine will perform a predefined function or action whose output or results can be integrated into the user subroutine. Some utility routines are only applicable to particular user subroutines. Each utility routine is discussed in detail in "Utility routines," Section 2.1 of the Abaqus User Subroutines Reference Guide.

Variables provided for use in utility routines

The following utility routines require the use of Abaqus-provided variables passed into the user subroutines from which they are called:

- **GETNODETOELEMCONN**
- **GETVRM**
- **GETVRMAVGATNODE**
- **GETVRN**
- **IGETSENSORID**
- **IVGETSENSORID**
- **MATERIAL_LIB_MECH**
- **MATERIAL_LIB_HT**

These variables will be defined properly when passed into your user subroutine; you cannot modify the variables or create alternative variables for use in the utility routines.

For example, the **GETVRM** utility routine requires the variable **JMAC**, which is passed from Abaqus/Standard into user subroutine **UVARM** and other user subroutines for which **GETVRM** is a supported utility. The variable **JMAC** represents an Abaqus data structure that requires no further manipulation on your part. If you use the **GETVRM** utility routine from within user subroutine **UVARM**, you will pass the **JMAC** variable from **UVARM** into **GETVRM**.

18.1.2 AVAILABLE USER SUBROUTINES

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CFD Abaqus/Aqua

References

- "User subroutines: overview," Section 18.1.1
- Abaqus User Subroutines Reference Guide

Overview

User subroutines allow advanced users to customize a wide variety of Abaqus capabilities. Information on writing user subroutines and detailed descriptions of each subroutine appear online in the Abaqus User Subroutines Reference Guide. A listing and explanations of associated utility routines also appear in that guide.

Available user subroutines for Abaqus/Standard

The available user subroutines for Abaqus/Standard are as follows:

- **CREEP**: Define time-dependent, viscoplastic behavior (creep and swelling).
- **DFLOW**: Define nonuniform pore fluid velocity in a consolidation analysis.
- **DFLUX**: Define nonuniform distributed flux in a heat transfer or mass diffusion analysis.
- **DISP**: Specify prescribed boundary conditions.
- **DLOAD**: Specify nonuniform distributed loads.
- **FILM**: Define nonuniform film coefficient and associated sink temperatures for heat transfer analysis.
- **FLOW**: Define nonuniform seepage coefficient and associated sink pore pressure for consolidation analysis.
- **FRIC**: Define frictional behavior for contact surfaces.
- **FRIC_COEF**: Define frictional coefficient for contact surfaces.
- **GAPCON**: Define conductance between contact surfaces or nodes in a fully coupled temperature-displacement analysis, a fully coupled thermal-electrical-structural analysis, or a pure heat transfer analysis.
- **GAPELECTR**: Define electrical conductance between surfaces in a coupled thermal-electric analysis or a fully coupled thermal-electrical-structural analysis.
- **HARDINI**: Define initial equivalent plastic strain and initial backstress tensor.
- **HETVAL**: Provide internal heat generation in heat transfer analysis.
- **MPC**: Define multi-point constraints.

- **ORIENT**: Provide an orientation for defining local material directions or local directions for kinematic coupling constraints or local rigid body directions for inertia relief.
- **RSURFU**: Define a rigid surface.
- **SDVINI**: Define initial solution-dependent state variable fields.
- **SIGINI**: Define an initial stress field.
- **UAMP**: Specify amplitudes.
- **UANISOHYPER_INV**: Define anisotropic hyperelastic material behavior using the invariant formulation.
- **UANISOHYPER_STRAIN**: Define anisotropic hyperelastic material behavior based on Green strain.
- **UCORR**: Define cross-correlation properties for random response loading.
- **UCREEPNETWORK**: Define time-dependent behavior (creep) for models defined within the parallel rheological framework.
- **UDECURRENT**: Define nonuniform volume current density in an eddy current or magnetostatic analysis.
- **UDEMPOTENTIAL**: Define nonuniform magnetic vector potential on a surface in an eddy current or magnetostatic analysis.
- **UDMGINI**: Define the damage initiation criterion.
- **UDSECURRENT**: Define nonuniform surface current density in an eddy current or magnetostatic analysis.
- **UEL**: Define an element.
- **UELMAT**: Define an element with access to Abaqus materials.
- **UEXPAN**: Define incremental thermal strains.
- **UEXTERNALDB**: Manage user-defined external databases and calculate model-independent history information.
- **UFIELD**: Specify predefined field variables.
- **UFLUID**: Define fluid density and fluid compliance for hydrostatic fluid elements.
- **UFLUIDCONNECTORLOSS**: Define the connector loss for fluid pipe connector elements.
- **UFLUIDCONNECTORVALVE**: Define the valve opening to control flow in fluid pipe connector elements.
- **UFLUIDLEAKOFF**: Define the fluid leak-off coefficients for pore pressure cohesive elements.
- **UFLUIDPIPEFRICTION**: Define the friction coefficient for fluid pipe elements.
- **UGENS**: Define the mechanical behavior of a shell section.
- **UHARD**: Define the yield surface size and hardening parameters for isotropic plasticity or combined hardening models.
- **UHYPEL**: Define a hypoelastic stress-strain relation.
- **UHYPER**: Define a hyperelastic material.
- **UINTER**: Define surface interaction behavior for contact surfaces.

- **UMASFL**: Specify prescribed mass flow rate conditions for a convection/diffusion heat transfer analysis.
- **UMAT**: Define a material's mechanical behavior.
- **UMATHT**: Define a material's thermal behavior.
- **UMESHMOTION**: Specify mesh motion constraints during adaptive meshing.
- **UMOTION**: Specify motions during cavity radiation heat transfer analysis or steady-state transport analysis.
- **UMULLINS**: Define damage variable for the Mullins effect material model.
- **UPOREP**: Define initial fluid pore pressure.
- **UPRESS**: Specify prescribed equivalent pressure stress conditions.
- **UPSD**: Define the frequency dependence for random response loading.
- **URDFIL**: Read the results file.
- **USDFLD**: Redefine field variables at a material point.
- **UTEMP**: Specify prescribed temperatures.
- **UTRACLOAD**: Specify nonuniform traction loads.
- **UTRS**: Define a reduced time shift function for a viscoelastic material.
- **UTRSNETWORK**: Define a reduced time shift function for models defined within the parallel rheological framework.
- **UVARM**: Generate element output.
- **UWAVE**: Define wave kinematics for an Abaqus/Aqua analysis.
- **UXFEMNONLOCALWEIGHT**: Define the weight function used to compute the average stress/strain to determine the crack propagation direction.
- **VOIDRI**: Define initial void ratios.

Available user subroutines for Abaqus/Explicit

The available user subroutines for Abaqus/Explicit are as follows:

- **VDFLUX**: User subroutine to specify nonuniform distributed fluxes in an explicit dynamic coupled temperature-displacement analysis.
- **VDISP**: Specify prescribed boundary conditions.
- **VDLOAD**: Specify nonuniform distributed loads.
- **VEXTERNALDB**: Control analyses to exchange data among Abaqus user subroutines and external programs or files.
- **VFABRIC**: Define fabric material behavior.
- **VFRIC**: Define contact frictional behavior between surfaces defined with the contact pair algorithm.
- **VFRIC_COEF**: Define contact frictional coefficient between surfaces defined with the general contact algorithm.

- **VFRICTION**: Define contact frictional behavior between surfaces defined with the general contact algorithm.
- **VUAMP**: Specify amplitudes.
- **VUANISOHYPER_INV**: Define anisotropic hyperelastic material behavior using the invariant formulation.
- **VUANISOHYPER_STRAIN**: Define anisotropic hyperelastic material behavior based on Green strain.
- **VUCHARLENGTH**: Define characteristic element length at a material point.
- **VUCREEPNETWORK**: Define time-dependent behavior (creep) for models defined within the parallel rheological framework.
- **VUEL**: Define an element.
- **VUEOS**: Define equation of a state material model.
- **VUFIELD**: Specify predefined field variables.
- **VUFLUIDEXCH**: Define mass/heat energy flow rates for fluid exchange.
- **VUFLUIDEXCHEFFAREA**: Define effective area for fluid exchange.
- **VUHARD**: Define the yield surface size and hardening parameters for isotropic plasticity or combined hardening models.
- **VUINTER**: Define the contact interaction between surfaces defined with the contact pair algorithm.
- **VUINTERACTION**: Define the contact interaction between surfaces defined with the general contact algorithm.
- **VUMAT**: Define material behavior.
- **VUMULLINS**: Define damage variable for the Mullins effect material model.
- **VUSDFLD**: Redefine field variables at a material point.
- **VUTRS**: Define a reduced time shift function for a viscoelastic material.
- **VUVISCOSITY**: Define the shear viscosity for equation of state models.
- **VWAVE**: Define wave kinematics for an Abaqus/Aqua analysis.

Available user subroutines for Abaqus/CFD

The available user subroutines for Abaqus/CFD are as follows:

- **SMACfdUserPressureBC**: Specify prescribed pressure boundary conditions.
- **SMACfdUserVelocityBC**: Specify prescribed velocity boundary conditions.

18.1.3 AVAILABLE UTILITY ROUTINES

Products: Abaqus/Standard Abaqus/Explicit Abaqus/Aqua

References

- "User subroutines: overview," Section 18.1.1
- "Utility routines," Section 2.1 of the Abaqus User Subroutines Reference Guide

Overview

A variety of utility routines are available to assist in the coding of user subroutines. When called, the utility routine will perform a predefined function or action whose output or results can be integrated into the user subroutine.

Available utility routines

The following utility routines are available for use in coding user subroutines in Abaqus:

- **GETENVVAR** or **VGETENVVAR** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to obtain the value of an environment variable ("Obtaining Abaqus environment variables," Section 2.1.1 of the Abaqus User Subroutines Reference Guide).
- **GETJOBNAME** or **VGETJOBNAME** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to obtain the name of the current analysis job ("Obtaining the Abaqus job name," Section 2.1.2 of the Abaqus User Subroutines Reference Guide).
- **GETOUTDIR** or **VGETOUTDIR** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to obtain the name of the directory where analysis job output is being placed ("Obtaining the Abaqus output directory name," Section 2.1.3 of the Abaqus User Subroutines Reference Guide).
- **GETNUMCPUS** can be called from any Abaqus/Standard user subroutine to obtain the number of MPI processes; **VGETNUMCPUS** can be called from any Abaqus/Explicit user subroutine in a domain-parallel run to obtain the number of processes used for the parallel run ("Obtaining parallel processes information," Section 2.1.4 of the Abaqus User Subroutines Reference Guide).
- **GETRANK** can be called from any Abaqus/Standard user subroutine to obtain the rank of the MPI process from which the function is called; **VGETRANK** can be called from any Abaqus/Explicit user subroutine in a domain-parallel run to obtain the individual process rank ("Obtaining parallel processes information," Section 2.1.4 of the Abaqus User Subroutines Reference Guide).
- **get_thread_id** can be called from any Abaqus user subroutine to retrieve the ID that Abaqus assigned to that thread ("Obtaining parallel processes information," Section 2.1.4 of the Abaqus User Subroutines Reference Guide).
- **GETPARTINFO** or **VGETPARTINFO** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to retrieve the part instance name and local node or element number

corresponding to an internal node or element number. **GETINTERNAL** or **VGETINTERNAL** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to retrieve the internal node or element number corresponding to a given part instance name and local number ("Obtaining part information," Section 2.1.5 of the Abaqus User Subroutines Reference Guide.)

- **GETVRM** provides access to material point information for Abaqus/Standard user subroutines **UVARM**, **UDMGINI**, or **USDFLD** ("Obtaining material point information in an Abaqus/Standard analysis," Section 2.1.6 of the Abaqus User Subroutines Reference Guide).
- **VGETVRM** provides access to selected output variables at material points for Abaqus/Explicit user subroutine **VUSDFLD** ("Obtaining material point information in an Abaqus/Explicit analysis," Section 2.1.7 of the Abaqus User Subroutines Reference Guide).
- **GETVRMAVGATNODE** provides access to material point information, extrapolated to and averaged at a node, for Abaqus/Standard user subroutine **UMESHMOTION** ("Obtaining material point information averaged at a node," Section 2.1.8 of the Abaqus User Subroutines Reference Guide).
- **GETVRN** provides access to node point information for Abaqus/Standard user subroutine **UMESHMOTION** ("Obtaining node point information," Section 2.1.9 of the Abaqus User Subroutines Reference Guide).
- **GETNODETOELEMCONN** can be called from user subroutine **UMESHMOTION** to retrieve a list of elements connected to a specific node. This element list can then be used with utility routine **GETVRMAVGATNODE** ("Obtaining node to element connectivity," Section 2.1.10 of the Abaqus User Subroutines Reference Guide).
- **SINV** determines the first and second stress invariants for a given stress tensor in Abaqus/Standard ("Obtaining stress invariants, principal stress/strain values and directions, and rotating tensors in an Abaqus/Standard analysis," Section 2.1.11 of the Abaqus User Subroutines Reference Guide).
- **SPRINC** or **VSPRINC** determines the principal values for a given stress or strain tensor in Abaqus/Standard or Abaqus/Explicit, respectively ("Obtaining stress invariants, principal stress/strain values and directions, and rotating tensors in an Abaqus/Standard analysis," Section 2.1.11 of the Abaqus User Subroutines Reference Guide, and "Obtaining principal stress/strain values and directions in an Abaqus/Explicit analysis," Section 2.1.12 of the Abaqus User Subroutines Reference Guide).
- **SPRIND** or **VSPRIND** determines both the principal values and principal directions for a given stress or strain tensor in Abaqus/Standard or Abaqus/Explicit, respectively ("Obtaining stress invariants, principal stress/strain values and directions, and rotating tensors in an Abaqus/Standard analysis," Section 2.1.11 of the Abaqus User Subroutines Reference Guide, and "Obtaining principal stress/strain values and directions in an Abaqus/Explicit analysis," Section 2.1.12 of the Abaqus User Subroutines Reference Guide).
- **ROTSIG** can be called from Abaqus/Standard user subroutine **UMAT** to perform the rotation of tensors when large-strain calculations are performed ("Obtaining stress invariants, principal stress/strain values and directions, and rotating tensors in an Abaqus/Standard analysis," Section 2.1.11 of the Abaqus User Subroutines Reference Guide).

