SANDIA REPORT

SAND2011-7597
Unlimited Release
Printed October 2011

Sierra/SolidMechanics 4.22 User's Guide

SIERRA Solid Mechanics Team
Computational Solid Mechanics and Structural Dynamics Department
Engineering Sciences Center

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

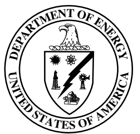
Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online



SAND2011-7597
Unlimited Release
Printed October 2011

Sierra/SolidMechanics 4.22 User's Guide

SIERRA Solid Mechanics Team
Computational Solid Mechanics and Structural Dynamics Department
Engineering Sciences Center
Sandia National Laboratories
Box 5800
Albuquerque, NM 87185-0380

Abstract

Sierra/SolidMechanics (Sierra/SM) is a Lagrangian, three-dimensional code for the analysis of solids and structures. It provides capabilities for explicit dynamic and implicit quasistatic and dynamic analyses. The explicit dynamics capabilities allow for the efficient and robust solution of models subjected to large, suddenly applied loads. For implicit problems, Sierra/SM uses a multi-level iterative solver, which enables it to effectively solve problems with large deformations, nonlinear material behavior, and contact. Sierra/SM has a versatile library of continuum and structural elements, and an extensive library of material models. The code is written for parallel computing environments, and it allows for scalable solutions of very large problems for both implicit and explicit analyses. It is built on the SIERRA Framework, which allows for coupling with other SIERRA mechanics codes. This document describes the functionality and input structure for Sierra/SM.

Acknowledgments

This document is the result of the collective effort of a number of individuals. The current development team responsible for Adagio and Presto, the SIERRA Solid Mechanics codes, includes Nathan K. Crane, Martin W. Heinstein, Alex J. Lindblad, David J. Littlewood, Kyran D. Mish, Kendall H. Pierson, Vicki L. Porter, Nathaniel S. Roehrig, Timothy R. Shelton, Gregory D. Sjaardema, Benjamin W. Spencer, Jesse D. Thomas, and Michael G. Veilleux. This document is written and maintained by this team.

Outside the core development team, there are number of other individuals who have contributed to this manual. Nicole L. Breivik, J. Franklin Dempsey, Jeffery D. Gruda, and Chi S. (David) Lo have provided valuable input from the user community as Product Managers.

Many others have contributed to this document, either directly or indirectly. These include, but are not limited to Manoj K. Bhardwaj, James V. Cox, Jason D. Hales, Arne S. Gullerud, Daniel C. Hammerand, J. Richard Koteras, Jakob T. Ostien, Rhonda K. Reinert, William M. Scherzinger, and Gerald W. Wellman.

Sierra/SM 4.22 Release Notes

Following is a list of new features and syntax changes made to Sierra/SM since the 4.20 release.

The separate Adagio and Presto manuals have been discontinued in favor of a unified Sierra/SM manual that documents features for both explicit and implicit analyses. See Section 1.3.7.

Piecewise analytic functions can now be defined. See Section 2.1.5.

Default values have been defined and documented for many of the implicit solver commands. See Chapter 3.5.

The solver will now report convergence if the residual norm reaches a value that is approximately zero to within the machine precision tolerance for the problem. This convergence will occur even if the relative residual is not reached and can allow correct handling of load-steps with nominally zero external load. See Section 4.2.1.

Solver convergence can now be based on an energy-based residual and residual reference. These energy norms are useful for evaluating the residual for problems that include combinations of forces and moments. See Section 4.2.1.

New options are provided to control the FETI solver's linear system conditioning to improve CG convergence. See Section 4.3.2.

Thermal strain calculations have been added to one-dimensional elements. See Section 5.1.3.1.

A fiber shell material model that adds plate-bending response to the existing fiber membrane model has been implemented and documented. See Section 5.2.23.

All Strumento material models have either been replaced by models in the LAME library or, in a few cases, removed. As a result, the `USE STRUMENTO|LAME` command has been removed. See Section 6.2.

Documentation has been added for fully-integrated solid elements. See Section 6.2.1.

An analytic, through-thickness integration scheme is now available for elastic shells. See Section 6.2.4.

The `SHELL DRILLING STIFFNESS` command has been removed from the `FULL TANGENT PRECONDITIONER` command block. It is now recommended to specify physical drilling stiffness in the shell section. See Section 6.2.4.

An option has been added to the element death capability to kill elements based on proximity to other nodes or elements. See Section 6.5.4.8

Parts of the input mesh can now be translated or rotated at the beginning of the run through commands in the input file. See Section 7.2.

Documentation has been added for periodic boundary conditions. See Section 7.4.5.

Pressure boundary conditions can now be applied to elements that have been converted to particles. See Section 7.6.1.

A velocity damping coefficient can now be applied to pressure boundary conditions. See Section 7.6.1.2.

Contact surfaces can now be subdivided based on normal directions. See Section 8.2.4

Examples on how contact surfaces are defined using block skinning, side sets, lofted geometry, and element death have been added. See Section 8.2.7.

Documentation has been added for general analytic rigid contact surfaces. See Section 8.2.8.1.

A velocity-dependent Coulomb friction model is now available for explicit dynamics. See Section 8.3.0.6.

The lofted radius of beam elements for contact can now optionally be specified. Additionally, the shapes of lofted beams and the shapes and radii of lofted particles can be specified by element block. See Section 8.5.8.

The name of the `SHELL LOFTING` command block for contact has been changed to `SURFACE OPTIONS`. See Section 8.5.8.

The `ACTIVE PERIODS` command can now be used to selectively enable contact by time period. See Section 8.6.

Additional user output options for operating on integration point quantities are now available. For example, element quantities for output can be computed based on an average of integration point quantities. See Section 9.2.2.2.

Variables can now be interpolated to face variables on a different mesh. In addition, the time derivative of an interpolated variable can also be computed. See Section 9.7.

The global angular momentum of a problem can now be requested as a output variable. See Section 9.9.

The tables of state variable output from material models have been updated to include more complete listings for many models. See Section 9.9.2.

Sierra/SM 4.22 Known Issues

Section 3.4.1.2: When using explicit control modes, the Lanczos and power method time step estimators can not yet be used with problems that have contact, rigid bodies, blocks in the fine mesh that are not controlled by the coarse mesh, or coarse elements that contain no fine nodes.

Section 4.3: Deactivation of element blocks (see Section 6.1.5.9) does not currently work in conjunction with the full tangent preconditioner in Adagio. To use this capability, one of the nodal preconditioners must be used.

Section 6.2.1: For problems with large rotations, the hourglass energies are known to spike exponentially with increases in total rotation. This behavior is observed for both midpoint and strongly objective strain incrementation, and for both incremental and total hourglass formulations. The large rotation issue is a known limitation in solid mechanics for all non-hyperelastic material models.

Section 6.2.11: For problems with rotations, the sph algorithm generates rotational spring like forces on the boundary because of known deficiencies in the algorithm in distinguishing between rigid body rotation and deformation. The sph algorithm does not conserve rotational or angular momentum.

Section 6.2.12: Superelements are not compatible with several modeling capabilities. They cannot be used with element death. They cannot be used with node-based, power method, or Lanczos critical time step estimation methods. They are also not compatible with some preconditioners (such as FETI) for implicit solutions.

Section 7.4.2.2: If a prescribed displacement with the `CYLINDRICAL AXIS` option is applied to nodes that fall on the axis, it will have no effect. Separate boundary conditions should be applied to those nodes to fix them in the plane normal to the axis.

Section 7.4.3.2: If a prescribed velocity with the `CYLINDRICAL AXIS` option is applied to nodes that fall on the axis, it will have no effect. Separate boundary conditions should be applied to those nodes to fix them in the plane normal to the axis.

Section 8.5.9: Attempting to use `GLOBAL SEARCH INCREMENT` with a value greater than 1, especially in a problem that contains shells and/or restart, will, in most cases, cause code failure. A `GLOBAL SEARCH INCREMENT` value greater than 1 will, under the best circumstances, give only a marginal improvement in speed.

Section 9.4: User defined variables (see Section 11.2.4) are not currently supported with heartbeat output.

Section 10.2: Currently, the J-Integral evaluation capability is based on assumptions of elastostatics and a stationary crack, and is only implemented for uniform gradient hex elements.

Contents

1 Introduction **37**
1.1 Document Overview . . . 38
1.2 Overall Input Structure . . . 40
1.3 Conventions for Command Descriptions . . . 43
1.3.1 Key Words . . . 43
1.3.2 User-Specified Input . . . 43
1.3.3 Optional Input . . . 44
1.3.4 Default Values . . . 44
1.3.5 Multiple Options for Values . . . 44
1.3.6 Known Issues and Warnings . . . 45
1.3.7 Commands Applicable to Only Explicit or Implicit Analyses . . . 45
1.4 Style Guidelines . . . 47
1.4.1 Comments . . . 47
1.4.2 Continuation Lines . . . 47
1.4.3 Case . . . 47
1.4.4 Commas and Tabs . . . 47
1.4.5 Blank Spaces . . . 48
1.4.6 General Format of the Command Lines . . . 48
1.4.7 Delimiters . . . 49
1.4.8 Order of Commands . . . 49
1.4.9 Abbreviated END Specifications . . . 49
1.4.10 Indentation . . . 50
1.4.11 Including Files . . . 50
1.5 Exodus II Database Naming Conventions . . . 51

1.6 Major Scope Definitions for an Input File 52
1.7 Input/Output Files 54
1.8 Obtaining Support 56
1.9 References 57

2 General Commands 59
2.1 SIERRA Scope 59
2.1.1 SIERRA Command Block 59
2.1.2 Title 60
2.1.3 Restart Control 60
2.1.3.1 Restart Time 61
2.1.3.2 Automatic Restart 61
2.1.4 User Subroutine Identification 61
2.1.5 Functions 62
2.1.6 Axes, Directions, and Points 67
2.1.7 Orientation 69
2.1.8 Coordinate System Block Command 73
2.1.9 Define Coordinate System Line Command 75
2.2 Procedure and Region 76
2.2.1 Procedure 77
2.2.2 Time Control 77
2.2.2.1 Explicit Dynamic Time Control 77
2.2.2.2 Implicit Time Control 78
2.2.3 Region 78
2.3 Use Finite Element Model 80
2.4 Element Distortion Metrics 81
2.5 Activation/Deactivation of Functionality 83
2.6 Error Recovery 83
2.7 Manual Job Control 84