- **GETWAVE** determines wave kinematic data associated with the applied wave theory in an Abaqus/Aqua analysis (“Obtaining wave kinematic data in an Abaqus/Aqua analysis,” Section 2.1.13 of the Abaqus User Subroutines Reference Guide).
- **GETWAVEVEL**, **GETWINDVEL**, and **GETCURRVEL** are used to obtain the wave, wind, and steady current velocity components, respectively, for a given point in an Abaqus/Aqua analysis (“Obtaining wave kinematic data in an Abaqus/Aqua analysis,” Section 2.1.13 of the Abaqus User Subroutines Reference Guide).
- **STDB_ABQERR** or **XPLB_ABQERR** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to print an informational, warning, or error message to the message file in Abaqus/Standard or the status file in Abaqus/Explicit (“Printing messages to the message or status file,” Section 2.1.14 of the Abaqus User Subroutines Reference Guide).
- **XIT** or **XPLB_EXIT** can be called from any Abaqus/Standard or Abaqus/Explicit user subroutine, respectively, to terminate an analysis (“Terminating an analysis,” Section 2.1.15 of the Abaqus User Subroutines Reference Guide).
- **IGETSENSORID** or **IVGETSENSORID** can be called from Abaqus/Standard user subroutine **UAMP** or Abaqus/Explicit user subroutine **VUAMP**, respectively, to obtain the ID of a user-defined sensor. **GETSENSORVALUE** or **VGETSENSORVALUE** can be called from Abaqus/Standard user subroutine **UAMP** or Abaqus/Explicit user subroutine **VUAMP**, respectively, to obtain the value of a user-defined sensor (“Obtaining sensor information,” Section 2.1.16 of the Abaqus User Subroutines Reference Guide).
- **MATERIAL_LIB_MECH** can be called from Abaqus/Standard user subroutine **UELMAT** to access the Abaqus material library (“Accessing Abaqus materials,” Section 2.1.17 of the Abaqus User Subroutines Reference Guide).
- **MATERIAL_LIB_HT** can be called from Abaqus/Standard user subroutine **UELMAT** to access the Abaqus thermal material library (“Accessing Abaqus thermal materials,” Section 2.1.18 of the Abaqus User Subroutines Reference Guide).
- **SMACfdUserSubroutineGetScalar** can be called from any Abaqus/CFD user subroutine to access selected output variables for elements or surface facets that are part of a boundary condition definition (“Obtaining scalar state information in an Abaqus/CFD analysis,” Section 2.1.19 of the Abaqus User Subroutines Reference Guide).
- **SMACfdUserSubroutineGetVector** can be called from any Abaqus/CFD user subroutine to access selected output variables for elements and surface facets that are part of a boundary condition definition (“Obtaining vector state information in an Abaqus/CFD analysis,” Section 2.1.20 of the Abaqus User Subroutines Reference Guide).
- **SMACfdUserSubroutineGetMpiComm** can be called from within any Abaqus/CFD user subroutine to obtain the MPI communicator used in a parallel analysis job (“Obtaining the MPI communicator in an Abaqus/CFD analysis,” Section 2.1.21 of the Abaqus User Subroutines Reference Guide).
- **MutexInit**, **MutexLock**, and **MutexUnlock** can be used to create and manipulate mutexes (“Ensuring thread safety,” Section 2.1.22 of the Abaqus User Subroutines Reference Guide).

- Dynamically allocatable arrays can be used to accumulate and store data. All basic types are supported. In addition, Abaqus provides arrays of type **`Real`** that change precision automatically ("Allocatable arrays," Section 2.1.23 of the Abaqus User Subroutines Reference Guide).
  - Real arrays will run in single precision when Abaqus/Explicit runs in single precision and will run in double precision when Abaqus/Explicit runs in double precision.
  - Arrays of structures can be used to store data of any user-defined type, including Fortran and C/C++ structures and classes with arbitrary sizes and number of fields.
- Thread-local arrays can be used to allocate storage local to a thread. **`SMALocalIntArrayCreate`** and **`SMALocalFloatArrayCreate`** can be used to create or resize a local array. **`SMALocalIntArrayAccess`** and **`SMALocalFloatArrayAccess`** can be used to locate an existing local array. **`SMALocalIntArrayDelete`** and **`SMALocalFloatArrayDelete`** can be used to delete a local array. **`SMALocalIntArraySize`** and **`SMALocalFloatArraySize`** can be used to get the size of the array ("Allocatable arrays," Section 2.1.23 of the Abaqus User Subroutines Reference Guide).
- Global arrays can also be used to allocate storage shared among all threads. **`SMAIntArrayCreate`** and **`SMAFloatArrayCreate`** can be used to create or resize a global array. **`SMAIntArrayAccess`** and **`SMAFloatArrayAccess`** can be used to locate an existing global array. **`SMAIntArrayDelete`** and **`SMAFloatArrayDelete`** can be used to delete a global array. **`SMAIntArraySize`** and **`SMAFloatArraySize`** can be used to get the size of the global array ("Allocatable arrays," Section 2.1.23 of the Abaqus User Subroutines Reference Guide).

19. Design Sensitivity Analysis

Design sensitivity analysis 19.1

19.1 Design sensitivity analysis

- "Design sensitivity analysis," Section 19.1.1

19.1.1 DESIGN SENSITIVITY ANALYSIS

Product: Abaqus/Design

References

- "Parametric input," Section 1.4.1
- "Parametric shape variation," Section 2.1.2
- *STEP
- *DESIGN PARAMETER
- *DESIGN RESPONSE

Overview

Design sensitivity analysis (DSA):

- is performed with Abaqus/Design, an add-on option for Abaqus/Standard;
- provides the sensitivities of responses with respect to specified design parameters;
- is available for static stress and frequency analysis using models that have only stress/displacement elements; and
- can include design parameters affecting: material properties (elastic, hyperelastic, and hyperfoam models); section properties; concentrated forces and moments; and nodal coordinates (and beam and shell normals if applicable).

Design sensitivity analysis

The design sensitivity analysis (DSA) capability provides the derivatives of certain output variables with respect to specified design parameters. These derivatives are commonly referred to as *sensitivities*, because they provide a first-order measure of how sensitive the output variable is to a change in the design parameter. The output variables for which sensitivities are computed are called *design responses* or simply *responses*. Design parameters are chosen from a set of existing analysis parameters. As an example, you can choose to obtain the derivatives of stresses with respect to Young's modulus; stress is the response, and Young's modulus is the design parameter. The sensitivities are computed based on the direct differentiation method used in conjunction with the semi-analytical computational technique. In the semi-analytical technique some derivatives are computed using numerical (finite) differencing, thus requiring perturbations of the design parameters. For these derivatives by default Abaqus/Design will use a central differencing scheme and automatically determine appropriate perturbation sizes based on a heuristic algorithm. You can override these defaults by specifying the numerical differencing method and the perturbation sizes directly. A full discussion of DSA theory is given in "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide.

Activating DSA

You activate DSA on a step-by-step basis.

Input File Usage: Use the following option to activate DSA in a particular step:

```
*STEP, DSA=YES
```

Activating DSA in multiple steps

Once DSA is activated in a general step, it remains active in all subsequent general steps until it is deactivated in a subsequent general step. Once DSA is activated in a perturbation step, it remains active in all subsequent consecutive perturbation steps until it is deactivated in a subsequent consecutive perturbation step. However, if DSA is activated in a step whose procedure is not supported for DSA, DSA will be deactivated until it is activated again.

Input File Usage: Use the following option to deactivate DSA in a particular step:

```
*STEP, DSA=NO
```

Specifying design parameters

You can define multiple parameters to be used in place of Abaqus input quantities for an analysis. You must indicate which of these parameters are to be considered as design parameters.

Input File Usage: Use the following option to define analysis parameters:

```
*PARAMETER
par1=x
par2=y
. . .
```

Use the following option to specify the design parameters:

```
*DESIGN PARAMETER
par1, par2, . . .
```

Restrictions on design parameters

The following are restrictions on design parameters:

- Design parameters can be associated only with floating point data. The following analysis components can include design-dependent data:
  - Beam sections integrated during analysis (“Using a beam section integrated during the analysis to define the section behavior,” Section 29.3.6)
  - Concentrated loads (“Concentrated loads,” Section 34.4.2)
  - Elastic materials (“Linear elastic behavior,” Section 22.2.1)
  - Friction (“Frictional behavior,” Section 37.1.5)
  - Gasket sections (“Gasket elements: overview,” Section 32.6.1)
  - Hyperelastic materials (“Hyperelastic behavior of rubberlike materials,” Section 22.5.1)

  - Hyperfoam materials (“Hyperelastic behavior in elastomeric foams,” Section 22.5.2)
  - Membrane sections (“Membrane elements,” Section 29.1.1)
  - Local orientations (“Orientations,” Section 2.2.5)
  - Shell sections integrated during analysis (“Using a shell section integrated during the analysis to define the section behavior,” Section 29.6.5)
  - Solid sections (“Solid (continuum) elements,” Section 28.1.1)
  - Transverse shear stiffnesses (“Choosing a beam element,” Section 29.3.3, or “Shell section behavior,” Section 29.6.4)
- Shape design parameters (i.e., design parameters that affect nodal coordinates and beam and/or shell normals) can be used only in conjunction with parametric shape variations (see “Parametric shape variation,” Section 2.1.2).
- Design parameters must be mutually independent.
- Design parameters cannot be tabularly dependent (see “Parametric input,” Section 1.4.1).

Specifying responses

Response requests are specified using a syntax analogous to that for specifying output requests to the output database. Except for eigenvalues and eigenfrequencies, there are no default responses—if no responses are requested, no response sensitivities will be output. If DSA is active in a frequency step, eigenvalue and eigenfrequency sensitivities will be output automatically. Specifying a response will cause output of both the response and the response sensitivities.

Input File Usage: Use the following options to request design responses:

∗DESIGN RESPONSE, FREQUENCY=*interval*, MODE LIST
∗CONTACT RESPONSE, MASTER=*master name*, NSET=*nset name*, SLAVE=*slave name*
∗ELEMENT RESPONSE, ELSET=*elset name*
∗NODE RESPONSE, NSET=*nset name*

Requesting responses in multiple steps

Unless respecified, response requests defined in a step propagate to subsequent steps according to the following rules:

1. Requests in general steps propagate to subsequent general steps.
2. Requests in linear perturbation steps propagate to subsequent consecutive linear perturbation steps.
3. When a non-DSA step appears between DSA steps, the responses must be respecified in the DSA step following the non-DSA step.

Restrictions on responses

The available responses are a subset of the existing output variables. The valid responses based on procedure type are described below.

- For static steps the valid responses are:
  - Node responses: U and RF
  - Element responses: S, SF, SINV, SP, E, SE, EP, EE, EEP, LE, LEP, NE, NEP, ENER, ELEN, EVOL, and MASS
  - Contact responses: CSTRESS and CDISP
- For frequency steps the valid responses are:
  - Node responses: None
  - Element responses: MASS
  - Contact responses: None
  - Eigenvalue (EIGVAL) and eigenfrequency (EIGFREQ) sensitivities are output automatically.

Specifying design gradients of design-dependent input data

The DSA calculations require the gradients of the design-dependent input data with respect to the design parameters. For example, if Poisson's ratio, ν, is made dependent on a design parameter, say h, the gradient $d\nu/dh$ is required. Design gradients with respect to shape design parameters are specified differently than those with respect to other design parameters.

Specifying design gradients with respect to shape design parameters

Gradients with respect to shape design parameters must be specified using a parametric shape variation definition (see "Parametric shape variation," Section 2.1.2). For the purposes of DSA if the parameter to which the shape variation data refer is a design parameter, the shape variation data are interpreted as the gradients of the nodal coordinates with respect to the design parameter. If a nonzero value is given for the shape parameter, Abaqus/Design will also perturb the base coordinates.

Input File Usage: Use the following option to specify the design gradients for shape design parameters:

∗PARAMETER SHAPE VARIATION, PARAMETER=*design parameter*

Specifying gradients for non-shape design parameters

For non-shape design parameters, by default Abaqus/Design will use numerical differentiation to calculate design gradients based on the information you provide. However, you can override this default behavior by specifying the gradients directly using Python expressions (see "Parametric input," Section 1.4.1). You specify a design parameter as the independent parameter and a list of the parameters that depend on that design parameter. Only one independent (design) parameter can be given for each design gradient definition.

Input File Usage: Use the following option to specify the design gradients for non-shape design parameters:

∗DESIGN GRADIENT, INDEPENDENT=*design parameter*,
DEPENDENT=*(list of dependent parameters)*

History dependence and formulation type in a multi-increment analysis

Both total and incremental formulations are implemented for DSA. The choice of formulation depends on whether or not an analysis is history dependent. Below is a brief description of these formulation types. A more detailed discussion can be found in "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide. By default, the incremental DSA formulation is used. You can specify the DSA formulation only for the entire model; this specification is ignored if given as part of a step definition.

Incremental DSA formulation

In the incremental formulation the problem is assumed to be history dependent. Abaqus/Design solves for the incremental displacement sensitivities, and the total displacement sensitivity is updated at the end of the increment. Due to the history dependence, the incremental displacement sensitivities for the current increment depend on the sensitivities of the state variables at the beginning of the increment, in the same sense that incremental displacements depend on the state variables at the beginning of the increment for equilibrium analyses. Thus, Abaqus/Design must also compute and update state variable sensitivities in each increment. Consequently, DSA must be activated for *all* steps up to the last step in which DSA is active, and the DSA calculations will be done at *all* increments in these steps, regardless of whether or not a design response is requested for a given step. If a response is requested for a step, the specified response frequency is ignored for the purposes of the DSA calculations (the frequency at which the output is written will still be governed by the specified response frequency).

The disadvantage of the incremental DSA formulation is its cost, due to the necessity of computing both state variable and incremental displacement sensitivities at each increment prior to the last DSA increment. This increased cost is unavoidable if the problem is history dependent but is unnecessary if the problem is history independent. Thus, the total DSA formulation should be chosen for problems that are not history dependent.

Input File Usage: *DSA CONTROLS, FORMULATION=INCREMENTAL

Total DSA formulation

In the total displacement formulation the total displacement sensitivities are calculated directly based on the assumption that the problem is not history dependent. In other words, the displacement sensitivities do not depend on sensitivity results calculated in previous increments. Thus, the advantage of the total formulation is that the sensitivity calculations need only be done at increments of interest. You can control when DSA calculations are done by activating DSA for only the desired steps and specifying the desired frequency for each design response request.

You may choose to use the total DSA formulation in problems that are known to be history dependent. However, in this case the DSA solution is approximate, with the degree of approximation increasing as the problem becomes more strongly history dependent. To assess the validity of using the total DSA formulation, it is recommended that you run both an incremental and total sensitivity analysis for a typical problem and compare the results.

Input File Usage: *DSA CONTROLS, FORMULATION=TOTAL

DSA in linear perturbation steps

The sensitivity of the perturbation response can be calculated in a linear perturbation step (see "General and linear perturbation procedures," Section 6.1.3). The perturbation response will include the effects of stress and load stiffening in the base state if geometric nonlinearity is considered. Since we need to calculate the sensitivity of an incremental (perturbation) response, the sensitivity of the stress and load stiffening effects must be known at the end of the base step. Thus, if geometric nonlinearity is considered in the base step, DSA must also be active in the base step, irrespective of the type of formulation (total or incremental).

Determination of design parameter perturbation sizes

The basis of the semi-analytic technique is the use of numerical differencing to obtain derivatives of certain element vectors and matrices (see "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide). Abaqus/Design will automatically determine appropriate perturbation sizes to be used in the semi-analytic technique unless you specify them directly. Abaqus/Design determines the perturbation sizes using a heuristic perturbation sizing algorithm based on the behavior of a scalar s associated with an element. By default, the perturbation sizing algorithm is applied only for the first increment (static procedure) or first mode (frequency procedure) in each step for which DSA is active. The perturbation sizes are then reused for the remaining increments or modes in the step for which DSA calculations are done.

The goal of the algorithm is to find perturbation sizes that are optimal for numerical differencing. Differencing formulas are based on Taylor series expansions, and the order of approximation of the derivative to be computed is reflected in the terms that are neglected in the series. The accuracy of the approximated derivatives often depends strongly on the perturbation size used in the differencing formula. Choosing a perturbation size that is too large will cause a truncation error, which occurs when the order of approximation is no longer valid (i.e., as a result of truncating higher-order terms in the Taylor series). A perturbation size that is too small will lead to inaccuracies in the differencing operations due to round-off, typically referred to as a cancellation error.

The algorithm attempts to find perturbation sizes giving the best compromise between cancellation and truncation errors by observing the behavior of s. For each design parameter s is computed for perturbation sizes spanning several orders of magnitude. The error in s between consecutive perturbation sizes is calculated as $e_i = |s_i - s_{i-1}|/s_i$. The perturbation size yielding an acceptable error, e_a, is chosen as the best perturbation size.

This scalar s is selected as follows:

- *Static procedure.* For static steps s is chosen as the norm of the element pseudoload (the partial derivative of the element residual with respect to the design parameters).
- *Frequency procedure.* For frequency steps s is computed from the element contribution to a matrix involving the derivatives of the mass and stiffness matrices (namely $\left(\frac{DK^{NM}}{Dh} - \lambda\frac{DM^{NM}}{Dh}\right)$, where K^{NM} is the stiffness, M^{NM} is the mass, h is the design parameter, and λ is an eigenvalue). The scalar s is taken as the projection of this matrix onto an eigenvector $\bar{\phi}^N$. If the perturbation sizing algorithm is applied to a mode with a distinct eigenvalue, $\bar{\phi}^N$ is taken as the eigenvector associated

with this mode. However, if a mode happens to be associated with a repeated eigenvalue, $\bar{\phi}^N$ is taken as the sum of all the eigenvectors associated with the repeated eigenvalue. Thus, the entire set of modes associated with a repeated eigenvalue will be treated simultaneously by the perturbation sizing algorithm (the eigenvalue sensitivities of a repeated eigenvalue are obtained simultaneously from the same reduced eigenvalue system).

See "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide, for further details on the selection of s.

Controlling the numerical differencing behavior

You can control various aspects of the numerical differencing operations. These aspects are described in detail in the following sections. You can specify DSA controls for the entire model and/or for individual steps. Specifying these controls for the entire model has the effect of creating new default values for the various settings. When you specify these controls for individual steps, the following propagation rules are enforced:

- Once DSA controls are specified in a non-perturbation step, they remain in effect for all subsequent non-perturbation steps, unless they are respecified or reset.
- Once DSA controls are specified in a perturbation step, they remain in effect for all subsequent consecutive perturbation steps, unless they are respecified or reset.

Resetting DSA controls

You can reset DSA controls only for individual steps. If DSA controls are specified for the entire model, resetting them in a particular step will reset the numerical differencing behavior to the behavior specified for the entire model; otherwise, the behavior will be reset to the original default values. Any additional changes specified will be applied after the behavior is reset.

Input File Usage: Use the following option to reset the DSA controls for a particular step:

*DSA CONTROLS, RESET

Changing the defaults for the heuristic perturbation sizing algorithm

The following two sections describe how certain parameters associated with the perturbation sizing algorithm can be changed from their default values for purposes of computational efficiency and accuracy.

Changing the default tolerance

By default, the tolerance e_a is set to 1.0×10^{-4}. Warning messages are written to the message file for elements for which this tolerance is not achieved. These elements are collected in element sets and can be viewed in the Visualization module of Abaqus/CAE. It is important to understand that this tolerance controls the effort expended in obtaining an optimum perturbation size; it is not a direct measure of the accuracy of the numerical differentiation.

Input File Usage: Use the following option to override the default tolerance:

*DSA CONTROLS, TOLERANCE=*tolerance*

Changing the frequency at which the perturbation sizing algorithm is used

Determining perturbation sizes using the heuristic algorithm is computationally intensive. You can specify the frequency at which the perturbation sizes are recalculated. For example, specifying a sizing frequency of *n* will cause Abaqus/Design to determine new perturbation sizes at every *n* increments or eigenmodes. The perturbation size will always be recalculated at the first increment or eigenmode in each step for which DSA is active, which is equivalent to specifiying a sizing frequency of 0. Since the perturbation sizing algorithm is computationally intensive, care should be exercised to ensure that the sizing frequency is as large as possible (or zero).

As discussed above, the perturbation sizing algorithm is applied simultaneously to all modes associated with a repeated eigenvalue. Thus, the actual number of modes associated with a repeated eigenvalue that are "hit" based on the sizing frequency is irrelevant, so long as it is at least one.

Input File Usage: Use the following option to specify the frequency at which the perturbation sizes are recalculated:

∗DSA CONTROLS, SIZING FREQUENCY=*frequency*

Overriding the default heuristic perturbation sizing algorithm

If an appropriate perturbation size is already known for a particular design parameter (from previous analyses of similar problems, for example), economy can be gained by applying this perturbation size directly rather than having Abaqus/Design automatically find the perturbation size. You can specify either forward differencing or central differencing directly together with an absolute perturbation size for each design parameter. If you override the default algorithm, it is up to you to choose perturbation sizes that will lead to accurate sensitivities.

Input File Usage: Use the following option to override the default heuristic perturbation sizing algorithm for a given design parameter:

∗DSA CONTROLS
design parameter, FD (forward differencing) or CD (central differencing), *absolute perturbation size*

For example, to specify an absolute perturbation size of 0.001 and forward differencing for design parameter **despar** use the following input:

```
*DSA CONTROLS
despar, FD, 0.001
```

This data line is specified for each design parameter for which the default scheme is to be overridden.

Accuracy of the DSA solution

As can be seen in "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide, the accuracy of the DSA solution is dictated by both the accuracy of the numerically computed derivatives and, for nonlinear static analysis, the accuracy of the tangent stiffness matrix. The accuracy of the numerically computed derivatives is governed by the semi-analytic DSA algorithm; you can control it by specifying

DSA controls. In nonlinear static analysis DSA uses the tangent stiffness matrix formed during the last equilibrium iteration. It is possible that the accuracy of the tangent stiffness matrix needed to achieve an accurate equilibrium solution may be insufficient to achieve an accurate DSA solution. In such cases you can tighten the convergence tolerances during the equilibrium analysis so that a more accurate tangent stiffness matrix is obtained (see "Commonly used control parameters," Section 7.2.2). Furthermore, an accurate equilibrium solution often can be obtained when unsymmetric terms in the tangent stiffness are ignored (i.e., the unsymmetric matrix storage and solution scheme is not used; see "Defining an analysis," Section 6.1.2). However, even if mildly unsymmetric stiffness terms are neglected, the DSA solution may be inaccurate. Therefore, it is recommended that the unsymmetric solution scheme be used for DSA when the tangent stiffness matrix is known to be unsymmetric.

In some cases a response at a certain instant in time may be discontinuous with respect to a design parameter. For example, at this point of discontinuity a variation in the design parameter may cause a node to come into contact, frictional behavior to change from sticking to sliding, or a material point to transition from elastic to inelastic behavior. Since the DSA calculations make use of numerical differencing, it is possible that the perturbation of the design parameter used in the differencing scheme may result in values of the response to be differenced that lie on opposite sides of the discontinuity. If this occurs, the accuracy of the computed derivative cannot be guaranteed. Mathematically speaking, the derivative (sensitivity) of the response with respect to the design parameter does not exist at the point of discontinuity. Practically speaking, it is unlikely that the response at any given instant will lie precisely on the discontinuity. In cases where the response is near a discontinuity, if you choose to use the default perturbation sizing algorithm, the algorithm will attempt to choose design parameter perturbation sizes such that the values of the perturbed responses remain on the same side of the discontinuity. In addition, for contact elements DSA calculations are not performed in increments in which the associated contact node is open. Typically, the global results in any increment are not affected by a few discontinuous points in the model.

Design dependence and supported features

Responses depend on design parameters *explicitly* and *implicitly*. Implicit design dependence is the dependence on the design parameter through the solution variables; therefore, this type of dependence can be quantified only after the DSA solution is obtained (recall that the DSA solution is the total displacement sensitivity for the total formulation and the incremental displacement sensitivity for the incremental formulation). All other design dependencies are explicit, meaning that they can be resolved without knowing the DSA solution. The types of dependencies can be identified by looking at the form of the sensitivity of a response, say r, with respect to a design parameter, say h. This sensitivity is expressed as

$$\frac{dr}{dh} = \frac{\partial r}{\partial h} + \frac{\partial r}{\partial u^N}\frac{du^N}{dh}$$

for the total formulation and

$$\frac{dr}{dh} = \frac{\partial r}{\partial h} + \frac{\partial r}{\partial \boldsymbol{\alpha}^t}\frac{d\boldsymbol{\alpha}^t}{dh} + \frac{\partial r}{\partial \Delta u^N}\frac{d\Delta u^N}{dh}$$

for the incremental formulation, where u^N is a displacement degree of freedom and $\boldsymbol{\alpha}^t$ represents state variables at the beginning of the increment (see "Design sensitivity analysis," Section 2.18.1 of the Abaqus Theory Guide, for further details). The state variables include the displacements at the beginning of the increment. In both cases the last term on the right-hand side represents the implicit design dependence through the solution variables.

It is observed from the incremental equation above that the explicit design dependence consists of two terms. The first of these, $\frac{\partial r}{\partial h}$, represents a *direct design dependence*, because this term arises from the direct dependence of the response on the design parameter. The second explicit term, $\frac{\partial r}{\partial \boldsymbol{\alpha}^t}\frac{d\boldsymbol{\alpha}^t}{dh}$, represents the dependence on the design parameter through the state variables at the beginning of the increment. For the total formulation, it is seen that the explicit term involves only direct design dependence.

Any feature for which direct design dependence calculations are implemented in Abaqus will be referred to as *supported* for DSA. Supported and unsupported features can be mixed in an analysis, unless the supported features cause unsupported features to become directly design dependent (an example of this would be making the Young's modulus for a frame element design dependent, since frames are not supported for DSA).

To make a clearer distinction between the types of design dependencies, consider the more concrete example of a linear elastic truss element, fixed at one end and pulled with a concentrated load at the other end. Let $u = u^t + \Delta u$ represent the displacement at the free end, $\boldsymbol{E}$ represent Young's modulus, and $\boldsymbol{L}$ represent the length of the truss. Consider the axial stress S_{11} as the response. Although it is clear in this simple example that the stress can be computed easily as the load divided by the cross-sectional area, the finite element analysis computes the stress equivalently as $S_{11} = Eu/L$. Choosing Young's modulus, $\boldsymbol{E}$, as the design parameter, the stress sensitivity is given by

$$\frac{dS_{11}}{dE} = \frac{\partial S_{11}}{\partial E} + \frac{\partial S_{11}}{\partial u}\frac{du}{dE} = \frac{u}{L} + \frac{E}{L}\frac{du}{dE}$$

for the total formulation and

$$\frac{dS_{11}}{dE} = \frac{\partial S_{11}}{\partial E} + \frac{\partial S_{11}}{\partial u^t}\frac{du^t}{dE} + \frac{\partial S_{11}}{\partial \Delta u}\frac{d\Delta u}{dE} = \frac{u^t + \Delta u}{L} + \frac{E}{L}\frac{du^t}{dE} + \frac{E}{L}\frac{d\Delta u}{dE}$$

for the incremental formulation. This example is a valid analysis since elastic materials and truss elements are supported for DSA. Suppose now that a frame element is added, extending the length of the structure. If the frame element shares the same Young's modulus, the analysis becomes invalid since the dependency on the design parameter $\boldsymbol{E}$ causes the frame element, which is unsupported, to become directly design dependent (i.e., the term $\frac{\partial S_{11}}{\partial E}$ would need to be computed). On the other hand, if the frame uses a different modulus, say E_1 that is not a design parameter, the analysis again becomes valid, since the frame no longer depends directly on the design parameter $\boldsymbol{E}$.

Contact interactions

Surface-based contact between deformable and rigid surfaces with small- or finite-sliding relative surface motion including friction is supported in a design sensitivity analysis. In all the friction models only the

friction coefficients (no test data input) can be made design dependent. Shape design parameters are not valid for rigid surfaces. Contact between deformable surfaces is not supported.

Restarting a design sensitivity analysis

A design sensitivity analysis can be restarted (see "Restarting an analysis," Section 9.1.1). However, DSA must have been active in the base analysis, and no design parameter or gradient data can be modified in the restart run. The restarted analysis will follow all the DSA propagation rules that are applicable to a regular analysis. For total formulation DSA, you may choose to activate or deactivate DSA in any new step that is added to the restart run. However, for the incremental formulation DSA must have been active in the step at which restart is attempted for you to continue doing DSA in the restarted analysis.

Procedures

DSA is available in the following analysis procedures:

- Frequency analysis
- Static stress analysis (including nonlinear geometric effects and contact)

The following analysis procedures and techniques are not supported:

- Static stress analysis with the Riks method
- Substructuring
- Mesh modification or replacement
- Importing and transferring results
- Symmetric model generation and results transfer
- Contour integrals
- Cyclic symmetry in frequency procedures

Submodeling limitations

Design sensitivity analysis can be performed in both the global model and submodel, with the limitation that the DSA solution will not be interpolated from the global model to the submodel. This means that DSA is valid in the submodel only if the global solution that is interpolated onto the boundary of the submodel can be considered independent of the design parameters chosen for the submodel sensitivity analysis.

Material options

The following material models are supported:

- Isotropic, orthotropic, and anisotropic elasticity
- Hyperelasticity
- Hyperfoam

In these models only directly input material coefficients (not test data) can be made design dependent. If test data are specified, that material definition can be replaced by specifying the material coefficients

calculated by Abaqus/Design directly. Supported and unsupported material models can be mixed in the same analysis.

Elements

Solid, truss, shell, beam, gasket, and membrane stress/displacement elements are supported. Shell elements with five degrees of freedom per node cannot be used in a total DSA formulation. Supported and unsupported elements can be mixed in the same analysis.

Output

The responses and response sensitivities (see "Specifying responses" above) are output only to the output database (sensitivity output to the data file and results file is not supported). The names of the sensitivities are related to the names of the responses as follows:

d_{response name}_{design parameter name}

For example, if the name of the response is S and the name of the design parameter is Young, the name of the sensitivity is d_S_Young.