3 Explicit Dynamic Time Step Control 86
3.1 Procedure Time Control 87
3.1.1 Command Blocks for Time Control and Time Stepping 89

3.1.2 Initial Time Step 91
3.1.3 Time Step Scale Factor 91
3.1.4 Time Step Increase Factor 91
3.1.5 Step Interval 91
3.1.6 Example 92
3.2 Other Critical Time Step Methods 94
3.2.1 Lanczos Method 95
3.2.1.1 Lanczos Method with Constant Time Steps 96
3.2.1.2 Controls for Lanczos Method 99
3.2.1.3 Scale Factor for Lanczos Method 100
3.2.1.4 Accuracy of Eigenvalue Estimate 101
3.2.1.5 Lanczos Parameters Command Block 102
3.2.2 Power Method 105
3.2.2.1 Power Method with Constant Time Steps 106
3.2.2.2 Controls for Power Method 107
3.2.2.3 Scale Factor for Power Method 108
3.2.2.4 Accuracy of Eigenvalue Estimate 108
3.2.2.5 Power Method Parameters Command Block 109
3.2.3 Node-Based Method 111
3.2.3.1 Node-Based Parameters Command Block 112
3.3 Mass Scaling 113
3.3.1 What is Mass Scaling? 113
3.3.2 Mass Scaling Command Block 114
3.3.3 Node Set Commands 115
3.3.3.1 Mass Scaling Commands 115
3.3.3.2 Additional Commands 116
3.4 Explicit Control Modes 116
3.4.1 Control Modes Region 117
3.4.1.1 Model Setup Commands 118
3.4.1.2 Time Step Control Commands 119
3.4.1.3 Mass Scaling Commands 120
3.4.1.4 Damping Commands 121

3.4.1.5 Kinematic Boundary Condition Commands 121
3.4.1.6 Output Commands 121
3.5 References 123

4 Implicit Solver, Time Step, and Dynamics 124
4.1 Multilevel Solver 126
4.2 Conjugate Gradient Solver 130
4.2.1 Convergence Commands 132
4.2.2 Preconditioner Commands 134
4.2.3 Line Search Command 137
4.2.4 Diagnostic Output Commands 138
4.2.5 CG Algorithm Commands 139
4.3 Full Tangent Preconditioner 141
4.3.1 Solver Selection Commands 142
4.3.2 Matrix Formation Commands 143
4.3.3 Reset and Iteration Commands 145
4.3.4 Fall-Back Strategy Commands 147
4.4 FETI Equation Solver 149
4.4.1 Convergence Commands 150
4.4.2 Memory Usage Commands 150
4.4.3 Solver Commands 151
4.5 Control Contact 153
4.5.1 Convergence Commands 157
4.5.2 Level Selection Command 158
4.5.3 Diagnostic Output Commands 159
4.5.4 Augmented Lagrange Enforcement Commands 159
4.5.4.1 Augmented Lagrange Adaptive Penalty Commands 160
4.5.4.2 Augmented Lagrange Tolerance and Convergence Options . . . 160
4.5.4.3 Augmented Lagrange Miscellaneous Options 161
4.6 Control Stiffness 162
4.6.1 Convergence Commands 168
4.6.2 Level Selection Command 170
4.6.3 Diagnostic Output Commands 171

4.7 Control Failure 172
4.7.1 Convergence Command 172
4.7.2 Level Selection Command 173
4.7.3 Diagnostic Output Commands 173
4.8 Control Modes 174
4.8.1 Control Modes Region 175
4.8.1.1 Model Setup Commands 176
4.8.1.2 Solver Commands 176
4.8.1.3 Kinematic Boundary Condition Commands 177
4.9 Predictors 178
4.9.1 Loadstep Predictor 178
4.9.1.1 Predictor Type 178
4.9.1.2 Scale Factor 179
4.9.1.3 Slip Scale Factor 179
4.9.2 Level Predictor 180
4.10 JAS3D Compatibility Mode 181
4.11 Time Step Control 182
4.11.1 Command Blocks for Time Control and Time Stepping 182
4.11.1.1 Time Increment 184
4.11.1.2 Number of Time Steps 184
4.11.1.3 Time Increment Function 184
4.11.2 Adaptive Time Stepping 185
4.11.2.1 Method 186
4.11.2.2 Target Iterations 186
4.11.2.3 Iteration Window 186
4.11.2.4 Cutback Factor 187
4.11.2.5 Growth Factor 187
4.11.2.6 Maximum Failure Cutbacks 187
4.11.2.7 Maximum Multiplier 187
4.11.2.8 Minimum Multiplier 188
4.11.2.9 Reset at New Period 188
4.11.2.10 Active Periods 188

4.11.3 Time Control Example 189
4.12 Implicit Dynamic Time Integration 190
4.12.1 Implicit Dynamics 190
4.12.1.1 Active Periods 190
4.12.1.2 Use HHT Integration 191
4.12.1.3 HHT Parameters 191
4.12.1.4 Implicit Dynamic Adaptive Time Stepping 191
4.13 References 193

5 Materials 194

5.1 General Material Commands 198
5.1.1 Density Command 198
5.1.2 Biot's Coefficient Command 198
5.1.3 Thermal Strain Behavior 198
5.1.3.1 Defining Thermal Strains 199
5.1.3.2 Activating Thermal Strains 201
5.2 Material Models 202
5.2.1 Elastic Model 202
5.2.2 Thermoelastic Model 204
5.2.3 Neo-Hookean Model 206
5.2.4 Elastic Fracture Model 208
5.2.5 Elastic-Plastic Model 210
5.2.6 Elastic-Plastic Power-Law Hardening Model 212
5.2.7 Ductile Fracture Model 214
5.2.8 Multilinear EP Hardening Model 216
5.2.9 Multilinear EP Hardening Model with Failure 218
5.2.10 Johnson-Cook Model 221
5.2.11 BCJ Model 223
5.2.12 Power Law Creep 225
5.2.13 Soil and Crushable Foam Model 227
5.2.14 Karagozian and Case Concrete Model 230
5.2.15 Foam Plasticity Model 233
5.2.16 Low Density Foam Model 236

5.2.17 Elastic Three-Dimensional Orthotropic Model 237
5.2.18 Wire Mesh Model 240
5.2.19 Orthotropic Crush Model 242
5.2.20 Orthotropic Rate Model 245
5.2.21 Elastic Laminate Model 248
5.2.22 Fiber Membrane Model 251
5.2.23 Fiber Shell Model 254
5.2.24 Incompressible Solid Model 256
5.2.25 Mooney-Rivlin Model 259
5.2.26 NLVE 3D Orthotropic Model 262
5.2.27 Stiff Elastic 266
5.2.28 Swanson Model 268
5.2.29 Viscoelastic Swanson Model 271
5.3 Cohesive Zone Material Models 275
5.3.1 Traction Decay 275
5.3.2 Tvergaard Hutchinson 276
5.3.3 Thouless Parmigiani 278
5.3.4 Compliant Joint 280
5.4 References 281

6 Elements **283**
6.1 Finite Element Model 284
6.1.1 Identification of Mesh File 287
6.1.2 Alias 288
6.1.3 Omit Block 288
6.1.4 Component Separator Character 288
6.1.5 Descriptors of Element Blocks 289
6.1.5.1 Material Property 291
6.1.5.2 Include All Blocks 291
6.1.5.3 Remove Block 292
6.1.5.4 Section 292
6.1.5.5 Linear and Quadratic Bulk Viscosity 292
6.1.5.6 Hourglass Control 292

6.1.5.7 Effective Moduli Model . . . 294
6.1.5.8 Element Numerical Formulation . . . 295
6.1.5.9 Activation/Deactivation of Element Blocks by Time . . . 296
6.2 Element Sections . . . 297
6.2.1 Solid Section . . . 297
6.2.2 Cohesive Section . . . 300
6.2.3 Localization Section . . . 301
6.2.4 Shell Section . . . 302
6.2.5 Membrane Section . . . 308
6.2.6 Beam Section . . . 312
6.2.7 Truss Section . . . 317
6.2.8 Spring Section . . . 318
6.2.9 Damper Section . . . 319
6.2.10 Point Mass Section . . . 320
6.2.11 Particle Section . . . 322
6.2.12 Superelement Section . . . 326
6.2.12.1 Input Commands . . . 327
6.2.13 Peridynamics Section . . . 329
6.3 Element-like Functionality . . . 332
6.3.1 Rigid Body . . . 332
6.3.1.1 Multiple Rigid Bodies from a Single Block . . . 336
6.3.2 Torsional Spring Mechanism . . . 337
6.4 Mass Property Calculations . . . 341
6.4.1 Block Set Commands . . . 341
6.4.2 Structure Command . . . 342
6.5 Element Death . . . 343
6.5.1 Block Set Commands . . . 345
6.5.2 Criterion Commands . . . 346
6.5.2.1 Nodal Variable Death Criterion . . . 346
6.5.2.2 Element Variable Death Criterion . . . 347
6.5.2.3 Global Death Criterion . . . 348
6.5.2.4 Always Death Criterion . . . 349