Input file template

***HEADING**
…
***PARAMETER**
Python expressions defining parameters.
***DESIGN PARAMETER**
List of independent parameters to be considered as design parameters.
…
NODE, NSET=nset*
Data lines to define the nodes.
PARAMETER SHAPE VARIATION, PARAMETER=parameter*
Data lines to define the gradients of coordinates with respect to the parameter.
…
ELEMENT, TYPE=solid element type***, ELSET=***elset_elastic*
Data lines to define the elements.
ELEMENT, TYPE=solid element type***, ELSET=***elset_hyper*
Data lines to define the elements.
SOLID SECTION, ELSET=elset_elastic***, MATERIAL=***elastic*
SOLID SECTION, ELSET=elset_hyper***, MATERIAL=***hyper*
MATERIAL, NAME=elastic*
***ELASTIC**
Data lines to define the elastic properties.
MATERIAL, NAME=hyper*

***HYPERELASTIC**
Data lines to define the hyperelastic properties.
…
***STEP,DSA**
***STATIC**
…
DESIGN RESPONSE, FREQUENCY=interval*
ELEMENT RESPONSE, ELSET=elset*
Data lines to specify the element response identifier keys.
NODE RESPONSE, NSET=nset*
Data lines to specify the nodal response identifier keys.
***END STEP**

20. Parametric Studies

Scripting parametric studies 20.1
Parametric studies: commands 20.2

20.1 Scripting parametric studies

- “Scripting parametric studies,” Section 20.1.1

20.1.1 SCRIPTING PARAMETRIC STUDIES

Products: Abaqus/Standard Abaqus/Explicit

References

- "Parametric input," Section 1.4.1
- "Parametric shape variation," Section 2.1.2
- "Parametric studies," Section 3.2.11

Overview

Parametric studies allow you to generate, execute, and gather the results of multiple analyses that differ only in the values of some of the parameters used in place of input quantities.

Parametric studies can be performed by:

- Creating a "template" parametrized input file from which the different parametric variations are generated.
- Preparing a script (a file with the **`.psf`** extension) that contains Python (Lutz, 1996) instructions to generate, execute, and gather output for the parametric variations of the parametrized input file.

The Python commands for scripting parametric studies are discussed in this section.

Introduction

Parametric studies require that multiple analyses be performed to provide information about the behavior of a structure or component at different design points in a design space. The inputs for these analyses differ only in the values assigned to the parameters of a parametrized keyword input file (identified with the **`.inp`** extension).

Parametric studies in Abaqus require a user-developed Python script in a file (identified with the **`.psf`** extension) that contains Python commands to define the parametric study. For example, consider a case where you wish to perform a parametric study in which the thickness of a shell is varied. You need to create a parametrized input file (in this example, a file named **`shell.inp`**) containing the parameter definition

```
*PARAMETER
thick1 = 5.
```

and the parameter usage:

```
*SHELL SECTION,ELSET=name, MATERIAL=name
<thick1>
```

You create the parametric study by developing a **`.psf`** file that contains a script of Python instructions specifying the different designs that are to be analyzed, as follows:

```
thick = ParStudy(par='thick1', name='shell')
thick.define(CONTINUOUS, par='thick1', domain=(10., 20.))
thick.sample(NUMBER, par='thick1', number=5)
thick.combine(MESH)
```

These scripting commands create five designs with corresponding section thicknesses of 10., 12.5, 15., 17.5, and 20.0. Each of these thicknesses will, in turn, replace the value of 5. specified in the parameter definition in **shell.inp**. You may then provide additional Python scripting commands in the **.psf** file instructing Abaqus to do the following:

- Generate a number of **shell_*id*.inp** files and corresponding Abaqus jobs using the **shell.inp** file as a template. (The identifier *id* that is appended to the input file name is unique to each design in the parametric study.) An example of the Python command for this is

  ```
  thick.generate(template='shell')
  ```

  In this example the **shell_*id*.inp** files will differ only in the value to be used for the shell thickness.
- Execute all the Abaqus jobs representing the different variations of the parametric study. The Python command for this is

  ```
  thick.execute(ALL)
  ```

You generally want to review certain key results from the large amount of data that is generated by a parametric study. Abaqus provides the following capabilities for this purpose:

- A command specifying the source from which the results of a parametric study will be gathered. For example:

  ```
  thick.output(file=ODB, step=1, inc=LAST)
  ```

  The command above sets the output location to the last frame of the first step in the output database (**.odb**) file. The default behavior is to gather results from the last frame of a given step in the results (**.fil**) file.
- Commands to gather the required results from the multiple analyses generated by the parametric study and report them in a file or table. For example, the sequence of Python scripting commands used to gather and report the value of a displacement at a key node for each of the designs is:

  ```
  thick.gather(results='n33_u', variable='U', node=33, step=1)
  thick.report(PRINT, par='thick1', results=('n33_u.2'))
  ```

  The commands above gather the results record **'n33_u'** (the displacement vector of node 33 at the end of Step 1 of the analysis) for each of the designs and then print a table of the U2 component (the second component of the results record) of displacement for all designs.
- The ability to visualize *X*–*Y* plot data gathered across multiple analyses using the Visualization module of Abaqus/CAE. A typical example is to obtain an *X*–*Y* plot of the value of the displacement at a key node versus the value of the shell thickness. This is done by gathering the appropriate

parametric study results in an ASCII file that can be read into the Visualization module to display the plot.

Organization of parametric studies

A parametric study in Abaqus is associated with a particular set of parameters that define the design space. Only the values of the parameters can change in a parametric study. A new parametric study must be created if you wish to consider a different set of parameters. Having selected the parameters to be considered in a parametric study, you must specify how each parameter is defined. Parameters are distinguished as either continuous or discrete in nature and may have a domain and reference value.

The design points in the design space that are to be analyzed are created by specifying sample values for each parameter (sampling) and by combining the parameter samples to create sets of design points. A few simple commands are provided for parameter value sampling and for combining the sampled parameter values; these commands are described in detail later.

An initial definition and sampling of the parameters in the parametric study must be given before any combinations of parameter samples can be specified. After the first combination the initial definition and/or sampling of any individual parameter can be changed before the next combination is specified, thus providing a great deal of flexibility within one parametric study.

The domain of possible values and the reference value for a parameter given in the parameter definition can be temporarily redefined in any sampling of that parameter by specifying them differently during the sampling. You need not specify the parameter domain and reference value in the parameter definition so long as these are specified during sampling.

Design constraints can be imposed on all of the designs. A design that violates any of the constraints will be eliminated.

Finally, after all parametric study variations have been analyzed, you can gather and report results across all or some of the designs of the parametric study.

In summary, parametric studies in Abaqus are organized as follows:

- Create parametric study.
- Define parameters: define parameter type (continuous or discrete valued) and possibly the parameter domain and reference value.
- Sample parameters: specify sampling option and data and possibly temporarily redefine the parameter domain and reference value.
- Combine parameter samples to create sets of designs.
- Constrain designs (optional).
- Generate designs and analysis job data.
- Execute the analysis jobs for selected designs of the study.
- Gather key results for selected designs of the study.
- Report gathered results.

Note: The sequence of steps—define, sample, and combine—can be repeated as often as is necessary to create all the required design sets. Multiple parametric studies can be performed on a model contained in one input file. In general, more parameters will be defined and used in place of input quantities in the input file than those involved in any particular parametric study. In these cases parameters not involved in a particular parametric study will retain their values defined in the input file for the purposes of that parametric study. Therefore, we can think of the parameter values defined in the input file as representing a nominal design; parametric studies create modified designs by overwriting the values of some (or all) parameters.

Defining the design space

The design space is defined by the selection of the parameters to be varied in the study as well as the specification of the parameter types and possible values they can have.

Parametric study creation

Use the ***aStudy*=ParStudy** scripting command (see "Create a parametric study.," Section 20.2.8) to create a parametric study and select the independent parameters to be considered for variation. ***aStudy*** is the Python variable name assigned by you to the parametric study object created by the command. The methods of the parametric study object are used to carry out all the actions of the parametric study.

Input File Usage: ***aStudy*=ParStudy** (**par**=, **name**=, **verbose**=, **directory**=)

Parameter definition

Use the ***aStudy*.define** command (see "Define parameters for parametric studies.," Section 20.2.3) to specify the parameter type (choose the CONTINUOUS or the DISCRETE token; a token is a symbolic constant used to select an option within a specific command) and, optionally, to specify the domain of possible parameter values and a reference value for the parameter. If the domain and/or reference value are not specified in this command, they can be specified in the parameter sampling.

Redefinitions of a parameter are treated as complete redefinitions; that is, no information is retained from the previous definition of that parameter.

Input File Usage: ***aStudy*.define** (*token*, **par**=, **domain**=, **reference**=)

CONTINUOUS parameter type

In this case the parameter can take any value in a continuous domain specified by minimum and maximum values; for example, **domain**=(3., 10.).

DISCRETE parameter type

In this case the parameter can take only the values specified in a list that defines the discrete domain; for example, **domain**=(1, 4, 9, 16).

Sampling and combining parameter values to create sets of design points

Each parameter in the parametric study must be sampled before the combination operation is used to create the first set of design points. Any parameter in the parametric study can be redefined or resampled before a subsequent combination operation is performed.

Parameter sampling

Use the ***aStudy*.sample** command (see "Sample parameters for parametric studies.," Section 20.2.10) and choose one of the available tokens (INTERVAL, NUMBER, REFERENCE, or VALUES) to select how the sampling is done. The sampling data that must be given depend on how the sampling is done, as described next.

Sampling by INTERVAL

This sampling command assumes that you specify a domain of possible parameter values and wish to sample parameter values at fixed intervals in the domain. Sampling of the extreme values of the parameter is always done. The number of parameter values sampled depends on the interval and the domain. Because the extreme values are sampled, the last sampling interval will generally be smaller than the interval you specify.

The domain specification in this sampling command is optional:

- If a domain is specified in this command, it temporarily redefines a domain specified in the **define** command.
- If a domain is not specified in this command, the domain specification from the **define** command is used for sampling.
- An error is flagged when a domain is not specified in this command or in the **define** command.

The sampling interval is interpreted differently for continuous and discrete parameters:

- For continuously valued parameters the interval at which the samples are spaced is based on a numerical value. For example, specifying **interval**=10. for a continuous parameter with **domain**=(10., 35.) will sample values of 10., 20., 30., and 35. for this parameter.
- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values. The index means the position of the entry in the list, starting at position 0 and continuing with positions 1, 2, 3, etc. In this case **interval** must be an integer number. For example, specifying **interval**=−2 for a discrete parameter with **domain**=(1., 2., 3., 5., 7., 10.) will create sample values of 10., 5., 2., and 1. for this parameter.

The interval can have a positive or negative value (zero is not permitted). A positive interval indicates that sampling starts at the minimum value for a continuous parameter or at the first value in the list of values for a discrete parameter (forward sampling). A negative interval indicates that sampling starts at the maximum value for a continuous parameter or at the last value in the list of values for a discrete parameter (reverse sampling). Reverse sampling is useful when the TUPLE combination operation is used (see the discussion of combination of parameter samples).

Two special cases of the INTERVAL option are noteworthy:

- A positive interval value larger than the range of continuous parameter values or the number of discrete parameter values will sample the minimum and maximum values of the continuous parameter or the first and last values in the discrete parameter list.
- A negative interval value larger (in absolute terms) than the range of continuous parameter values or the number of discrete parameter values will sample the maximum and minimum values of the continuous parameter or the last and first values in the discrete parameter list.

Input File Usage: ***aStudy*.sample** (INTERVAL, **par**=, **interval**=, **domain**=)

Sampling by NUMBER

This sampling option assumes that you specify a domain of possible parameter values and wish to sample a fixed number of parameter values in the domain. Except for a special case documented below, sampling of the extreme values of the parameter is always done. The parameter is sampled at equally spaced intervals (with some exceptions for discrete parameters, as discussed below) and the size of the interval depends on the number of values sampled as well as the domain.

The domain specification in this sampling command is optional:

- If a domain is specified in this command, it temporarily redefines the domain specified in the **define** command.
- If a domain is not specified in this command, the domain specification from the **define** command is used for sampling.
- An error is flagged when a domain is not specified in this command or in the **define** command.

The sampling interval is calculated and interpreted differently for continuous and discrete parameters:

- For continuous valued parameters the interval at which the samples are spaced is based on a numerical value. For example, specifying **number**=4 for a continuous parameter with **domain**=(10., 25.) will sample values of 10., 15., 20., and 25. for this parameter.
- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values (indexing starts at zero). For example, specifying **number**=3 for a discrete parameter with **domain**=(1., 2., 3., 5., 7., 10., 12.) will create sample values of 1., 5., and 12. for this parameter. The number of discrete parameter samples specified by you may not allow equally spaced sampling; for example, specifying **number**=5 or **number**=6 for the discrete parameter above does not allow equally spaced sampling. This is resolved by sampling the parameter values that are closest to being equally spaced by rounding the sampling index to the closest index in the list of values. For example, specifying **number**=5 for the discrete parameter above will create sample values of 1., 3., 5., 10., and 12. The values 1. and 12. are sampled because they are the extreme values. The explanation for the second sampled value being the third value in the list (the value 3.) is as follows: the sampling interval is (highest index − lowest index)/(**number** − 1) = (6 − 0)/(5 − 1) = 1.5; the second sampled value should then be the one with index = 0 + 1.5 = 1.5 in the list; since the index has to be an integer number, we round off to index = 2 and, thus, sample the third value in the list. The other sampled values can be explained similarly. The same rule is used for character string type discrete parameters. For example, specifying **number**=3 for a discrete parameter with

domain=('C3D8', 'C3D8R', 'C3D8I', 'C3D8H') will create sample values of 'C3D8', 'C3D8I', and 'C3D8H'.

Three special cases of the NUMBER option are noteworthy:

- Specifying **number**=1 will sample the central value of a parameter, which is useful when the center of the design space is of interest. It is the only case in which the use of the NUMBER option does not sample the extreme values of the parameter.
- Specifying **number**=2 will sample the extreme values of a parameter, which is useful when the boundaries of the design space are of interest.
- Specifying **number**=3 will sample the central and the extreme values of a parameter, which is useful when the center and the boundaries of the design space are of interest.

Specification of **number**=0 is not permitted. A negative value for **number** is permitted; this indicates that the sampling is to be in reverse order. For continuous parameters reverse order means that the first sampled value is the largest and the last sampled value is the smallest. For discrete parameters reverse order means that the first sampled value is the last in the list of values and the last sampled value is the first in the list of values. Sampling in reverse order is useful when the TUPLE combination operation is used (see the discussion of combination of parameter samples).

Input File Usage: ***aStudy*.sample** (NUMBER, **par**=, **number**=, **domain**=)

Sampling by REFERENCE

This sampling option allows you to specify a reference value for the parameter and to sample parameter values with respect to this reference value. It is useful for studying alternate designs with respect to an existing (reference) design.

This sampling command creates sample values symmetrically about the reference value at multiples of a given interval; in addition, the reference value is also sampled. The number of parameter values in the sample depends on the number of symmetrical pairs of values you specify.

The reference value specification in this sampling option is optional:

- If a reference value is specified in this command, it temporarily redefines the reference specified in the **define** command.
- If a reference value is not specified in this command, the reference specification from the **define** command is used for sampling.
- An error is flagged when a reference value is not specified in this command or in the **define** command.

The reference value is interpreted differently for continuous and discrete parameters:

- For continuous valued parameters **reference** is the parameter's numerical value about which a symmetrical sample will be created.
- For discrete valued parameters **reference** is the index of the list of values about which a symmetrical sample will be created.

A reference value that falls outside the domain definition for the parameter is flagged as an error.

The sampling interval is interpreted differently for continuous and discrete parameters:

- For continuous valued parameters the interval at which the samples are taken is based on a numerical value. For example, specifying **reference**=50., **interval**=10., and **numSymPairs**=2 for a continuous parameter will create sample values of 30., 40., 50., 60., and 70. for this parameter.
- For discrete valued parameters the interval at which the samples are spaced is based on the index of the list of values (indexing starts at zero); in this case **interval** must be an integer value. For example, specifying **reference**=5, **interval**=−2 and **numSymPairs**=2 for a discrete parameter with **domain**=[1, 2, 3, 5, 7, 10, 12, 15, 20, 25] will create sample values of 25, 15, 10, 5, and 2 for this parameter.

The specified **interval** can have a positive or negative value, but a value of zero is not permitted. A positive interval indicates that the list of sampled values starts with the smallest sampled value for a continuous parameter or with the sampled value closest to the beginning of the list of values for a discrete parameter (forward sampling). A negative interval indicates that the list of sampled values starts with the largest sampled value for a continuous parameter or with the sampled value closest to the end of the list of values for a discrete parameter (reverse sampling). Reverse sampling is useful when the TUPLE combination operation is used (see the discussion on combination of parameter samples).

The number of symmetrical pairs you specify must be zero or a positive integer; setting the number of symmetrical pairs equal to zero indicates that only the reference value is sampled.

The domain specification in this command is optional:

- If a domain is specified in this command, it temporarily redefines the domain specified in the **define** command.
- If a domain is not specified in this command, the domain specification from the **define** command is used for sampling.
- An error is flagged in the case of discrete valued parameters when a domain is not specified in this command or in the **define** command.

A domain specification (either in this command or in the **define** command) is required for discrete valued parameters because the possible discrete values that can be sampled must be known. Although a domain specification is not required for continuous valued parameters, it may be given. In either the case of discrete parameters or the case of continuous parameters, a domain specification can be used to limit the number of values sampled using the REFERENCE option since the domain is treated as a bound on the possible sampling values. For example, specifying **reference**=50., **interval**=10., and **numSymPairs**=3 for a continuous parameter with **domain**=(35., 100.) will sample values of 40., 50., 60., 70., and 80. for this parameter. The minimum value of the domain acts as a bound in this sampling.

Input File Usage: ***aStudy*.sample** (REFERENCE, **par**=, **reference**=, **interval**=, **numSymPairs**=, **domain**=)

Sampling by VALUES

This sampling option assumes that you wish to create the parameter sample values directly. You must specify the actual parameter values, irrespective of whether the parameter is continuous or discrete.

A parameter domain specified in the **define** command does not affect the values sampled for the parameter when this option is used.

Input File Usage: ***aStudy*.sample** (VALUES, **par**=, **values**=)

Combination of parameter samples

Use the ***aStudy*.combine** command (see "Combine parameter samples for parametric studies.," Section 20.2.1) to create sets of design points from the parameter samples. Choose how the combining is done using one of the following tokens: MESH, TUPLE, or CROSS. The use of each combination command results in the creation of a number of design points, which are grouped into design sets. If a combine operation creates a design that duplicates a design in an existing design set, the duplicate design is deleted immediately. The total number of designs in a parametric study (before the application of any design constraints) is the sum of the number of designs in each design set.

You can name a design set; if you do not, it is named by default. The default naming convention is *p1* for the first non-user-named design set in the parametric study, *p2* for the second non-user-named design set, and so on. The design set name is used to help identify individual designs. A design set named by you with a name identical to a previously specified design set name indicates that it is a respecification of the design set and, thus, overwrites the previously existing one.

Input File Usage: ***aStudy*.combine** (*token*, **name**=)

MESH combination

This combine option indicates that every sampled value for a parameter is to be combined with every sampled value of every other parameter in the parametric study.

The following examples illustrate the use of the MESH combine option. In a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13))
study.sample(REFERENCE, par='par1', reference=0,
    interval=2, numSymPairs=2)
study.define(CONTINUOUS, par='par2', domain=(10., 60.))
study.sample(INTERVAL, par='par2', interval=20.)
```

the combine command

```
study.combine(MESH, name='dSet1')
```

creates the following 12 design points (**par1**, **par2**): (1, 10.), (5, 10.), (9, 10.), (1, 30.), (5, 30.), (9, 30.), (1, 50.), (5, 50.), (9, 50.), (1, 60.), (5, 60.), and (9, 60.) (see Figure 20.1.1–1).

A second use of the combine command preceded by a respecification of the parameter sampling

```
study.sample(NUMBER, par='par1', number=3)
study.sample(NUMBER, par='par2', number=3)
study.combine(MESH, name='dSet2')
```

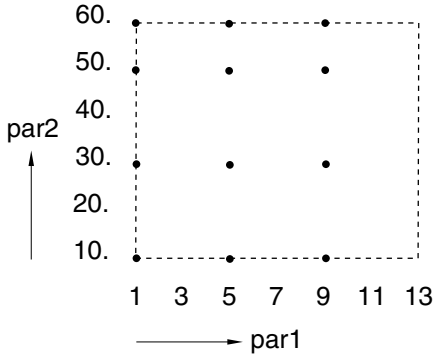


Figure 20.1.1–1 Design points in design set **`dSet1`** created with the MESH option of the combine command.

creates designs at the following nine points: (1, 10.), (7, 10.), (13, 10.), (1, 35.), (7, 35.), (13, 35.), (1, 60.), (7, 60.), and (13, 60.) (see Figure 20.1.1–2). The extreme and center values of both parameters are combined.

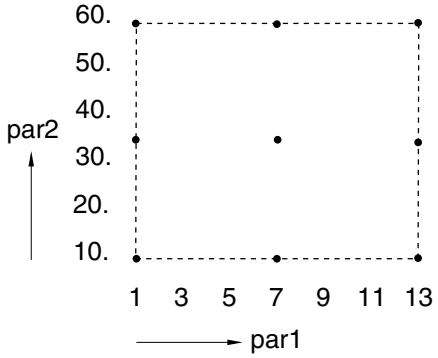


Figure 20.1.1–2 Design points in design set **`dSet2`** created with the MESH option of the combine command after the parameter sampling is redefined.

TUPLE combination

This combine option creates design sets consisting of *n*-tuples of the sampled parameter values, where *n* is the number of parameters in the parametric study. Each *n*-tuple consists of one sampled value for each parameter. For example, in a three-parameter study the first sampled value of each of the three parameters makes up the first 3-tuple, the second sampled value of each of the three parameters makes up the second 3-tuple, and so on. The creation of tuples ceases when any of the parameter samples runs out of sampled values.

The following examples illustrate the use of the TUPLE combination operation. For a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13))
study.define(CONTINUOUS, par='par2', domain=(10., 60.))
study.sample(INTERVAL, par='par1', interval=1)
study.sample(INTERVAL, par='par2', interval=10.)
```

the combination operation

```
study.combine(TUPLE, name='dSet3')
```

creates designs at the following 6 points: (1, 10.), (3, 20.), (5, 30.), (7, 40.), (9, 50.), and (11, 60.) (see Figure 20.1.1–3). This represents a diagonal pattern in the two-parameter space. We see that all **par2** values are used in the tuple combination but the last **par1** value is not used because there are no more **par2** sample values to form additional tuples.

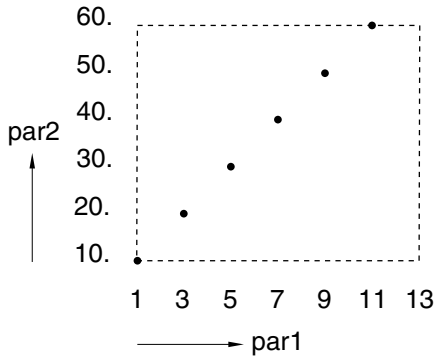


Figure 20.1.1–3 Design points in design set **dSet3** created with the TUPLE option of the combine command.

A second invocation of the above combine command after respecifying the **par2** sampling as

```
study.sample(INTERVAL, par='par2', interval=-10.)
study.combine(TUPLE, name='dSet4')
```

creates designs at the following 6 points: (1, 60.), (3, 50.), (5, 40.), (7, 30.), (9, 20.), and (11, 10.) (see Figure 20.1.1–4). This represents the other diagonal in the two-parameter space.

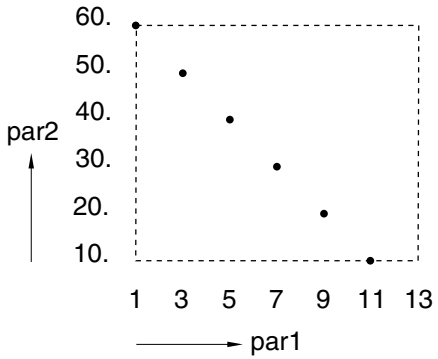


Figure 20.1.1–4 Design points in design set **dSet4** created with the TUPLE option of the combine command after the parameter sampling is redefined.

CROSS combination

This combine option creates designs in the form of "cross-shaped" patterns as follows: each value sampled for an individual parameter is combined with the reference value, as specified in the **define** command, of all the other parameters used in the parametric study. To use the CROSS combine option, it is necessary to specify a reference value in the **define** command for all parameters in the parametric study.

The reference value specified for a parameter in the **define** command does not have to coincide with a value sampled by that parameter's sampling rule. However, if the reference value does not coincide with a sampled value, the reference parameter value is not added to the list of sampled values for that parameter; it is used only to form the CROSS combination.

The following examples illustrate the use of the CROSS combine option. For a two-parameter study with the parameters defined and sampled as

```
study=ParStudy(par=('par1', 'par2'))
study.define(DISCRETE, par='par1',
    domain=(1, 3, 5, 7, 9, 11, 13), reference=3)
study.define(CONTINUOUS, par='par2',
    domain=(10., 60.), reference=40.)
study.sample(REFERENCE, par='par1', interval=1,
    numSymPairs=3)
study.sample(INTERVAL, par='par2', interval=10.)
```

the combine cross option

```
study.combine(CROSS, name='dSet5')
```

creates designs at the following 12 points: (1, 40.), (3, 40.), (5, 40.), (7, 40.), (9, 40.), (11, 40.), (13, 40.), (7, 10.), (7, 20.), (7, 30.), (7, 50.), and (7, 60.) (see Figure 20.1.1–5). This combination is a cross-shaped pattern where the cross intersection is at (7, 40.) as specified [7 is the fourth value (**reference**=3) of the discrete parameter **par1**].

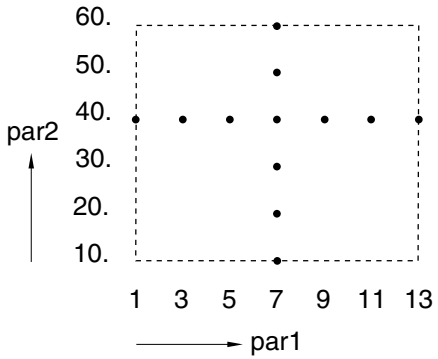


Figure 20.1.1–5 Design points in design set **dSet5** created with the CROSS option of the combine command.

A second invocation of the above combine command after respecifying **par2** as

```
study.define(CONTINUOUS, par='par2', domain=(10., 60.),
    reference=45.)
study.combine(CROSS, name='dSet6')
```

creates designs at the following 13 design points: (1, 45.), (3, 45.), (5, 45.), (7, 45.), (9, 45.), (11, 45.), (13, 45.), (7, 10.), (7, 20.), (7, 30.), (7, 40.), (7, 50.), and (7, 60.) (see Figure 20.1.1–6).

Constraining designs

A constraint that determines the allowable design points in the parametric study can be specified using the ***aStudy*.constrain** scripting command (see "Constrain parameter value combinations in parametric studies.," Section 20.2.2). When such a constraint is specified, existing designs that violate the constraint are eliminated immediately.

For example, the constrain command

```
aStudy.constrain('height*width < 12.')
```

where **height** and **width** are parameters in the parametric study, can be used to enforce that the cross-sectional area of a rectangular beam must be less than 12.0 in all designs.

Input File Usage: ***aStudy*.constrain** ('*constraint expression*')

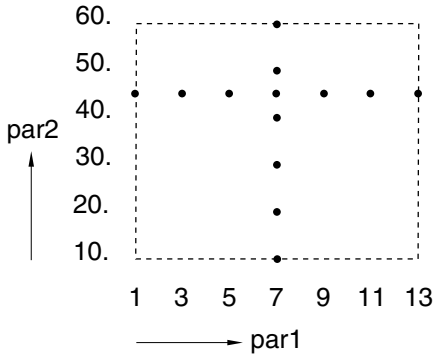


Figure 20.1.1–6 Design points in design set **dSet6** created with the CROSS option of the combine command when the parameter **par2** is redefined.

Generation and execution of the designs of a parametric study

Once the required design points have been specified, it is necessary to generate the corresponding analysis job data and execute the analyses.

Generation of analysis job data

Use the ***aStudy*.generate** scripting command (see "Generate the analysis job data for a parametric study.," Section 20.2.6) to generate an input file for each design.

The name of the parametrized template input file from which the input files of each design are generated must be specified.

The naming convention for the input files generated by the parametric study is as follows:

- The name of each analysis job will start with the template input file name; for example, **shell**.
- The name of the parametric study (specified by you in the **ParStudy** command using the **name=** option) is appended, preceded by an underscore (_); for example, **shell_thickness** for a parametric study defined with the **study = ParStudy(name='thickness')** command. (If no parametric study name has been given, the parametric study name defaults to the name of the Python script file in which the study is defined.) If the template input file name and parametric study name are the same, the name is not repeated.
- The name of the design set (specified or created by default in the **combine** command) is appended, preceded by an underscore (_); for example, **shell_thickness_p1** for the first design set (named by default) of the above parametric study.
- The design name (created automatically in the **combine** command) is appended, preceded by an underscore (_); for example, **shell_thickness_p1_c1** for the first design in the first design set of the above parametric study.

As usual, the input files each have the extension **.inp**. You can examine and/or edit these input files before execution.

The **generate** command creates a file with the **.var** extension that contains a description of the parametric study. This file is given the parametric study name—for example, *studyName*.**var**—and contains a list of all the designs generated, together with the parameter values associated with each design. You can examine and/or edit this file; however, editing this file will affect the gathering of results across the designs of the parametric study (see "Gathering results").

Each time the **generate** command is used, new versions of the *studyName*.**var** files are created reflecting the designs specified by all previous **combine** commands.