6.5.2.5 Material Death Criterion 349
6.5.2.6 Subroutine Death Criterion 350
6.5.3 Evaluation Commands 351
6.5.4 Miscellaneous Option Commands 351
6.5.4.1 Summary Output Commands 351
6.5.4.2 Death on Inversion 352
6.5.4.3 Death on Ill-defined Contact 352
6.5.4.4 Death Steps 352
6.5.4.5 Degenerate Mesh Repair 353
6.5.4.6 Aggressive Contact Cleanup 353
6.5.4.7 Death Method 353
6.5.4.8 Death on Proximity 354
6.5.4.9 Particle Conversion 355
6.5.4.10 Active Periods 357
6.5.5 Cohesive Zone Setup Commands 357
6.5.6 Example 358
6.5.7 Element Death Visualization 358
6.6 Particle Embedding 360
6.7 Explicitly Computing Derived Quantities 361
6.8 Mesh Rebalancing 362
6.8.1 Rebalance 362
6.8.1.1 Rebalance Command Lines 363
6.8.1.2 Zoltan Command Line 363
6.8.2 Zoltan Parameters 365
6.9 Remeshing 366
6.9.1 Remeshing Commands 367
6.9.2 Remesh Block Set 369
6.9.3 Adaptive Refinement 370
6.9.3.1 Adaptive Refinement Control Commands 371
6.9.3.2 Tool Mesh Entity Commands 372
6.9.3.3 Activation Commands 372
6.10 References 373

7 Boundary Conditions and Initial Conditions 375

7.1 General Boundary Condition Concepts . . . 376

7.1.1 Mesh-Entity Assignment Commands . . . 376

7.1.2 Methods for Specifying Boundary Conditions . . . 378

7.2 Initial Mesh Modification . . . 379

7.3 Initial Variable Assignment . . . 380

7.3.1 Mesh-Entity Set Commands . . . 381

7.3.2 Variable Identification Commands . . . 382

7.3.3 Specification Command . . . 382

7.3.4 Weibull Probability Distribution Commands . . . 382

7.3.5 External Mesh Database Commands . . . 384

7.3.6 User Subroutine Commands . . . 385

7.3.7 Additional Command . . . 385

7.4 Kinematic Boundary Conditions . . . 387

7.4.1 Fixed Displacement Components . . . 387

7.4.1.1 Node Set Commands . . . 388

7.4.1.2 Specification Commands . . . 388

7.4.1.3 Additional Commands . . . 388

7.4.2 Prescribed Displacement . . . 389

7.4.2.1 Node Set Commands . . . 390

7.4.2.2 Specification Commands . . . 390

7.4.2.3 User Subroutine Commands . . . 392

7.4.2.4 External Mesh Database Commands . . . 392

7.4.2.5 Additional Commands . . . 393

7.4.3 Prescribed Velocity . . . 395

7.4.3.1 Node Set Commands . . . 396

7.4.3.2 Specification Commands . . . 396

7.4.3.3 User Subroutine Commands . . . 398

7.4.3.4 External Mesh Database Commands . . . 398

7.4.3.5 Additional Commands . . . 399

7.4.4 Prescribed Acceleration . . . 400

7.4.4.1 Node Set Commands . . . 401

7.4.4.2 Specification Commands 401
7.4.4.3 User Subroutine Commands 402
7.4.4.4 External Mesh Database Commands 403
7.4.4.5 Additional Commands 404
7.4.5 Periodic Boundary Condition 405
7.4.6 Fixed Rotation 407
7.4.6.1 Node Set Commands 407
7.4.6.2 Specification Commands 408
7.4.6.3 Additional Commands 408
7.4.7 Prescribed Rotation 409
7.4.7.1 Node Set Commands 410
7.4.7.2 Specification Commands 411
7.4.7.3 User Subroutine Commands 411
7.4.7.4 External Mesh Database Commands 412
7.4.7.5 Additional Commands 413
7.4.8 Prescribed Rotational Velocity 414
7.4.8.1 Node Set Commands 415
7.4.8.2 Specification Commands 416
7.4.8.3 User Subroutine Commands 417
7.4.8.4 External Mesh Database Commands 417
7.4.8.5 Additional Commands 418
7.4.9 Reference Axis Rotation 420
7.4.9.1 Block Command 421
7.4.9.2 Specification Commands 421
7.4.9.3 Rotation Commands 422
7.4.9.4 Torque Command 422
7.4.9.5 Additional Commands 422
7.4.10 Subroutine Usage for Kinematic Boundary Conditions 423
7.5 Initial Velocity Conditions 424
7.5.1 Node Set Commands 425
7.5.2 Direction Specification Commands 425
7.5.3 Angular Velocity Specification Commands 426

7.5.4 User Subroutine Commands . . . 426
7.6 Force Boundary Conditions . . . 428
7.6.1 Pressure . . . 428
7.6.1.1 Surface Set Commands . . . 429
7.6.1.2 Specification Commands . . . 430
7.6.1.3 User Subroutine Commands . . . 430
7.6.1.4 External Pressure Sources . . . 431
7.6.1.5 Output Command . . . 432
7.6.1.6 Additional Commands . . . 432
7.6.2 Traction . . . 434
7.6.2.1 Surface Set Commands . . . 435
7.6.2.2 Specification Commands . . . 435
7.6.2.3 User Subroutine Commands . . . 436
7.6.2.4 Additional Commands . . . 437
7.6.3 Prescribed Force . . . 438
7.6.3.1 Node Set Commands . . . 439
7.6.3.2 Specification Commands . . . 439
7.6.3.3 User Subroutine Commands . . . 440
7.6.3.4 Additional Commands . . . 441
7.6.4 Prescribed Moment . . . 442
7.6.4.1 Node Set Commands . . . 443
7.6.4.2 Specification Commands . . . 443
7.6.4.3 User Subroutine Commands . . . 444
7.6.4.4 Additional Commands . . . 445
7.7 Gravity . . . 446
7.8 Centripetal Force . . . 448
7.9 Prescribed Temperature . . . 449
7.9.1 Block Set Commands . . . 450
7.9.2 Specification Command . . . 450
7.9.3 User Subroutine Commands . . . 451
7.9.4 External Mesh Database Commands . . . 451
7.9.5 Coupled Analysis Commands . . . 453

7.9.6 Additional Commands 453
7.10 Pore Pressure 454
7.10.1 Block Set Commands 455
7.10.2 Specification Command 455
7.10.3 User Subroutine Commands 455
7.10.4 External Mesh Database Commands 456
7.10.5 Coupled Analysis Commands 457
7.10.6 Additional Commands 457
7.11 Fluid Pressure 459
7.11.1 Surface Set Commands 460
7.11.2 Specification Commands 460
7.11.3 Additional Commands 461
7.12 Specialized Boundary Conditions 462
7.12.1 Cavity Expansion 462
7.12.2 Silent Boundary 465
7.12.3 Spot Weld 466
7.12.4 Line Weld 471
7.12.5 Viscous Damping 474
7.12.5.1 Block Set Commands 474
7.12.5.2 Viscous Damping Coefficient 475
7.12.5.3 Additional Command 475
7.12.6 Volume Repulsion Old 476
7.12.6.1 Block Set 476
7.12.7 General Multi-Point Constraints 478
7.12.7.1 Master/Slave Multi-Point Constraints 478
7.12.7.2 Tied Contact 479
7.12.7.3 Tied Multi-Point Constraints 481
7.12.7.4 Resolve Multiple MPCs 481
7.12.7.5 Constraining a Subset of all DOFs 481
7.12.8 Submodel 483
7.13 References 484
8 Contact **485**

8.1 Contact Definition Block . . . 488
8.2 Defining the Contact Surfaces . . . 495
8.2.1 Contact Surface Command Line . . . 496
8.2.2 Contact Node Set Command Line . . . 497
8.2.3 Contact Surface Command Block . . . 498
8.2.3.1 Surface Subsetting . . . 498
8.2.4 Skin All Blocks Command . . . 500
8.2.5 Overlapping Contact Surfaces . . . 501
8.2.6 Element Death, Remeshing, and Surface Updates . . . 502
8.2.7 Block Skinning and Surface Definition Examples . . . 503
8.2.7.1 Simple Block Skinning . . . 503
8.2.7.2 Block Skinning With Multiple Element Blocks . . . 503
8.2.7.3 More Complex Block Skinning Cases . . . 504
8.2.7.4 Combining Block Skinning, Side Sets, and Element Death . . . 505
8.2.7.5 Equivalenced Solid and Shell Meshes . . . 506
8.2.8 Analytic Contact Surfaces . . . 507
8.2.8.1 General Analytic Surfaces . . . 507
8.2.8.2 Plane . . . 510
8.2.8.3 Cylinder . . . 511
8.2.8.4 Sphere . . . 511
8.3 Friction Models . . . 512
8.3.0.5 Frictionless Model . . . 512
8.3.0.6 Constant Friction Model . . . 512
8.3.0.7 Velocity Dependent Coulomb Friction Model . . . 513
8.3.0.8 Tied Model . . . 513
8.3.0.9 Dynamic Tied Model . . . 513
8.3.0.10 Glued Model . . . 514
8.3.0.11 Spring Weld Model . . . 514
8.3.0.12 Surface Weld Model . . . 515
8.3.0.13 Area Weld Model . . . 516
8.3.0.14 Adhesion Model . . . 517
8.3.0.15 Cohesive Zone Model . . . 517

8.3.0.16 Junction Model 518
8.3.0.17 Threaded Model 519
8.3.0.18 PV_Dependent Model 520
8.3.0.19 Hybrid Model 521
8.3.0.20 Time Variant Model 522
8.3.0.21 User Subroutine Friction Models 523
8.4 Definition of Interactions 524
8.4.1 Default Values for Interactions 525
8.4.1.1 Surface Identification 526
8.4.1.2 Self-Contact and General Contact 526
8.4.1.3 Friction Model 527
8.4.1.4 Automatic Kinematic Partition 527
8.4.1.5 Interaction Behavior 528
8.4.1.6 Constraint Formulation 529
8.4.2 Values for Specific Interactions 530
8.4.2.1 Surface Identification 531
8.4.2.2 Kinematic Partition 533
8.4.2.3 Tolerances 534
8.4.2.4 Friction Model 535
8.4.2.5 Interface Material 535
8.4.2.6 Automatic Kinematic Partition 535
8.4.2.7 Interaction Behavior 536
8.4.2.8 Constraint Formulation 536
8.4.2.9 Pushback Factor 536
8.4.2.10 Tension Release 536
8.4.2.11 Tension Release Function 537
8.4.2.12 Friction Coefficient 537
8.4.2.13 Friction Coefficient Function 537
8.4.3 Interaction Behavior for Particle Element Blocks 538
8.5 Contact Algorithm Options 539
8.5.1 Contact Library 539
8.5.1.1 How Dash and ACME differ 539