The define, sample, and combine steps are performed before the generate command is executed. It is, thus, possible for you to refer to parameters that do not exist or are not independent parameters in the template input file. These errors are detected and flagged by the **generate** command.

Input File Usage: ***aStudy*.generate (template)**

Execution of the analysis of the designs of a parametric study

Use the ***aStudy*.execute** command (see "Execute the analysis of parametric study designs.," Section 20.2.4) to execute the analysis of the designs of the study.

The command will submit a number of analysis jobs for execution under the control of a Python process. All designs can be evaluated without further user interaction by specifying the ALL or DISTRIBUTED options of this command, or you can control the execution of the analyses interactively by specifying the INTERACTIVE option. In the interactive case you are prompted for further execution instructions. The prompt allows you to:

- Specify a number of analyses to be executed before the process pauses and prompts you again.
- Execute the remaining analyses without further user interaction.
- Specify a number of analyses whose execution is to be skipped before the process pauses and prompts you again.
- Stop execution.

The interactive option is useful because it provides the opportunity to:

- Study the results of the analyses already executed.
- Delete unnecessary files generated by the analyses to conserve disk space when many designs are being analyzed.
- Analyze only certain designs of the parametric study.

The ALL and INTERACTIVE tokens are used for sequential execution of the Abaqus analyses on your machine. The DISTRIBUTED token can be used to schedule analysis jobs on multiple machines or multiple CPUs of one machine.

The DISTRIBUTED option is available only on variants of the Linux operating system. In particular the implementation depends on the operating system for support of the **rsh**, **rcp**, and **xhost** Linux commands. Because of the use of these commands, it is necessary that for distributed execution of parametric studies the parametric study must itself be executed on your local computer. If binary results are output during the analyses, both the local and remote computers must be binary

compatible. Before the distributed execution capability can be used, it is necessary to configure the appropriate queue interfaces in the Abaqus environment file.

Each analysis of a design of the parametric study that is executed when you issue the **execute** command is, by default, executed by Abaqus in background mode, irrespective of the command option used. Files created by the Abaqus analysis of each design will overwrite any existing files of the same name without you being prompted.

You can add any necessary Abaqus execution options (refer to "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2) to the execution command for each of the analyses by specifying them with the **execOptions** option.

Input File Usage: ***aStudy*.execute** (*token*, **files**= , **queues**= , **execOptions**=)

Parametric study results

Once the analyses associated with a parametric study have been executed, the variation of key results across the different designs can be examined. First, results must be gathered from the results file or output database of each of the analyses; then, these results must be reported.

The ***aStudy*.output** command (see "Specify the source of parametric study results.," Section 20.2.7) can be used to specify the source of the results to be gathered. All arguments to the ***aStudy*.output** command are optional: the specification of the file, the analysis step, and the increment (for non-frequency steps) or the mode (for frequency steps) from which the results are to be gathered. If the file is not specified, the results (`.fil`) file will be used. If the step is not specified, it must be specified in the **gather** command (see the discussion that follows). The defaults of the increment (for non-frequency steps) or the mode (for frequency steps) are the last increment of the step and the first mode calculated in the step unless specified in the **gather** command. Some arguments are applicable only to the output (`.odb`) database: the instance name, the request type (field or history), the frame value where the results are to be gathered, and whether the memory used to access an output database should be overlayed when a different output database is accessed.

The specification of the source of the gathered results remains in effect for all subsequent **gather** commands until the source is respecified. Respecification of the gathering source is treated as a complete respecification; that is, nothing is retained from the previous specification of the gathering source.

Input File Usage: ***aStudy*.output** (**file**=, **instance**=, **overlay**=, **request**=, **step**=, **frameValue**= | **inc**= | **mode**=)

Gathering results

Use the ***aStudy*.gather** command (see "Gather the results of a parametric study.," Section 20.2.5) to gather results from the results file or output database of each of the analyses.

In each use of the **gather** command you must specify a name that is associated with the gathered result record. This label is used to refer to the results record in the **report** commands.

When gathering results from the results (`.fil`) file, each result record to be gathered must be chosen by specifying one of the available output variable identifier keys appearing under the `.fil` column heading in "Abaqus/Standard output variable identifiers," Section 4.2.1, or "Abaqus/Explicit output variable identifiers," Section 4.2.2. For example, the U or S variable identifier keys can be

specified, but the U1 or S11 variable identifier keys cannot be specified. In addition, the MODAL variable identifier key can be specified to gather eigenvalue results records (those written to the results file with the record key 1980); in this case, no variable location data are required.

When gathering results from the output (**.odb**) database, each result record to be gathered is chosen by specifying one of the available output variable identifier keys appearing under the **.odb** column heading in "Abaqus/Standard output variable identifiers," Section 4.2.1, or "Abaqus/Explicit output variable identifiers," Section 4.2.2. For field output the component must not be specified, while for history output the component number is required; for example, the U or S variable identifier keys can be specified for field output, while the U1 or S11 variable identifier keys can be specified for history output. Unless the output is at the assembly level, an instance name must be provided as an argument to the **gather** command. An exception to this instance-name requirement is the case where the output (**.odb**) database is generated from a model not defined as an assembly of part instances, which is inferred from the presence in the output database of a single assembly named Assembly-1 and a single part instance named PART–1–1. In this case you need not explicitly refer to the instance PART–1–1.

The names of components of the result records are created automatically. For example, the command

```
myStudy.gather(results='e52_stress', variable='S', element=52)
```

creates a result record **e52_stress** whose six components (in the case of a three-dimensional solid element) are named: **e52_stress.1** (the S11 stress component), **e52_stress.2** (the S22 stress component), **e52_stress.3** (the S33 stress component), **e52_stress.4** (the S12 stress component), **e52_stress.5** (the S13 stress component), and **e52_stress.6** (the S23 stress component).

The variable location data that must be given depend on the output variable identifier key specified. (Refer to "Gather the results of a parametric study.," Section 20.2.5, for a description of the location data.) Enough variable location data must be given to define a unique result record.

Input File Usage: ***aStudy*.gather** (**request**=, **results**=, **step**=, **frameValue**= | **inc**= | **mode**=, **variable**=, *additional location data*)

Reporting results

Use the ***aStudy*.report** scripting command (see "Report parametric study results.," Section 20.2.9) to report results gathered from the results files of the parametric study. Use the PRINT, FILE, or XYPLOT options to specify the kind of report to be produced:

- PRINT indicates that a table of results (with headings) is to be printed to the default output device (the screen). You may wish to limit the number of columns in a table so as to make the table readable.
- FILE indicates that a table of results (with headings) is to be written to an ASCII file.
- XYPLOT indicates that a table of results (without headings) is to be written to an ASCII file that can later be read into the Visualization module in Abaqus/CAE to display *X*–*Y* plots.

Each row in a table represents a design in the parametric study. A column in a table can represent values of a parameter, values of a gathered result, or design names.

One or more parameters can be specified in the **report** command. If no parameters are specified, the default is that all parameters in the parametric study are included in the table. The column corresponding to each parameter shows the value of that parameter in each of the designs included in the table.

A design set name can be specified to restrict the rows in the table to designs that are part of the set (refer to the **combine** command described earlier). If a design set is not specified, the default is that all designs are included in the table.

Use **variations**=ON to specify that the first column in a table must show the design names. If **variations**=ON is not specified or **variations**=OFF is included, the column of design names is not included in the table.

The names of the results to be reported must be specified as a sequence; for example, the Mises stress of element 33, the S22 stress of element 52, and the U3 displacement of node 10 are gathered in the following three separate commands:

```
myStudy.gather(results='e33_sinv', variable='SINV', element=33)
myStudy.gather(results='e52_s', variable='S', element=52)
myStudy.gather(results='n10_u', variable='U', node=10)
```

These results can be printed in a single table using the following **report** command:

```
myStudy.report(PRINT,
    results=('e33_sinv.1', 'e52_s.2', 'n10_u.3'))
```

This example shows not only how gathered results of different types (element, nodal, etc.) can be collected in a single table but also how to refer to components of results records (the Mises stress is the first component of SINV, S22 is the second component of S, and U3 is the third component of U—refer to "Results file output format," Section 5.1.2, or the Abaqus Scripting User's Guide for a description of how the results are stored in the results file and the output database, respectively).

When either the FILE token or the XYPLOT token is used, a file name must be given to specify the file to which the results are to be written. A subsequent **report** command issued in the same session using the same file name will append the new results to the file. However, a subsequent **report** command issued in a different session using the same file name overwrites the existing file.

Input File Usage: ***aStudy*.report** (*token*, **results**=, **par**=, **designSet**=, **variations**=, **file**=)

Execution of parametric studies

To carry out a parametric study, you must prepare a parametrized input file (*inputFile*`.inp`). This input file is the template used for the generation of the parametric variations of the study and must contain the parameter definitions necessary to use parameters in place of input quantities. The parameters must be defined in the template file; they cannot be defined in any include files that are referred to by the template file.

In addition, you must prepare a Python scripting file, *scriptFile*`.psf`, containing instructions that script the actions of the parametric study.

Typically, you prepare the Python scripting file using an editor and then invoke execution of this file using the Abaqus execution command **abaqus script**=*scriptFile*. This command starts the Python

interpreter and executes the instructions in the scripting file. Alternatively, you simply start the Python interpreter, without giving a scripting file, with the Abaqus execution command **abaqus script**. In this case the Python interpreter remains active, and you can execute additional commands interactively or execute additional commands contained in a file (for example, *fileName*) using the Python command **execfile('*fileName*')**. The Python interpreter can be terminated using [Ctrl] + d on a Linux machine or [Ctrl] + z on a Windows machine.

It is normally preferable to execute a previously prepared scripting file because it is likely that you will want to develop the script iteratively; in this case you simply have to go back and edit the scripting file and re-execute it until satisfied with the result.

You can monitor the progress of the parametric variation analyses using normal operating system commands.

Execution in more than one session

You may want to gather and report results multiple times, after the parametric variations of the study have been executed. It is possible to define, generate, and execute a parametric study in one session and gather and report results in a separate session. Only the command used to create the parametric study needs to be reissued when you start a new session.

Visualization of parametric study results

The results of the analysis of a particular parametric study variation can be visualized like any other results of a single analysis.

Visualization of results gathered across designs of a parametric study requires gathering of results. For visualization the results must be reported in ASCII files (using the XYPLOT option in the **gather** command), which can be read by the Visualization module in Abaqus/CAE to produce *X–Y* plots of results versus parameter values or design names.

Scripting commands

Parametric studies are scripted in files with the **`.psf`** extension using the Python language (Lutz, 1996). A parametric study object, constructed from the **ParStudy** class, is provided whose methods make scripting of parametric studies straightforward; these methods are described in this chapter.

Syntax of scripting commands

Scripting commands generally have the following form:

```
aStudy.method (token, data)
```

aStudy is the parametric study object to which the method applies; this object is constructed using the parametric study constructor command. ***method*** is the method to be used; for example, **define**, **sample**, or **execute**.

Most (but not all) commands have a *token* that selects an option of the command; for example, ***aStudy*.define (CONTINUOUS, par=)** indicates that one or more continuous parameters are being defined in a parametric study. Tokens are always given in capital letters and are mutually exclusive.

For most (but not all) commands additional *data* must be specified.

Python language rules

The parametric study scripts contained in scripting files must follow the syntax and semantics of the Python language. Some important aspects of this language are described here (more general Python language rules are discussed in "Parametric input," Section 1.4.1).

Comments

Comments must be preceded by the # symbol. The comment is understood to continue to the end of the line. For example,

```
#
# This parametric study ...
#
studyTempEffects.generate(template='shell') #use shell input file
```

Case sensitivity

All variable and method names, tokens, and character string literals are case sensitive across all operating systems. For example,

```
study.execute( ) # is valid
study.Execute( ) # is not valid because of the capital E

study.sample(NUMBER, ...) # uses the valid token NUMBER
study.sample(number, ...) # lower case token is not valid

study.generate(template='aFile') # 'aFile' is different
study.generate(template='afile') # from 'afile'
```

Character strings

Character strings are indicated by using paired single (' ') or double (" ") quotation marks. Backward single quotation marks (` `) cannot be used for this purpose. For example,

```
"double quoted string"
'single quoted string'
```

Printing

The Python **print** command can be used to obtain a printed representation of any Python object. For example,

```
print 'MY TEXT'
```

will print **`MY TEXT`** on the standard output device.

Lists and tuples

The scripting methods for parametric studies accept integer, real, and character string types. These primitive types can, in many cases, optionally be contained within tuple or list structures. Although there are some differences between lists and tuples in Python, they can be used interchangeably in the parametric study scripting commands; they simply represent ordered sequences of items. Items in lists or tuples must be separated by commas and enclosed in parentheses or brackets. For example,

```
aStudy.define(CONTINUOUS, par=('xCoord',))# tuple contains a
                                          # single string item
aStudy.define(CONTINUOUS, par=['xCoord']) # list contains a
                                          # single string item
aStudy.define(CONTINUOUS, par=('xCoord', 'yCoord')) # tuple
aStudy.define(CONTINUOUS, par=['xCoord', 'yCoord']) # list
```

Indentation

Python uses indentation to group blocks of statements. Therefore, a Python statement should begin in the same column as the preceding statement except where grouping of statements is required by Python.

Accessing the data of a parametric study

In some cases it is desirable to have direct programmatic access to the data of a parametric study. Therefore, all of the important data of the study are stored in repositories that can be accessed as data members of the parametric study object. The repositories have a similar interface and similar behavior to that of Python dictionaries. Repository data are stored as key, value pairs; and methods are provided for accessing the repository keys and values. The syntax ***aValue = aRepository[aKey]*** is used to retrieve the value associated with a repository key. A list of the keys of the repository is obtained using the **keys()** method of the repository; for example, ***allKeys = aRepository*.keys()**. Similarly, a list of the values of the repository is obtained using the **values()** method of the repository; for example, ***allValues = aRepository*.values()**. The following parametric study script shows an example of how the parameter repository of a parametric study can be accessed and how a list of the parameter names and a list of the sample values for the parameter **t1** can be obtained:

```
studyTempEffect = ParStudy(par=('t1', 't2'))
studyTempEffect.define(CONTINUOUS, par=('t1', 't2'))
studyTempEffect.sample(VALUES, par='t1', values=(200.,300.,400.))
studyTempEffect.sample(VALUES, par='t2', values=(250.,350.,450.))
parRepository = studyTempEffect.parameter
listOfParameters = parRepository.keys()
t1Sample = parRepository['t1'].sample
```

The script results in the following assignments: **listOfParameters = ['t1', 't2']** and **t1Sample = [200.0, 300.0, 400.0]**. The Python **print** command can be used to obtain information on the contents of a repository.