8.5.1.2 Current Dash Usage Guidelines . . . 540
8.5.2 Dash Specific Options . . . 541
8.5.3 Enforcement . . . 542
8.5.4 Remove Initial Overlap . . . 543
8.5.5 Angle for Multiple Interactions . . . 545
8.5.6 Surface Normal Smoothing . . . 547
8.5.7 Eroded Face Treatment . . . 549
8.5.8 Lofted Surface Options . . . 550
8.5.8.1 Examples of lofted geometry . . . 552
8.5.9 Search Options . . . 554
8.5.9.1 Search Tolerances . . . 555
8.5.10 User Search Box . . . 557
8.5.10.1 Search Box Location . . . 557
8.5.10.2 Search Box Size . . . 558
8.5.11 Enforcement Options . . . 559
8.5.12 Legacy Contact . . . 561
8.6 Active Periods . . . 562
8.7 Contact-Specific Outputs . . . 563
8.7.1 Contact Debugging . . . 564
8.8 Examples . . . 565
8.8.0.1 Example 1 . . . 565
8.8.0.2 Example 2 . . . 567
8.9 Common Contact Algorithmic Issues . . . 570
8.10 References . . . 571

9 Output **572**
9.1 Syntax for Requesting Variables . . . 573
9.1.1 Example 1 . . . 573
9.1.2 Example 2 . . . 574
9.1.3 Other command blocks . . . 575
9.1.4 Rigid Body Variables . . . 575
9.2 Results Output . . . 576
9.2.1 Exodus Results Output File . . . 577

9.2.1.1 Output Nodal Variables 580
9.2.1.2 Output Node Set Variables 581
9.2.1.3 Output Face Variables 582
9.2.1.4 Output Element Variables 584
9.2.1.5 Subsetting of Output Mesh 587
9.2.1.6 Output Mesh Selection 587
9.2.1.7 Component Separator Character 587
9.2.1.8 Output Global Variables 588
9.2.1.9 Set Begin Time for Results Output 589
9.2.1.10 Adjust Interval for Time Steps 589
9.2.1.11 Output Interval Specified by Time Increment 589
9.2.1.12 Additional Times for Output 589
9.2.1.13 Output Interval Specified by Step Increment 590
9.2.1.14 Additional Steps for Output 590
9.2.1.15 Set End Time for Results Output 590
9.2.1.16 Synchronize Output 590
9.2.1.17 Use Output Scheduler 591
9.2.1.18 Write Results If System Error Encountered 591
9.2.2 User-Defined Output 592
9.2.2.1 Mesh-Entity Set Commands 594
9.2.2.2 Compute Result Commands 595
9.2.2.3 User Subroutine Commands 596
9.2.2.4 Copy Command 597
9.2.2.5 Variable Transformation Command 598
9.2.2.6 Data Filtering Commands 598
9.2.2.7 Compute at Every Step Command 599
9.2.2.8 Additional Command 600
9.3 History Output 601
9.3.1 Output Variables 603
9.3.1.1 Global Output Variables 603
9.3.1.2 Mesh Entity Output Variables 604
9.3.1.3 Nearest Point Output Variables 604

9.3.2 Outputting History Data on a Node Set 605
9.3.3 Set Begin Time for History Output 606
9.3.4 Adjust Interval for Time Steps 606
9.3.5 Output Interval Specified by Time Increment 606
9.3.6 Additional Times for Output 607
9.3.7 Output Interval Specified by Step Increment 607
9.3.8 Additional Steps for Output 607
9.3.9 Set End Time for History Output 607
9.3.10 Synchronize Output 607
9.3.11 Use Output Scheduler 608
9.3.12 Write History If System Error Encountered 608
9.4 Heartbeat Output 610
9.4.1 Output Variables 612
9.4.1.1 Global Output Variables 612
9.4.1.2 Mesh Entity Output Variables 613
9.4.1.3 Nearest Point Output Variables 613
9.4.2 Outputting Heartbeat Data on a Node Set 615
9.4.3 Set Begin Time for Heartbeat Output 615
9.4.4 Adjust Interval for Time Steps 615
9.4.5 Output Interval Specified by Time Increment 616
9.4.6 Additional Times for Output 616
9.4.7 Output Interval Specified by Step Increment 616
9.4.8 Additional Steps for Output 616
9.4.9 Set End Time for Heartbeat Output 616
9.4.10 Synchronize Output 617
9.4.11 Use Output Scheduler 617
9.4.12 Write Heartbeat On Signal 617
9.4.13 Heartbeat Output Formatting Commands 618
9.4.13.1 CTH SpyHis output format 619
9.4.13.2 Specify floating point precision 619
9.4.13.3 Specify Labeling of Heartbeat Data 619
9.4.13.4 Specify Existence of Legend for Heartbeat Data 620

9.4.13.5 Specify format of timestamp . . . 620
9.4.14 Monitor Output Events . . . 621
9.5 Restart Data . . . 622
9.5.1 Restart Options . . . 623
9.5.1.1 Automatic Read and Write of Restart Files . . . 624
9.5.1.2 User-Controlled Read and Write of Restart Files . . . 627
9.5.1.3 Overwriting Restart Files . . . 630
9.5.1.4 Recovering from a Corrupted Restart . . . 631
9.5.2 Overwrite Command in Restart . . . 632
9.5.3 Set Begin Time for Restart Writes . . . 632
9.5.4 Adjust Interval for Time Steps . . . 632
9.5.5 Restart Interval Specified by Time Increment . . . 632
9.5.6 Additional Times for Restart . . . 633
9.5.7 Restart Interval Specified by Step Increment . . . 633
9.5.8 Additional Steps for Restart . . . 633
9.5.9 Set End Time for Restart Writes . . . 633
9.5.10 Overlay Count . . . 633
9.5.11 Cycle Count . . . 634
9.5.12 Synchronize Output . . . 635
9.5.13 Use Output Scheduler . . . 635
9.5.14 Write Restart If System Error Encountered . . . 636
9.6 Output Scheduler . . . 637
9.6.1 Output Scheduler Command Block . . . 637
9.6.1.1 Set Begin Time for Output Scheduler . . . 638
9.6.1.2 Adjust Interval for Time Steps . . . 638
9.6.1.3 Output Interval Specified by Time Increment . . . 638
9.6.1.4 Additional Times for Output . . . 638
9.6.1.5 Output Interval Specified by Step Increment . . . 638
9.6.1.6 Additional Steps for Output . . . 639
9.6.1.7 Set End Time for Output Scheduler . . . 639
9.6.2 Example of Using the Output Scheduler . . . 639
9.7 Variable Interpolation . . . 641

9.8 Global Output Options . . . 644
9.9 Variables . . . 645
9.9.1 Global, Nodal, Face, and Element Variables . . . 645
9.9.2 Variables for Material Models . . . 660
9.9.2.1 State Variable Output for LAME Solid Material Models . . . 660
9.9.2.2 State Variable Tables for Solid Material Models . . . 660
9.9.2.3 Variables for Shell/Membrane Material Models . . . 673
9.9.3 Variables for Surface Models . . . 675
9.9.3.1 State Variable Tables for Surface Models . . . 675
9.10 References . . . 677

10 Special Modeling Techniques 678
10.1 Representative Volume Elements . . . 678
10.1.1 RVE Processing . . . 679
10.1.2 Mesh Requirements . . . 679
10.1.3 Input Commands . . . 680
10.1.3.1 RVE Material Model . . . 681
10.1.3.2 Embedded Coordinate System . . . 681
10.1.3.3 RVE Region . . . 682
10.1.3.4 Definition of RVEs . . . 682
10.1.3.5 Multi-Point Constraints . . . 683
10.1.3.6 RVE Boundary Conditions . . . 684
10.2 J-Integrals . . . 685
10.2.1 Technique for Computing J . . . 685
10.2.2 Input Commands . . . 686
10.2.3 Output . . . 688
10.3 Peridynamics . . . 689
10.3.1 Overview . . . 689
10.3.2 Linear Peridynamic Solid Material Model . . . 690
10.3.3 Interface to Classical Material Models . . . 690
10.3.4 Modeling Fracture . . . 691
10.3.5 Peridynamics and Contact . . . 692
10.3.6 Usage Guidelines . . . 692

10.4 References 695

11 User Subroutines **696**

11.1 User Subroutines: Programming 700
11.1.1 Subroutine Interface 701
11.1.2 Query Functions 701
11.1.2.1 Parameter Query 703
11.1.2.2 Function Data Query 707
11.1.2.3 Time Query 707
11.1.2.4 Field Variables 707
11.1.2.5 Global Variables 716
11.1.2.6 Topology Extraction 720
11.1.3 Miscellaneous Query Functions 726
11.2 User Subroutines: Command File 728
11.2.1 Subroutine Identification 728
11.2.2 User Subroutine Command Lines 728
11.2.2.1 Type 728
11.2.2.2 Debugging 729
11.2.2.3 Parameters 729
11.2.3 Time Step Initialization 731
11.2.3.1 Mesh-Entity Set Commands 731
11.2.3.2 User Subroutine Commands 732
11.2.3.3 Additional Command 733
11.2.4 User Variables 734
11.3 User Subroutines: Compilation and Execution 737
11.4 User Subroutines: Examples 738
11.4.1 Pressure as a Function of Space and Time 738
11.4.2 Error Between a Computed and an Analytic Solution 741
11.4.3 Transform Output Stresses to a Cylindrical Coordinate System 745
11.5 User Subroutines: Library 751
11.5.1 aupst_cyl_transform 751
11.5.2 aupst_rec_transform 752
11.5.3 copy_data 753