Parametric study repositories

A parametric study has the following repositories and objects as data members:

- ***aStudy*.parameter**: A repository for parameter objects keyed by parameter name. Each parameter object has a **name**, **type**, **domain**, **reference**, and **sample** data member.
- ***aStudy*.designSet**: A repository for design sets keyed by design set name. Each design set is represented as a list of design points.
- ***aStudy*.job**: A repository for analysis job objects keyed by the name of the corresponding analysis input file name (without the **`.inp`** extension). Each job object has a **design**, **status**, **root**, **designSet**, and **designName** data member.
- ***aStudy*.resultData**: A repository for result records keyed by a name constructed by successively appending the result name, the underscore character (_), and the design name. For results retrieved from the result (**`.fil`**) file, each result record is in the format of the result (**`.fil`**) file record for the corresponding output variable. For field results retrieved from the output (**`.odb`**) database, each result record will be a tuple containing the components of the results. The result record for history results from the output (**`.odb`**) database, which can only be retrieved for a single component, will be a tuple containing a single value.
- ***aStudy*.table**: A table object containing a representation of the table formatted by the last use of the **report** command. The table object has **title**, **variation**, **designs**, and **results** data members.

Additional reference

- Lutz, M., *Programming Python,* O'Reilly & Associates, Inc., 1996.

20.2 Parametric studies: commands

- “Combine parameter samples for parametric studies.,” Section 20.2.1
- “Constrain parameter value combinations in parametric studies.,” Section 20.2.2
- “Define parameters for parametric studies.,” Section 20.2.3
- “Execute the analysis of parametric study designs.,” Section 20.2.4
- “Gather the results of a parametric study.,” Section 20.2.5
- “Generate the analysis job data for a parametric study.,” Section 20.2.6
- “Specify the source of parametric study results.,” Section 20.2.7
- “Create a parametric study.,” Section 20.2.8
- “Report parametric study results.,” Section 20.2.9
- “Sample parameters for parametric studies.,” Section 20.2.10

20.2.1 *aStudy*.combine(): Combine parameter samples for parametric studies.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to combine the sampled parameter values in a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.combine (token*, *additional data*)

Tokens:

CROSS

Use this token to create designs in a "cross-shaped" pattern from the sampled values of the parameters in the parametric study.

MESH

Use this token to create designs in a "mesh" pattern in which every sampled value of a parameter is combined with every sampled value of every other parameter in the parametric study.

PRINT

Use this token to print the design points created for the parametric study.

TUPLE

Use this token to create designs in a "tuple" pattern consisting of tuples of sampled values of the parameters in the parametric study.

Additional data for CROSS, MESH, and TUPLE:

Optional data:

name

Set **name** equal to the name of the design set being defined; this name must be enclosed by quotation marks. A default design set name is created if a name is not specified.

Additional data for PRINT:

Optional data:

name

Set **name** equal to the name of the design set for which information is being printed; this name must be enclosed by quotation marks. If a design set name is not specified, information is printed for all the design sets in the parametric study.

20.2.2 *aStudy*.constrain(): Constrain parameter value combinations in parametric studies.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to define constraints on parameter value combinations; combinations that violate any of the constraints are eliminated from the parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.constrain (constraint expression*)

Required data:

constraint expression

Provide a constraint expression enclosed by matching quotation marks. This expression may involve operations among parameters, numbers, and previously defined Python variables; for example, `'height*width < maxArea-2.0'`. The constraint can be an equality or an inequality.

20.2.3 *aStudy*.define(): Define parameters for parametric studies.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to define the parameters specified for a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.define (token, additional data***)**

Tokens:

CONTINUOUS

Use this token to indicate that the parameter is continuous valued.

DISCRETE

Use this token to indicate that the parameter is discrete valued.

PRINT

Use this token to print parameter definitions.

Additional data for CONTINUOUS:

Required data:

par

Set **par** equal to the name of the parameter or the sequence of parameters being defined. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a list of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

Optional data:

domain

Set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, `(10., 20.)` or `[10., 20.]`.

If **domain** is omitted from this command and the parameter is later sampled using a method that requires a domain definition, the domain must be specified in the **sample** command.

reference

Set **reference** equal to the reference value of the parameter.

If **reference** is omitted from this command and the parameter is later sampled using a method that requires a reference definition, the reference must be specified in the **sample** command.

Additional data for DISCRETE:

Required data:

par

Set **par** equal to the name of the parameter or the list of parameters being defined. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a list of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

Optional data:

domain

Set **domain** equal to the sequence of values that the parameter may have. The values must be separated by commas and enclosed by parentheses or brackets; for example, `(1., 2., 5., 3.)` or `[1., 2., 5., 3.]`.

If **domain** is omitted from this command and the parameter is later sampled using a method that requires a domain definition, the domain must be specified in the **sample** command.

reference

Set **reference** equal to the index in the sequence of parameter values. Indexing starts at zero, so that the first value of the sequence corresponds to index zero and the last value of the sequence corresponds to an index equal to the number of values in the sequence minus one.

If **reference** is omitted from this command and the parameter is later sampled using a method that requires a reference definition, the reference must be specified in the **sample** command.

Additional data for PRINT:

Optional data:

par

Set **par** equal to the name of the parameter or the sequence of parameters whose definition is to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

If **par** is omitted, parameter definitions are printed for all parameters in the parametric study.

20.2.4 *aStudy*.execute(): Execute the analysis of parametric study designs.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to execute the analyses of the designs generated by a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.execute (token*, **execOptions= ,** *additional data***)**

Tokens:

ALL

Use this token to sequentially execute the analyses of all the designs of the parametric study. This option is the default.

DISTRIBUTED

Use this token to execute the analyses of all designs using the specified queue interfaces of the local and/or remote computers. A similar number of analyses will be distributed to each of the specified queues.

INTERACTIVE

Use this token to sequentially execute the analyses of all the designs of the parametric study in interactive mode. In this case the process pauses to prompt you for further execution instructions. The prompt allows you to specify the number of analyses to be executed, to execute the remaining analyses, to specify the number of analyses whose execution is to be skipped, or to skip all the remaining analyses.

Optional data:

execOptions

Set **execOptions** equal to a character string of Abaqus execution options (refer to "Abaqus/Standard, Abaqus/Explicit, and Abaqus/CFD execution," Section 3.2.2) that are to be added to the Abaqus execution command when executing the analyses of the designs of the parametric study; this string must be enclosed in matching quotation marks.

Additional data for DISTRIBUTED:

Required data:

queues

Set **queues** equal to the queue interface name or a sequence of queue interface names. If a single name is given, it must be enclosed in matching quotation marks. If a sequence of names is given, it must be

enclosed in parentheses or brackets and contain queue interface names enclosed in matching quotation marks and separated by commas.

Optional data:

files

Set **files** equal to the symbolic constant or a sequence of symbolic constants that identifies the file or files that must be returned to the local computer after remote execution. The sequence items must be separated by commas, and the sequence must be enclosed in parentheses or brackets.

The allowed symbolic constants are: **DAT**, **LOG**, **FIL**, **SEL**, **MSG**, **STA**, **ODB**, **IPM**, **RES**, **ABQ**, and **PAC**. The default value is **files = (DAT, FIL, LOG, ODB, SEL)**.

Defining queues and queue interfaces:

Before being used for a distributed parametric study, queue interfaces must be defined within the **design_startup** portion of the Abaqus environment file. For example, to define a queue interface for an existing queue **short** on the remote computer **server**, the following entry in the environment file is required:

```
def onDesignStartup():
    from par_Session import Queue
    import os

    # convenience assignment
    SCRATCH = '/scratch/' + os.environ['USER']

    # create remote queue interface
    Queue(name='short_interface', hostName='server',
          driver='abaqus', queueName='short', directory=SCRATCH)
```

If, in addition, a local queue is required, the entry must be expanded to:

```
def onDesignStartup():
    from par_Session import Queue
    import os

    # convenience assignment
    SCRATCH = '/scratch/' + os.environ['USER']

    # create remote queue interface
    Queue(name='short_interface', hostName='server',
          driver='abaqus', queueName='short', directory=SCRATCH)

    # create local queue interface
    Queue(name='local_interface', driver='abaqus',
```

```
          queueName='local')

queue_name="local"
local="echo "./%S 1>%L 2>&1" | batch"
```

20.2.5 *aStudy*.gather(): Gather the results of a parametric study.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to gather analysis results across the designs of a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.gather (request= , results= , step= , frameValue= | inc= | mode= , variable= ,** *additional data)**

Required data:

results

Set **results** equal to a name that will be used to identify the results record gathered by this command. This name must be enclosed in matching quotation marks.

variable

Set **variable** equal to an output variable identifier key; this key must be enclosed in matching quotation marks.

For gathering results from the results (`.fil`) file only those output variable identifier keys appearing under the **.fil** column heading in "Abaqus/Standard output variable identifiers," Section 4.2.1, or "Abaqus/Explicit output variable identifiers," Section 4.2.2, are available. For example, the U or S variable identifier keys can be specified, but the U1 or S11 variable identifier keys cannot be specified. In addition, the MODAL variable identifier key can be specified to gather frequency results (those written to the results file with the record key 1980); in this case no additional data are required in this command.

When gathering results from the output database (`.odb`) file, each result record to be gathered is chosen by specifying one of the available output variable identifier keys appearing under the **.odb** column heading in "Abaqus/Standard output variable identifiers," Section 4.2.1, or "Abaqus/Explicit output variable identifiers," Section 4.2.2. For field output the component must not be specified, while for history output the component number is required; for example, the U or S variable identifier keys can be specified for field output, while the U1 or S11 variable identifier keys can be specified for history output.

Optional data:

request

This option is applicable only if the results are to be gathered from the output database file.

Set **request** equal to FIELD or HISTORY to specify whether the results must be gathered from the field data or the history data in the output database file.

If **request** is omitted from this command, the results will be gathered from the field data.

step

Set **step** equal to the analysis step number from which the results are to be gathered.

If **step** is specified in this command as well as in the **output** command, the **step** specification in this command is used.

If **step** is omitted from this command, it must have been specified in the **output** command.

Optional and mutually exclusive data:

frameValue

This option is applicable only if the results are to be gathered from the output database file.

Set **frameValue** equal to the step time or frequency value of the analysis increment in the analysis step specified from which the results are to be gathered. **frameValue** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step. If no results are available at the **frameValue** specified, a warning will be issued and the results will be gathered from the closest increment.

If **frameValue** is specified in this command as well as in the **output** command, the **frameValue** specification in this command is used for gathering.

If **frameValue** is omitted from this command, the results are gathered for the **frameValue** specified in the **output** command or are gathered from the last increment in the step.

inc

Set **inc** equal to the number of the analysis increment of the non-frequency analysis step specified from which the results are to be gathered across the parametric study variations. **inc** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step.

If **inc** is specified in this command as well as in the **output** command, the **inc** specification in this command is used for gathering.

If **inc** is omitted from this command, the results are gathered from the increment specified in the **output** command or are gathered from the last increment in the step.

This option is not valid for gathering history results from the output database file.

mode

Set **mode** equal to the number of the mode of the frequency analysis step specified from which the results are to be gathered across the parametric study variations.

If **mode** is specified in this command as well as in the **output** command, the **mode** specification in this command is used.

If **mode** is omitted from this command, the results are gathered from the mode specified in the **output** command or are gathered from the first mode in the step.

Additional data for element integration point variables:

Required data:

element

Set **element** equal to the number of the element for which results are to be gathered.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Optional data:

centroid

Set **centroid** equal to the symbolic constant ON to indicate that the results are to be gathered at the centroid of the element. This option is valid only when the results have been written to the results file at the centroid of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

int

Set **int** equal to the number of the integration point of the element for which results are to be gathered. This option is valid only when the results have been written to the results file at the integration points of the element.

centroid, **int**, and **node** are mutually exclusive.

If **int** is omitted, the default is **int**=1. If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

node

Set **node** equal to the number of the node in the element for which results are to be gathered. This option is valid only when the results have been written to the results file at the nodes of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

rbnum

Set **rbnum** equal to the number of the rebar for which results are to be gathered. The rebar number is consistent with the order, per element, in which you define the rebar (see “Defining rebar as an element property,” Section 2.2.4).

If **rbnum** is omitted, the default is **rbnum**=1.

Rebar information cannot be gathered from the output database file.

rebar

Set **rebar** equal to the name of the rebar for which results are to be gathered (defined as described in "Defining rebar as an element property," Section 2.2.4). Rebar results can be obtained for continuum and beam elements only at integration points; for shell and membrane elements rebar results can be obtained at integration points and at the centroid of the element.

Rebar information cannot be gathered from the output database file.

section

Set **section** equal to the number of the section point of the element for which results are to be gathered. This applies to beam, shell, or layered solid elements. **section** is not relevant for rebar results.

If **section** is omitted, the default is **section**=1.

Additional data for element section variables:

Required data:

element

Set **element** equal to the number of the element for which results are to be gathered.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Optional data:

centroid

Set **centroid** equal to the symbolic token ON to indicate that the results are to be gathered at the centroid of the element. This option is valid only when the results have been written to the results file or output database at the centroid of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

int

Set **int** equal to the number of the integration point of the element for which results are to be gathered. This option is valid only when the results have been written to the results file or output database at the integration points of the element.

centroid, **int**, and **node** are mutually exclusive.

If **int** is omitted, the default is **int**=1. If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

node

Set **node** equal to the number of the node in the element for which results are to be gathered. This option is valid only when the results have been written to the results file or output database at the nodes of the element.

centroid, **int**, and **node** are mutually exclusive.

If **centroid**, **int**, and **node** are omitted, the default is **int**=1.

Additional data for whole element variables:

Required data:

element

Set **element** equal to the number of the element for which results are to be gathered.

int

Set **int** = –1.

instance

This option is required only if results are gathered for an element on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Additional data for partial model (element set) or whole model variables:

Optional data:

elset

Set **elset** equal to the element set name for which results are to be gathered. If **elset** is omitted, results will be gathered for the whole model. This name must be enclosed in matching quotation marks.

instance

This option is required only if results are gathered from an output database file and if the element set is defined on an instance.

Set **instance** equal to the name of the instance on which the element set is defined. This name must be enclosed in matching quotation marks.

If the element set is defined on the assembly, **instance** must not be specified in both this command and the **output** command.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Additional data for nodal variables:

Required data:

node

Set **node** equal to the number of the node for which results are to be gathered.

instance

This option is required only if results are gathered for a node on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Additional data for modal variables:

There are no additional data.

Additional data for contact surface variables:

Required data:

master

Set **master** equal to the name of the master surface of the contact pair for which results are to be gathered. This name must be enclosed in matching quotation marks.

slave

Set **slave** equal to the name of the slave surface of the contact pair for which results are to be gathered. This name must be enclosed in matching quotation marks.

Optional data:

masterInstance

This option is required only if results are gathered from an output database file and if the master surface is defined on an instance.

Set **masterInstance** equal to the name of the instance on which the master surface is defined. This name must be enclosed in matching quotation marks.

slaveInstance

This option is required only if results are gathered from an output database file and if the slave surface is defined on an instance.

Set **slaveInstance** equal to the name of the instance on which the slave surface is defined. This name must be enclosed in matching quotation marks.