11.5.4 trace 754

12 Transfers **756**

12.1 SIERRA Transfers 757

12.2 Inter-procedural Transfers 757

12.2.1 Copying Data with Inter-procedural Transfers 758

12.2.2 Interpolating Data with Interpolation Transfers 758

A Explicit Dynamic Example Problem **760**

B Implicit Quasistatic Example Problem **769**

C Command Summary **777**

D Consistent Units **854**

E Constraint Enforcement Hierarchy **856**

Index **857**

List of Figures

1.1 Input/output files . . . 54

2.1 Piecewise linear and piecewise constant functions . . . 64
2.2 Adjacent shell elements with nonaligned local coordinate systems . . . 70
2.3 Rectangular coordinate system . . . 71
2.4 Z-Rectangular coordinate system. . . . 71
2.5 Cylindrical coordinate system. . . . 72
2.6 Spherical coordinate system. . . . 72
2.7 Rotation about 1 . . . 73
2.8 Examples of elements with varying nodal Jacobians . . . 82

4.1 Reaching convergence with controls and the multilevel solver. . . . 126
4.2 Contact configuration at the beginning of the contact update. . . . 154
4.3 Contact gap removal (after contact search). . . . 154
4.4 Contact slip calculations. . . . 155
4.5 Control stiffness softening behavior in the first model problem. . . . 164
4.6 Control stiffness softening behavior in the second model problem. . . . 164
4.7 Control stiffness stiffening behavior in the first model problem. . . . 165
4.8 Control stiffness stiffening behavior in the second model problem. . . . 165
4.9 Control stiffness softening behavior convergence . . . 167

6.1 Association between command lines and command block. . . . 293
6.2 Location of geometric plane of shell for various lofting factors. . . . 306
6.3 Local rst coordinate system for a shell element. . . . 307
6.4 Rotation of 30 degrees about the 1-axis (X'-axis). . . . 308
6.5 Integration points for rod and tube . . . 315

6.6 Integration points for bar and box. . . . 316
6.7 Integration points for I-section. . . . 316
6.8 Schematic for torsional spring. . . . 338
6.9 Positive direction of rotation for torsional spring. . . . 339
6.10 Examples of how an element is converted and when it is not converted. . . . 356

7.1 Force-displacement curve for spot weld normal force. . . . 467
7.2 Force-displacement curve for spot weld tangential force. . . . 467
7.3 Sign convention for spot weld normal displacements. . . . 468
7.4 Sign convention for spot weld normal displacements with ignore initial offsets on. . . . 470

8.1 Two blocks at time step n before contact . . . 485
8.2 Two blocks at time step $n + 1$, after penetration . . . 486
8.3 Simple block skinning example . . . 503
8.4 Block skinning with multiple blocks example . . . 504
8.5 Block skinning advanced example . . . 504
8.6 Contact surface definition examples . . . 505
8.7 Hex and shell contact surface definitions . . . 507
8.8 Illustration of kinematic partition values . . . 533
8.9 Illustrations of multiple interactions at a node. . . . 545
8.10 Example lofted geometries produced by shell lofting . . . 552
8.11 Lofted shell geometry . . . 553
8.12 Lofted beam geometry . . . 553
8.13 Lofted particle geometry . . . 553
8.14 Illustration of normal and tangential tolerances . . . 556
8.15 Problem with two blocks coming into contact . . . 565
8.16 Problem with three blocks coming into contact . . . 567

10.1 Example of meshes for RVE analysis . . . 680
10.2 Example weight functions for a J-integral integration domain . . . 688

11.1 Overview of components required to implement functionality. . . . 699

A.1 Mesh for example problem. . . . 760
A.2 Mesh with blue and green surfaces removed. . . . 761

B.1 Eraser schematic 769
B.2 Complete eraser mesh 770

List of Tables

4.1 Log File Convergence Markings . . . 135

9.1 Global Variables For All Analyses . . . 647
9.2 Global Variables for Rigid Bodies . . . 648
9.3 Global Variables for J-Integral . . . 648
9.4 Nodal Variables for All Analyses . . . 649
9.5 Nodal Variables for Implicit Analyses . . . 649
9.6 Nodal Variables for Shells and Beams . . . 649
9.7 Nodal Variables for Spot Welds . . . 650
9.8 Nodal Variables for Contact . . . 651
9.9 Nodal Variables for J-Integral . . . 653
9.10 Element Variables for All Elements . . . 654
9.11 Element Variables for Solid Elements . . . 655
9.12 Element Variables for Membranes . . . 656
9.13 Element Variables for Shells . . . 656
9.14 Element Variables for Trusses . . . 657
9.15 Element Variables for Cohesive Elements . . . 657
9.16 Element Variables for Beams . . . 658
9.17 Element Variables for Springs . . . 658
9.18 Element Variables for J-Integral . . . 659
9.19 State Variables for ELASTIC Model . . . 660
9.20 State Variables for ELASTIC FRACTURE Model . . . 661
9.21 State Variables for ELASTIC PLASTIC Model . . . 661
9.22 State Variables for EP POWER HARD Model . . . 661
9.23 State Variables for DUCTILE FRACTURE Model . . . 662

9.24 State Variables for MULTILINEAR EP Model . . . 663
9.25 State Variables for ML EP FAIL Model . . . 664
9.26 State Variables for FOAM PLASTICITY Model . . . 665
9.27 State Variables for WIRE MESH Model . . . 665
9.28 State Variables for HONEYCOMB Model . . . 665
9.29 State Variables for HYPERFOAM Model . . . 666
9.30 State Variables for JOHNSON COOK Model . . . 666
9.31 State Variables for LOW DENSITY FOAM Model . . . 666
9.32 State Variables for MOONEY RIVLIN Model . . . 667
9.33 State Variables for NEO HOOKEAN Model . . . 667
9.34 State Variables for ORTHOTROPIC CRUSH Model . . . 667
9.35 State Variables for ORTHOTROPIC RATE Model . . . 667
9.36 State Variables for PIEZO Model . . . 667
9.37 State Variables for POWER LAW CREEP Model . . . 668
9.38 State Variables for SHAPE MEMORY Model . . . 668
9.39 State Variables for SOIL FOAM Model . . . 668
9.40 State Variables for SWANSON Model . . . 669
9.41 State Variables for VISCOELASTIC SWANSON Model . . . 670
9.42 State Variables for THERMO EP POWER Model . . . 670
9.43 State Variables for THERMO EP POWER WELD Model . . . 671
9.44 State Variables for UNIVERSAL POLYMER Model . . . 671
9.45 State Variables for VISCOPLASTIC Model . . . 672
9.46 State Variables for ELASTIC PLASTIC Model for Shells . . . 673
9.47 State Variables for EP POWER HARD Model for Shells . . . 673
9.48 State Variables for MULTILINEAR EP Model for Shells . . . 673
9.49 State Variables for ML EP FAIL Model for Shells . . . 674
9.50 State Variables for TRACTION DECAY Surface Model . . . 675
9.51 State Variables for TVERGAARD HUTCHINSON Surface Model . . . 675
9.52 State Variables for THOULESS PARMIGIANI Surface Model . . . 676

11.1 Subroutine Input Parameters . . . 701
11.2 Subroutine Output Parameters . . . 702
11.3 aupst_get_real_param Arguments . . . 704

11.4 aupst_get_integer_param Arguments 705
11.5 aupst_get_string_param Arguments 706
11.6 aupst_evaluate_function Arguments 707
11.7 aupst_get_time Argument 707
11.8 aupst_check_node_var Arguments 709
11.9 aupst_check_elem_var Arguments 710
11.10 aupst_get_node_var Arguments 711
11.11 aupst_get_elem_var Arguments 712
11.12 aupst_get_elem_var_offset Arguments 713
11.13 aupst_put_node_var Arguments 714
11.14 aupst_put_elem_var Arguments 715
11.15 aupst_put_elem_var_offset Arguments 716
11.16 aupst_check_global_var Arguments 718
11.17 aupst_get_global_var Arguments 718
11.18 aupst_put_global_var Arguments 719
11.19 aupst_local_put_global_var Arguments 720
11.20 Topologies Used by Sierra/SM 721
11.21 aupst_get_elem_topology Arguments 722
11.22 aupst_get_elem_nodes Arguments 723
11.23 aupst_get_face_topology Arguments 724
11.24 aupst_get_face_nodes Arguments 725
11.25 aupst_get_elem_centroid Arguments 726
11.26 aupst_get_point Arguments 727
11.27 aupst_get_proc_num Arguments 727

D.1 Consistent Unit Sets 855

E.1 Constraint Enforcement Order 856

Chapter 1

Introduction

This document is a user's guide for Sierra/SolidMechanics (Sierra/SM). Sierra/SM is a three-dimensional solid mechanics code with a versatile element library, nonlinear material models, large deformation capabilities, and contact. It is built on the SIERRA Framework [1, 2]. SIERRA provides a data management framework in a parallel computing environment that allows the addition of capabilities in a modular fashion. Contact capabilities are parallel and scalable.

The *Sierra/SolidMechanics 4.22 User's Guide* provides information about the functionality in Sierra/SM and the command structure required to access this functionality in a user input file. This document is divided into chapters based primarily on functionality. For example, the command structure related to the use of various element types is grouped in one chapter; descriptions of material models are grouped in another chapter.

Sierra/SM provides both explicit transient dynamics and implicit quasistatics and dynamics capabilities. Both the explicit and implicit modules are are highly scalable in a parallel computing environment. In the past, the explicit and implicit capabilities were provided by two separate codes, known as Presto and Adagio, respectively. These capabilities have been consolidated into a single code. The executables are still named Presto and Adagio, but the two executables are now identical, and provide the full suite of solid mechanics capabilities.

Important references for both the implicit and explicit capabilities are given in the references section at the end of this chapter. Adagio was preceded by the codes JAC and JAS3D; JAC is described in Reference 4; JAS3D is described in Reference 5. Presto was preceded by the code Pronto3D. Pronto3D is described in References 6 and 7. Some of the fundamental nonlinear technology used by Sierra/SM are described in References 8, 9, and 10. Sierra/SM uses the Exodus II database and the XDMF database; Exodus II is more commonly used than XDMF. (Other options may be added in the future.) The Exodus II database format is described in Reference 11, and the XDMF database format is described in Reference 12. Important information about contact is provided in the reference document for ACME [13]. ACME is a third-party library for contact.

One of the key concepts for the command structure in the input file is a concept referred to as *scope*. A detailed explanation of scope is provided in Section 1.2. Most of the command lines in Chapter 2 are related to a certain scope rather than to some particular functionality.

1.1 Document Overview

This document describes how to create an input file for Sierra/SM. Highlights of the document contents are as follows:

- Chapter 1 presents the overall structure of the input file, including conventions for the command descriptions, style guidelines for file preparation, and naming conventions for input files that reference the Exodus II database [11]. The chapter also gives an example of the general structure of an input file that employs the concept of scope.

- Chapter 2 explains some of the commands that are general to various applications based on the SIERRA Framework. These commands let you define scopes, functions, and coordinate systems, and they let you set up some of the main time control parameters (begin time, end time, time blocks) for your analysis. (Time control and time step control are discussed in more detail in Chapters 2.7 and 3.5.) Other capabilities documented in this chapter are available for calculating element distortion and for activating and deactivating functionality at different times throughout an analysis.

- Chapter 2.7 describes how to set the start time, end time, and time blocks for an explicit dynamic analysis. This chapter also discusses various options for controlling the critical time step for transient dynamics.

- Chapter 3.5 discusses the multilevel, nonlinear iterative solver used for implicit calculations. This chapter also describes how to set start time, end time, and time blocks for an implicit analysis.

- Chapter 5 describes material models that can be used in conjunction with the elements. Most of the material models can be used for both explicit and implicit analyses. Even though a material model can be used by both cases, it may be that the use of the material model is better suited for one type of analysis. For example, a material model set up to characterize behavior over a long time would be better suited for use in implicit analyses than in explicit analyses. In such cases, this will be noted. Chapter 5 also discusses the application of temperature to a mesh and the computation of thermal strains (isotropic and anisotropic).

- Chapter 6 lists the available elements and describes the commands used to access the various options for the elements. Most elements can be used for both explicit and implicit analyses. If that is not the case, a note is made to that effect.

  Chapter 6 also includes descriptions of the commands for mass property calculations, element death, and mesh rebalancing. Two "element-like" capabilities, rigid bodies and torsional springs, are also discussed in Chapter 6.

- Chapter 7 documents how to use kinematic boundary conditions, force boundary conditions, initial conditions, and specialized boundary conditions.

- Chapter 8 discusses how to define interactions of contact surfaces.

- Chapter 9 details the various options for obtaining output.

- Chapter 10 documents special modeling techniques.
- Chapter 11 provides an overview of the user subroutine functionality.
- Chapter 12 documents how to perform transfers between procedures.
- Appendix A provides a sample input file for an explicit dynamic analysis of 16 lead spheres being crushed together inside a steel box. Appendix B provides a sample input file for an implicit, quasistatic analysis of an eraser being pulled across a surface. These problems both emphasize large deformation and contact.
- Appendix C lists all the permissible Sierra/SM input lines in their proper scope.
- The index allows you to find information about command blocks and command lines. In general, single-level entries identify the page where the command syntax appears, with discussion following soon thereafter—on the same page or on a subsequent page. Page ranges are not provided in this index. Some entries consist of two or more levels. Such entries are typically based on context, including such information as the command blocks in which a command line appears, the location of the discussion related to a particular command line, and tips on usage. The electronic version of this document contains hyperlinked entries from the page numbers listed in the index to the text in the body of the document.

1.2 Overall Input Structure

Sierra/SM is one of many mechanics codes built on the SIERRA Framework. The SIERRA Framework provides the capability to perform multiphysics analyses by coupling together SIERRA codes appropriate for the mechanics of interest. Input files may be set up for analyses using only Sierra/SM, or they may be set up to couple Sierra/SM and one or more other SIERRA analysis codes. For example, you might run an implicit Sierra/SM analysis to compute a stress state, and then use the results of this analysis as initial conditions for an explicit analysis. For a Sierra multiphysics analysis, the commands for all physics modules used in the analysis will all be in the same input file. Therefore, the input for Sierra/SM reflects the fact that it could be part of a multiphysics analysis. (Note that not all codes built on the SIERRA Framework can be coupled. Consult with the authors of this document to learn about the codes that can be coupled with Sierra/SM.)

To create files defining multiphysics analyses, the input files use a concept called "scope." Scope is used to group similar commands; a scope can be nested inside another scope. The broadest scope in the input file is the SIERRA scope. The SIERRA scope contains information that can be shared among different physics. Examples of physics information that can be shared are definitions of functions and materials. For example, in a coupled implicit/explicit Sierra/SM analysis, both the implicit and explicit components could reference functions to define such things as time histories for boundary conditions or stress-strain curves. Material model definitions could also be shared in a similar manner.

Within the SIERRA scope are two other important scopes: the procedure scope and the region scope. The region is nested inside the procedure, and the procedure is nested inside the SIERRA scope. The procedure scope controls the overall analysis from the start time to the end time; the region scope controls a single time step. For a multiphysics analysis, the SIERRA scope could contain several different procedures and several different regions.

Inside the procedure scope (but outside of the region scope) are commands that set the start time and the end time for the analysis.

Inside the region scope for Sierra/SM are such things as definitions for boundary conditions and contact. In a multiphysics analysis, there would be more than one region. In a coupled implicit/explicit solid mechanics example, there would be both a Presto region and an Adagio region, each within its respective procedures. The definitions for boundary conditions and contact and the mesh specification for explicit dynamics would appear in the Presto region; the definitions for boundary conditions and contact and the mesh specification for the implicit analysis would appear in the Adagio region.

The input for Sierra/SM consists of command blocks and command lines. The command blocks define a scope. These command blocks group command lines or other command blocks that share a similar functionality. A command block will begin with an input line that has the word "begin"; the command block will end with an input line that has the word "end". The SIERRA scope, for example, is defined by a command block that begins with an input line of the following form:

```
BEGIN SIERRA my_problem
```

The two character strings `BEGIN` and `SIERRA` are the key words for this command block. An input line defining a command block or a command line will have one or more key words. The string

`my_problem` is a user-specified name for this SIERRA scope. The SIERRA scope is terminated by an input line of the following form:

```
END SIERRA my_problem
```

In the above input line, `END` and `SIERRA` are the key words to end this command block. The SIERRA scope can also be terminated simply by using the following key word:

```
END
```

The above abbreviated command line will be discussed in more detail in later chapters. There are similar input lines used to define the procedure and region scopes. Boundary conditions are another example where a scope is defined. A particular instance of a boundary condition for a prescribed displacement boundary condition is defined with a command block. The command block for the boundary condition begins with an input line of the form:

```
BEGIN PRESCRIBED DISPLACEMENT
```

and ends with an input line of either of the following forms:

```
END PRESCRIBED DISPLACEMENT
```

```
END
```

Command lines appear within the command blocks. The command lines typically have the form `keyword = value`, where `value` can be a real, an integer, or a string. In the previous example of the prescribed displacement boundary condition, there would be command lines inside the command block that are used to set various values. For example, the boundary condition might apply to all nodes in node set 10, in which case there would be a command line of the following form:

```
NODE SET = nodelist_10
```

If the prescribed displacement were to be applied along a given component direction, there would be a command line of this form:

```
COMPONENT = X
```

The form above would specify that the prescribed displacement would be in the *x*-direction. Finally, if the displacement magnitude is described by a time history function with the name `cosine_curve`, there would be a command line of this form:

```
FUNCTION = cosine_curve
```

The command block for the boundary condition with the appropriate command lines would appear as follows:

```
BEGIN PRESCRIBED DISPLACEMENT
  NODE SET = nodelist_10
  COMPONENT = X
  FUNCTION = cosine_curve
END PRESCRIBED DISPLACEMENT
```

It is possible to have a command line with the same key words appearing in different scopes. For example, we might have a command line identified by the word `TYPE` in two or more different

scopes. The command line would perform different functions based on the scope in which it appeared, and the associated value could be different in the two locations.

The input lines are read by a parser that searches for recognizable key words. If the key words in an input line are not in the list of key words used by Sierra/SM to describe command blocks and command lines, the parser will generate an error. A set of key words defining a command line or command block for Sierra/SM that is not in the correct scope will also cause a parser error. For example, the key words `STEP INTERVAL` define a valid command line in the scope of the `TIME CONTROL` command block. However, if this command line was to appear in the scope of one of the boundary conditions, it would not be in the proper scope, and the parser would generate an error. Once the parser has an input line with any recognizable key words in the proper scope, a method can be called that will handle the input line.

There is an initial parsing phase that checks only the parser syntax. If the parser encounters a command line it cannot parse within a certain scope, the parser will indicate it cannot recognize the command line and will list the various command lines that can appear within that scope. The initial parsing phase will catch errors such as the one described in the previous paragraph (a command line in the wrong scope). It will also catch misspelled key words. The initial parsing does not catch some other types of errors, however. If you have specified a value on a command line that is out of a specified range for that command line, the initial parsing will not catch this error. If you have some combination of command lines within a command block that is not allowed, the initial parsing will not catch this error. These other errors are caught after the initial parsing phase and are handled one error at a time.

1.3 Conventions for Command Descriptions

The conventions below are used to describe the input commands for Sierra/SM. A number of the individual command lines discussed in the text appear on several text lines. In the text of this document, the continuation symbols that are used to continue lines in an actual input file (`\#` and `\$`, Section 1.4.2) are not used for those instances where the description of the command line appears on several text lines. The description of command lines will clearly indicate all the key words, delimiters, and values that constitute a complete command line. As an example, the `DEFINE POINT` command line (Section 2.1.6) is presented in the text as follows:

```
DEFINE POINT <string>point_name WITH COORDINATES
     <real>value_1 <real>value_2 <real>value_3
```

If the `DEFINE POINT` command line were used as a command line in an input file and spread over two input lines, it would appear, with actual values, as follows:

```
DEFINE POINT center WITH COORDINATES \#
  10.0 144.0 296.0
```

In the above example, the `\#` symbol implies the first line is continued onto the second line.