Required data when slave surface node variable results are requested:

node

If slave surface node variable results are to be gathered, set **node** equal to the number of the node for which results are to be gathered.

nset and **node** are mutually exclusive.

instance

This option is required only if results are gathered for a node on a part instance in an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Optional data when whole surface variable results are requested:

nset

Set **nset** equal to the name of the node set for which whole surface variable results are to be gathered. This name must be enclosed in matching quotation marks.

If **nset** is omitted, the default is the whole surface.

If the results are collected from the output database file and the node set is defined on an instance, the node set name must be prefixed with the instance name and a period (for example: “PART–1–1.TOP”).

nset and **node** are mutually exclusive.

instance

This option is required only if results are gathered from an output database file and if the node set is defined on an instance.

Set **instance** equal to the name of the instance on which the node set is defined. This name must be enclosed in matching quotation marks.

If the node set is defined on the assembly, **instance** must not be specified in both this command and the **output** command.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Additional data for cavity radiation surface variables:

Required data:

element

Set **element** equal to the number of the element underlying the cavity facet for which results are to be gathered.

elface

Set **elface** equal to the face identifier of the face of the element underlying the cavity facet for which results are to be gathered.

instance

This option is required only if results are gathered from an output database file generated from a model described as an assembly of part instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If **instance** is specified in this command as well as in the **output** command, the **instance** specification in this command is used.

Additional data for section file output:

Required data:

sectionName

Set **sectionName** equal to the name of the section for which results are to be gathered. This name must be enclosed in matching quotation marks (see "Section output from Abaqus/Standard" in "Output to the data and results files," Section 4.1.2).

20.2.6 *aStudy*.generate(): Generate the analysis job data for a parametric study.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to generate the analysis input files for a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

***aStudy*.generate (template=)**

Required data:

template

Set **template** equal to the name of the template input file from which the input files of each of the parametric study variations are to be generated; this name must be enclosed in matching quotation marks.

20.2.7 *aStudy*.output(): Specify the source of parametric study results.

Products: Abaqus/Standard Abaqus/Explicit

This command should precede any results gather commands. It is used to specify from where the results of a parametric study will be gathered.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

***aStudy*.output (file=, instance=, overlay=, request=, step= , frameValue= | inc= | mode=)**

Optional data:

file

Set **file** equal to FIL or ODB to specify that results must be read from the results (`.fil`) file or output database (`.odb`) file.

If **file** is omitted from this command, the results will be read from the results (`.fil`) file.

instance

This option is applicable only if results are gathered from an output (`.odb`) database containing multiple instances.

Set **instance** equal to the name of the part instance for which results are to be gathered. This name must be enclosed in matching quotation marks.

If results are gathered for a set defined on the assembly, **instance** must not be specified in both this command and the **gather** command.

If **instance** is omitted from this command and an instance name is required to gather the results, it must be specified in the **gather** command.

overlay

This option is applicable only if results are gathered from an output (`.odb`) database.

This option is used to control the trade-off between memory usage and execution speed. Set **overlay** equal to OFF (the default) or ON to specify whether the memory used for an output database should be overwritten. If memory usage is a problem, set **overlay** equal to ON to specify that the memory used for an output database must be overwritten after a **gather** is performed for the specific output database. If execution speed is more important than memory usage, set **overlay** equal to OFF to specify that memory must be allocated for each output database.

The **overlay** option affects your ability to separately manipulate the output database (`.odb`) file. When you set **overlay** equal to OFF, the file will remain open after accessing the results from the **gather** command, which may affect your ability to separately access or manipulate the file (to delete the file, for

example). Set **overlay** equal to ON to separately access the **.odb** file after accessing the **gather** based results.

request

This option is applicable only if the results are to be gathered from the output (**.odb**) database.

Set **request** equal to FIELD or HISTORY to specify whether the results must be gathered from the field data or the history data in the output database.

If **request** is omitted from this command, the results will be gathered from the field data.

step

Set **step** equal to the analysis step number from which the results are to be gathered.

If **step** is omitted from this command, it must be specified in the **gather** command.

Optional and mutually exclusive data:

frameValue

This option is applicable only if the results are to be gathered from the output (**.odb**) database.

Set **frameValue** equal to the step time or frequency value of the analysis increment in the analysis step specified from which the results are to be gathered. **frameValue** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step. If no results are available at the **frameValue** specified, a warning will be issued and the results will be gathered from the closest increment.

If **frameValue** is omitted from this command, the results are gathered from the last increment in the step or are gathered from the increment specified in the **gather** command.

inc

Set **inc** equal to the number of the analysis increment of the non-frequency analysis step specified from which the results are to be gathered. **inc** can also be set equal to the symbolic constant LAST to specify that results are to be gathered from the last increment of the step specified.

If **inc** is omitted from this command, the results are gathered from the last increment in the step or are gathered from the increment specified in the **gather** command.

This option is not valid for gathering history results from the output (**.odb**) database.

mode

Set **mode** equal to the mode number of the frequency analysis step specified from which the results are to be gathered across the parametric study variations.

If **mode** is omitted, the results are gathered from the mode specified in the **gather** command or are gathered from the first mode in the step.

20.2.8 *aStudy*=ParStudy(): Create a parametric study.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to create a parametric study. It must precede any other scripting commands that refer to the parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

***aStudy*=ParStudy (par= , name= , verbose= , directory=)**

Required data:

par

Set **par** equal to the sequence of independent input parameters selected for the parametric study. This sequence must be given inside parentheses or brackets and must contain independent parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`. If only one parameter is to be listed, its name can be given enclosed by matching quotation marks; for example, `'par1'`.

Optional data:

name

Set **name** equal to the name of the parametric study; the name must be enclosed in matching quotation marks. If a name is not specified, its value defaults to the name of the Python script file that contains the parametric study commands.

verbose

Set **verbose** equal to the symbolic token OFF to suppress the printing of comment and warning messages. The default is **verbose**=ON.

directory

Set **directory** equal to the symbolic token ON to select that subdirectories of the current directory are used to organize the files of the parametric study. A subdirectory will be created for each design that is analyzed. The default is **directory**=OFF.

20.2.9 *aStudy*.report(): Report parametric study results.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to report results gathered across the designs of a parametric study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.report (token*, **results= , par= , designSet= , variations=, truncation= ,** *additional data***)**

Tokens:

FILE

Use this token to specify that results are to be written to an ASCII file as a table with the relevant headings.

PRINT

Use this token to specify that results are to be printed as a table with the relevant headings. Since the results are printed to the default output device (the screen), you may wish to limit the number of columns in a table so as to make the table readable.

XYPLOT

Use this token to specify that results are to be written to an ASCII file as a table without headings. This table can subsequently be read by the Visualization module in Abaqus/CAE to display *X*–*Y* plots of result and parameter values.

Required data:

results

Set **results** equal to the sequence of result names to be reported; this sequence must be enclosed by parentheses or brackets. For example, **`results=('e33_sinv.1', 'e52_strain', 'n25_u.3')`**, where **`'e33_sinv.1'`** is the Mises stress of element 33 (Mises is the first component of the SINV record), **`'e52_strain'`** are all the strain components of element 52, and **`'n25_u.3'`** is the third component of displacement of node 25. This example assumes that the three results above

were gathered in previous **gather** commands by requesting the SINV, E, and U variable identifier keys, respectively.

Optional data:

par

Set **par** equal to the name of the parameter or the sequence of parameters to be included in the report table. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

If **par** is omitted, all parameters in the parametric study are included in the report table.

designSet

Set **designSet** equal to the name of the design set whose results are to be included in the report table; this name must be enclosed by matching quotation marks.

If **designSet** is omitted, results for all design sets in the parametric study are included in the report table.

variations

Set **variations** equal to ON to indicate that the first column of the report table must show the names of the designs being reported. Set **variations** equal to OFF to indicate that the names of the designs being reported are not to be given in the first column of the report table.

If **variations** is omitted, the column of design names is not included in the report table.

truncation

Set **truncation** equal to ON to indicate that the data of the report table must be reported with limited precision. Set **truncation** equal to OFF to indicate that the data of the report table must be reported with full precision.

If **truncation** is omitted, the data of the report table are reported with limited precision.

Additional data for FILE and XYPLOT:

Required data:

file

Set **file** equal to the name of the file to which the report table is to be written. The file name must be enclosed by matching quotation marks.

20.2.10 *aStudy*.sample(): Sample parameters for parametric studies.

Products: Abaqus/Standard Abaqus/Explicit

This command is used to create samples of the values of the parameters of the study.

Reference:

- "Scripting parametric studies," Section 20.1.1

Command:

aStudy*.sample (token*, *additional data*)

Tokens:

INTERVAL

Use this token to sample a parameter at equal intervals.

NUMBER

Use this token to sample a given number of values of a parameter.

PRINT

Use this token to print parameter samples.

REFERENCE

Use this token to sample parameter values specified with respect to a reference parameter value.

VALUES

Use this token to sample particular values of a parameter.

Additional data for INTERVAL:

Required data:

interval

Set **interval** equal to the sampling interval. For a continuous valued parameter, values are sampled at equally spaced intervals based on the numerical value of the parameter. For a discrete valued parameter, values are sampled at equally spaced intervals based on the indexing of the sequence of parameter values.

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and

must contain parameter names enclosed by matching quotation marks and separated by commas; for example, **`('par1', 'par2', 'par3')`** or **`['par1', 'par2', 'par3']`**.

Optional data:

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, **`(10., 20.)`** or **`[10., 20.]`**. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, **`(1., 2., 5., 3.)`** or **`[1., 2., 5., 3.]`**.

If **domain** is specified in this command as well as in the **define** command for this parameter, the **domain** specification in this command is used for sampling.

If **domain** is omitted from this command, it must have been specified in the **define** command.

Additional data for NUMBER:

Required data:

number

Set **number** equal to the number of equally spaced values to be sampled for the parameter.

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, **`'par1'`**. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, **`('par1', 'par2', 'par3')`** or **`['par1', 'par2', 'par3']`**.

Optional data:

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, **`(10., 20.)`** or **`[10., 20.]`**. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, **`(1., 2., 5., 3.)`** or **`[1., 2., 5., 3.]`**.

If **domain** is specified in this command as well as in the **define** command for this parameter, the **domain** specification in this command is used for sampling.

If **domain** is omitted from this command, it must have been specified in the **define** command.

Additional data for PRINT:

Optional data:

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

If **par** is omitted, parameter samplings are printed for all parameters in the parametric study.

Additional data for REFERENCE:

Required data:

interval

Set **interval** equal to the sampling interval. For a continuous valued parameter, values are sampled at equally spaced intervals about the reference value based on the numerical value of the parameter. For a discrete valued parameter, values are sampled at equally spaced intervals about the reference value based on the indexing of the sequence of parameter values.

numSymPairs

Set **numSymPairs** equal to the number of pairs of parameter values to be sampled symmetrically about the reference value of the parameter.

par

Set **par** equal to the name of the parameter being sampled. This name must be enclosed by matching quotation marks; for example, `'par1'`.

Optional data:

domain

For a continuous valued parameter, set **domain** equal to the minimum and maximum values of the parameter separated by a comma and enclosed by parentheses or brackets; for example, `(10., 20.)` or `[10., 20.]`. For a discrete valued parameter, set **domain** equal to the sequence of values that the parameter may have separated by commas and enclosed by parentheses or brackets; for example, `(1., 2., 5., 3.)` or `[1., 2., 5., 3.]`.

If **domain** is specified in this command as well as in the **define** command for this parameter, the **domain** specification in this command is used for sampling.

In the case of a discrete valued parameter if **domain** is omitted from this command, it must have been specified in the **define** command.

reference

For a continuous valued parameter, set **reference** equal to the reference value of the parameter. For discrete valued parameters, set **reference** equal to the index in the sequence of parameter values; indexing starts at zero, so that the first value of the sequence corresponds to index zero and the last value of the sequence corresponds to an index equal to the number of values in the sequence minus one.

If **reference** is specified in this command as well as in the **define** command for this parameter, the **reference** specification in this command is used for sampling.

If **reference** is omitted from this command, it must have been specified in the **define** command.

Additional data for VALUES:

Required data:

par

Set **par** equal to the name of the parameter or the sequence of parameters whose samples are to be printed. If a single parameter is specified, it must be enclosed by matching quotation marks; for example, `'par1'`. If a sequence of parameters is specified, it must be given inside parentheses or brackets and must contain parameter names enclosed by matching quotation marks and separated by commas; for example, `('par1', 'par2', 'par3')` or `['par1', 'par2', 'par3']`.

values

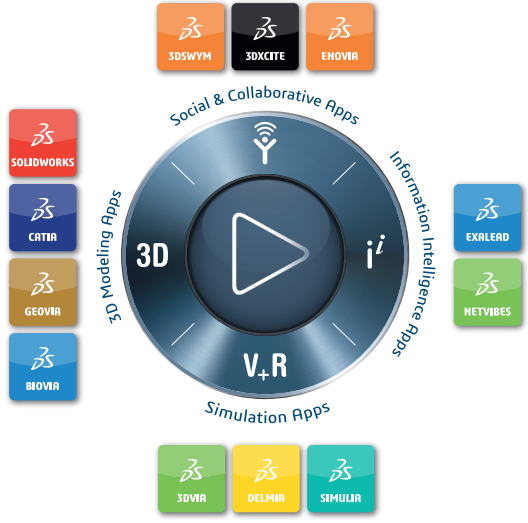
Set **values** equal to the sequence of parameter values that constitute the sample. This sequence must be given inside parentheses or brackets and must contain values separated by commas; for example, `('CAX4', 'CAX4R', 'CAX4H')` or `[10., 20., 40.]`.

About SIMULIA

Dassault Systèmes SIMULIA applications, including Abaqus, Isight, Tosca, and Simulation Lifecycle Management, enable users to leverage physics-based simulation and high-performance computing to explore real-world behavior of products, nature, and life. As an integral part of Dassault Systèmes' **3D**EXPERIENCE platform, SIMULIA applications accelerate the process of making highly informed, mission-critical design and engineering decisions before committing to costly and time-consuming physical prototypes. **www.3ds.com/simulia**

Our **3D**EXPERIENCE Platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes, the **3D**EXPERIENCE Company, provides business and people with virtual universes to imagine sustainable innovations. Its world-leading solutions transform the way products are designed, produced, and supported. Dassault Systèmes' collaborative solutions foster social innovation, expanding possibilities for the virtual world to improve the real world. The group brings value to over 170,000 customers of all sizes in all industries in more than 140 countries. For more information, visit **www.3ds.com**.



DASSAULT SYSTEMES | The **3D**EXPERIENCE® Company

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes K.K.
ThinkPark Tower
2-1-1 Osaki, Shinagawa-ku,
Tokyo 141-6020
Japan

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

©2015 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the Compass icon and the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3D VIA, BIOVIA, NETVIBES, and 3DXCITE are commercial trademarks or registered trademarks of Dassault Systèmes or its subsidiaries in the U.S. and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.