1.3.1 Key Words

The key word or key words for a command are shown in uppercase letters. For actual input, you can use all uppercase letters for the key words, all lowercase letters for the key words, or some combination of uppercase and lowercase letters for the key words.

1.3.2 User-Specified Input

The input that you supply is typically shown in lowercase letters. (Occasionally, uppercase letters may be used for user input for purposes of clarity or in examples.) The user-supplied input may be a real number, an integer, a string, or a string list. For the command descriptions, a type appears before the user input. The type (real, integer, string, string list) description is enclosed by angle brackets, `<>`, and precedes the user-supplied input. For example:

```
<real>value
```

indicates that the quantity `value` is a real number. For the description of an input command, you would see the following:

```
FUNCTION = <string>function_name
```

Your input would be

```
FUNCTION = my_name
```

if you have specified a function name called `my_name`.

Valid user input consists of the following:

`<integer>`	Integer data is a single integer number.
`<real>`	Real data is a single real number. It may be formatted with the usual conventions, such as `1234.56` or `1.23456e+03`.
`<string>`	String data is a single string.
`<string list>`	A string list consists of multiple strings separated by white space, a comma, a tab, or white space combined with a comma or a tab.

1.3.3 Optional Input

Anything in an input line that is enclosed by square brackets, [ ], represents optional input within the line. Note, however, that this convention is not used to identify optional input lines. Any command line that is optional (in its entirety) will be described as such within the text.

1.3.4 Default Values

A value enclosed by parentheses, (), appearing after the user input denotes the default value. For example:

```
SCALE FACTOR = <real>scale_factor(1.0)
```

implies the default value for `scale_factor` is 1.0. Any value you specify will overwrite the default.

For your actual input file, you may simply omit a command line if you want to use the default value associated with the command line. For example, there is a `TIME STEP SCALE FACTOR` command line used to set one of the time control parameters; the parameter for this command line has a default value of 1.0. If you want to use the default value of 1.0 for this parameter, you do not have to include the `TIME STEP SCALE FACTOR` command line in the `TIME CONTROL` command block.

1.3.5 Multiple Options for Values

Quantities separated by the `|` symbol indicate that one and only one of the possible choices must be selected. For example:

```
EXPANSION RADIUS = <string>SPHERICAL|CYLINDRICAL
```

implies that expansion radius must be defined as `SPHERICAL` or `CYLINDRICAL`. One of the values must appear. This convention also applies to some of the command options within a begin/end block. For example:

```
SURFACE = <string>surface_name|
     NODE SET = <string>nodelist_name
```

in a command block specifies that either a surface or a node set must be specified.

Quantities separated by the / symbol can appear in any combination, but any one quantity in the sequence can appear only once. For example,

```
COMPONENTS = <string>X/Y/Z
```

implies that components can equal any combination of X, Y, and Z. Any value (X or Y or Z) can appear at most once, and at least one value of X, Y, or Z must appear. Some examples of valid expressions in this case are as follows:

```
COMPONENTS = Z

COMPONENTS = Z X

COMPONENTS = Y X Z

COMPONENTS = Z Y X
```

An example of an invalid expression would be the following:

```
COMPONENTS = Y Y Z
```

1.3.6 Known Issues and Warnings

Where there are known issues with the code, these are documented in the following manner:

Known Issue: A description of the known issue with the code would be provided here.

Similarly, warnings regarding usage of code features that are not defective, but must be used with care because of their nature, are documented as follows:

Warning: A description of the warning related to the usage of a code feature would be provided here.

1.3.7 Commands Applicable to Only Explicit or Implicit Analyses

As mentioned previously, Sierra/SM has the ability to run either explicit dynamic analyses and implicit quasistatic and dynamic analyses. This User's Guide documents all features of the code for either of these analysis types. In most cases, the features available for explicit analyses are also available for implicit analysis, and vice versa. However, there are some capabilities that are only available for one type of analysis. These are denoted in this document with the following icons in

the margin.

Features and commands only available for explicit dynamics have this icon in the margin next to their documentation.

Features and commands only available for implicit dynamics and quasistatics have this icon in the margin next to their documentation.

In listings of command lines and blocks, smaller versions of these icons are used to denote commands only available for explicit or implicit analyses. These appear in the margin next to either an individual command line, or next to the opening line of a command block, as shown in the following examples:

```
#Example explicit command line:
DEATH ON INVERSION = OFF|ON(OFF)          [Explicit]

#Example implicit command line:
CAPTURE TOLERANCE = <real>cap_tol         [Implicit]

#Example explicit command block:
BEGIN LINE WELD                           [Explicit]
  ...
END [LINE WELD]

#Example implicit command block:
BEGIN SOLVER                              [Implicit]
  ...
END [SOLVER]
```

1.4 Style Guidelines

This section gives information that will affect the overall organization and appearance of your input file. It also contains recommendations that will help you construct input files that are readable and easy to proof.

1.4.1 Comments

A comment is anything between the `#` symbol or the `$` symbol and the end-of-line. If the first non-blank character in a line is a `#` or `$`, the entire line is a comment line. You can also place a `#` or `$` (preceded by a blank space) after the last character in an input line used to define a command block or command line.

1.4.2 Continuation Lines

An input line can be continued by placing a `\#` pair of characters (or `\$`) at the end of the line. The following line is then taken to be a continuation of the preceding line that was terminated by the `\#` or `\$`. Note that everything after the line-continuation pair of characters is discarded, including the end-of-line.

1.4.3 Case

Almost all the character strings in the input lines are case insensitive. For example, the `BEGIN SIERRA` key words could appear as one of the following:

```
BEGIN SIERRA
begin sierra
Begin Sierra
```

You could specify a `SIERRA` command block with:

```
BEGIN SIERRA BEAM
```

and terminate the command block with this input line:

```
END SIERRA beam
```

Case is important only for file name specifications. If you have defined a restart file with uppercase and lowercase letters and want to use this file for a restart, the file name you use to request this restart file must exactly match the original definition you chose.

1.4.4 Commas and Tabs

Commas and tabs in input lines are ignored.

1.4.5 Blank Spaces

We highly recommend that everything be separated by blank spaces. For example, a command line of the form

```
node set = nodelist_10
```

is recommended over the following forms:

```
node set= nodelist_10
```

```
node set =nodelist_10
```

Both of the above two lines are correct, but it is easier to check the first form (the equal sign surrounded by blank space) in a large input file.

The parser will accept the following line:

```
BEGIN SIERRABEAM
```

However, it is harder to check this line for the correct spelling of the key words and the intended SIERRA scope name than this line:

```
BEGIN SIERRA BEAM
```

It is possible to introduce hard-to-detect errors because of the way in which the blank spaces are handled by the command parser. Suppose you type

```
begin definition for functions my_func
```

rather than the following correct form:

```
begin definition for function my_func
```

For the incorrect form of this command line (in which `functions` is used rather than `function`), the parser will generate a string name of

```
s my_func
```

for the function name rather than the following expected name:

```
my_func
```

If you attempt to use a function named `my_func`, the parser will generate an error because the list of function names will include `s my_func` but not `my_func`.

1.4.6 General Format of the Command Lines

In general, command lines have the following form:

```
keyword = value
```

This pattern is not always followed, but it describes the vast majority of the command lines.

1.4.7 Delimiters

The delimiter used throughout this document is "=" (the equal sign). Typically, but not always, the = separates key words from input values in a command line. Consider the following command line:

```
COMPONENTS = X
```

Here, the key word `COMPONENTS` is separated from its value, a string in this case, by the =. Some command lines do allow for other delimiters. The use of these alternate delimiters is not consistent, however, throughout the various command lines. (This lack of consistency has the potential for introducing errors in this document as well as in your input.) The = provides a strong visual cue for separating key words from values. By using the = as a delimiter, it is much easier to proof your input file. It also makes it easier to do "cut and paste" operations. If you accidentally delete =, it is much easier to detect than accidentally removing part of one of the other delimiters that could be used.

1.4.8 Order of Commands

There are no requirements for ordering the commands. Both the input sequence:

```
BEGIN PRESCRIBED DISPLACEMENT
  NODE SET = nodelist_10
  COMPONENT = X
  FUNCTION = cosine_curve
END PRESCRIBED DISPLACEMENT
```

and the input sequence:

```
BEGIN PRESCRIBED DISPLACEMENT
  FUNCTION = cosine_curve
  COMPONENT = X
  NODE SET = nodelist_10
END PRESCRIBED DISPLACEMENT
```

are valid, and they produce the same result. Remember, that command lines and command blocks must appear in the proper scope.

1.4.9 Abbreviated END Specifications

It is possible to terminate a command block without including the key word or key words that identify the block. You could define a specific instance of the prescribed displacement boundary condition with:

```
BEGIN PRESCRIBED DISPLACEMENT
```

and terminate it simply with:

```
END
```

as opposed to the following specification:

```
END PRESCRIBED DISPLACEMENT
```

Both the short termination (END only) and the long termination (END followed by identification, or name, of the command block) are valid. It is recommended that the long termination be used for any command block that becomes large. The RESULTS OUTPUT command block described in later chapters can become fairly lengthy, so this is probably a good place to use the long termination. For most boundary conditions, the command block will typically consist of five lines. In such cases, the short termination can be used. Using the long termination for the larger command blocks will make it easier to proof your input files. If you use the long termination, the text following the END key word must exactly match the text following the BEGIN key word. You could not have BEGIN PRESCRIBED DISPLACEMENT paired with an END PRESCRIBED DISPL to define the beginning and ending of a command block.

1.4.10 Indentation

When constructing an input file, it is useful, but not required, to indent a scope that is nested inside another scope. Command lines within a command block should also be indented in relation to the lines defining the command block. This will make it easier to construct the input file with everything in the correct scope and with all the command blocks in the correct structure.

1.4.11 Including Files

External text files containing input commands can be included at any point in the Sierra/SM input file using the INCLUDEFILE command. This command can be used in any context in the input file. To use this command, simply use the command INCLUDEFILE followed by the name of the file to be included. For example, the command:

```
INCLUDEFILE displacement_history.i
```

would include the displacement_history.i as if the contents of that file were places in the position that it is included in the input file. The included file is contained in the standard echo of the input that is provided at the beginning of the log file.

1.5 Exodus II Database Naming Conventions

When the mesh file has an Exodus II format, there are three basic conventions that apply to user input for various command lines. First, for a mesh file with the Exodus II format, the Exodus II side set is referenced as a surface. In SIERRA, a surface consists of element faces plus all the nodes and edges associated with these faces. A surface definition can be used not only to select a group of faces but also to select a group of edges or a group of nodes that are associated with those faces. In the case of boundary conditions, a surface definition can be used not only to apply boundary conditions that typically use surface specifications (pressure) but also to apply boundary conditions for what are referred to as nodal boundary conditions (fixed displacement components). For nodal boundary conditions that use the surface specification, all the nodes associated with the faces on a specific surface will have this boundary condition applied to them. The specification for a surface identifier in the following chapters is `surface_name`. It typically has the form `surface_integerid`, where `integerid` is the integer identifier for the surface. If the side set identifier is 125, the value of `surface_name` would be `surface_125`. It is also possible to generate an alias for the side set[1] and use this for `surface_name`. If `surface_125` is aliased to `outer_skin`, then `surface_name` becomes `outer_skin` in the actual input line. It is also possible to name a surface in some mesh generation programs and that name can be used in the input file.

Second, for a mesh file with the Exodus II format, the Exodus II node set is still referenced as a node set. A node set can be used only for cases where a group of nodes needs to be defined. The specification for a node set identifier in the following chapters is `nodelist_name`. It typically has the form `nodelist_integerid`, where `integerid` is the integer identifier for the node set. If the node set number is 225, the value of `nodelist_name` would be `nodelist_225`. It is also possible to generate an alias for the node set and use this for `nodelist_name`. If `nodelist_225` is aliased to `inner_skin`, then `nodelist_name` becomes `inner_skin` in the actual input line. It is also possible to name a nodelist in some mesh generation programs and that name can be used in the input file.

Third, an element block is referenced as a block. The specification for an element block identifier in the following chapters is `block_name`. It typically has the form `block_integerid`, where `integerid` is the integer identifier for the block. If the element block number is 300, the value of `block_name` would be `block_300`. It is also possible to generate an alias for the block and use this for `block_name`. If `block_300` is aliased to `big_chunk`, then `block_name` becomes `big_chunk` in the actual input line. It is also possible to name an element block in some mesh generation programs and that name can be used in the input file.

A group of elements can also be used to select other mesh entities. In SIERRA, a block consists of elements plus all the faces, edges, and nodes associated with the elements. The block and surface concepts are similar in that both have associated derived quantities. Chapters 7 and 8 show how this concept of derived quantities is used in the input command structure.

[1] See the `ALIAS` command in Section 6.1.2

1.6 Major Scope Definitions for an Input File

The typical input file will have the structure shown in the two examples below. The major scopes—SIERRA, procedure, and region—are delineated with input lines for command blocks. Comment lines are included that indicate some of the key scopes that will appear within the major scopes. Note the indentation used to clarify scopes. The following is an example of the structure an explicit dynamic (Presto) input file:

Explicit

```
BEGIN SIERRA <string>some_name
  #
  # All command blocks and command lines in the SIERRA
  # scope appear here. The PROCEDURE PRESTO command
  # block is the beginning of the next scope.
  #
  # function definitions
  # material descriptions
  # description of mesh file
  #
  BEGIN PRESTO PROCEDURE <string>procedure_name
    #
    # time step control
    #
    BEGIN PRESTO REGION <string>region_name
      #
      # All command blocks and command lines in the
      # region scope appear here
      #
      # specification for output of result
      # specification for restart
      # boundary conditions
      # definition of contact
      #
    END [PRESTO REGION <string>region_name]
  END [PRESTO PROCEDURE <string>procedure_name]
END [SIERRA <string>some_name]
```

This is an example of an implicit (Adagio) input file:

Implicit

```
BEGIN SIERRA <string>some_name
  #
  # All command blocks and command lines in the SIERRA
  # scope appear here. The PROCEDURE ADAGIO command
  # block is the beginning of the next scope.
  #
  # function definitions
  # material descriptions
```

```
  # description of mesh file
  #
  BEGIN ADAGIO PROCEDURE <string>procedure_name
    #
    # time step control
    #
    BEGIN ADAGIO REGION <string>region_name
      #
      # All command blocks and command lines in the
      # region scope appear here
      #
      # solver commands
      # specification for output of result
      # specification for restart
      # boundary conditions
      # definition of contact
      #
    END [ADAGIO REGION <string>region_name]
  END [ADAGIO PROCEDURE <string>procedure_name]
END [SIERRA <string>some_name]
```

1.7 Input/Output Files

The primary user input to Sierra/SM is the input file introduced in this chapter. Throughout this document, we explain how to construct a valid input file. It is important to be aware that Sierra/SM also processes a number of other types of input files and produces a variety of output files. These additional files are also discussed in this document where applicable. Figure 1.1 shows a simple schematic diagram of the various input and output files.

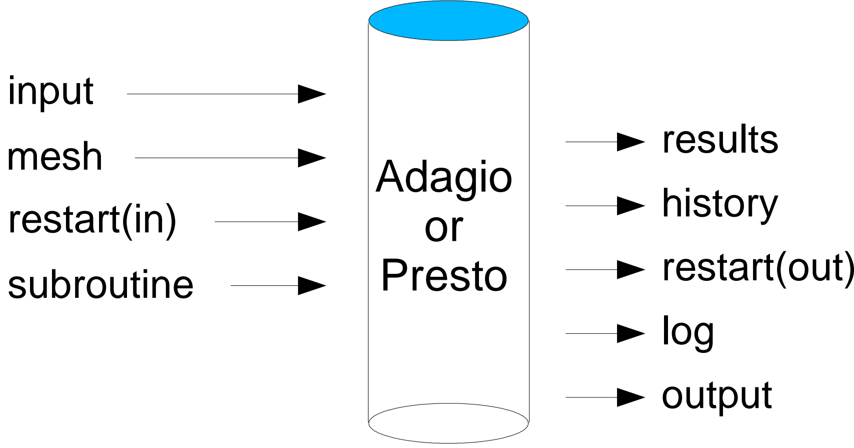


Figure 1.1: Input/output files

As shown in Figure 1.1, Sierra/SM uses the input file, mesh files, restart files, and user subroutine files. The input file, which is required, is a set of valid Sierra/SM command lines. Another required input is a mesh file, which provides a description of the finite element mesh for the object being analyzed. Restart and user subroutine files are optional inputs. The restart functionality lets you break an analysis from the start time to the termination time into a sequence of runs. The files generated by the restart functionality contain a complete state description for a problem at various analysis times, which we will refer to as restart times. You can restart Sierra/SM at any of these restart times because the complete state description is known (see Chapter 9). The user subroutine files let you build and incorporate specialized functionality into Sierra/SM (Chapter 11).

As also shown in Figure 1.1, Sierra/SM can generate a number of files. These include results files, history files, restart files, a log file, and an output file. Typically, only the log file and the output file are produced automatically. Generation of the other types of files is based on user settings in the input file for the particular kinds of output desired. Results files provide the values of global variables, element variables, and node variables at specified times (see Chapter 9). History files will also provide values of global variables, element variables, and node variables at specified times (see Chapter 9). History files are set up to provide a specific value at a specific node, for example, whereas results files provide a nodal value for large subsets of nodes or, more typically, all nodes. History files provide a much more limited set of information than results files. As noted above, restart files can be generated at various analysis times. The log file contains a variety of information such as the Sierra/SM version number, a listing of the input file, initialization information, some model information (mass, critical time steps for element blocks, etc.), and information at various

time steps. At every n^{th} step, where n is user selected, the log file gives the current analysis time; the current time step; the kinetic, internal, and external energies; the error in the energy; and computing time information. You can monitor step information in the log file to gain information about how your analysis is progressing. The output file contains error information.

1.8 Obtaining Support

Support for all SIERRA Mechanics codes, including Sierra/SM, can be obtained by contacting the SIERRA Mechanics user support hotline by email at sierra-help@sandia.gov, or by telephone at (505)845-1234.

1.9 References

1. Edwards, H. C., and J. R. Stewart. "SIERRA: A Software Environment for Developing Complex Multi-Physics Applications." In *First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, 1147–1150. Amsterdam: Elsevier, 2001.

2. Koteras, J. R., A. S. Gullerud, V. L. Porter, W. M. Scherzinger, and K. H. Brown. "PRESTO: Impact Dynamics with Scalable Contact Using the SIERRA Framework." In *First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, 294–296. Amsterdam: Elsevier, 2001.

3. Mitchell, J. A., A. S. Gullerud, W. M. Scherzinger, J. R. Koteras, and V. L. Porter. "ADAGIO: Non-Linear Quasi-Static Structural Response Using the SIERRA Framework." In *First MIT Conference on Computational Fluid and Solid Mechanics*, edited by K. J. Bathe, 361–364. Amsterdam: Elsevier, 2001.

4. Biffle, J. H. *JAC – A Two-Dimensional Finite Element Computer Program for the Non-Linear Quasi-Static Response of Solids with the Conjugate Gradient Method*, SAND81-0998. Albuquerque, NM: Sandia National Laboratories, April 1984. pdf.

5. Blanford, M. L., M. W. Heinstein, and S. W. Key. *JAS3D – A Multi-Strategy Iterative Code for Solid Mechanics Analysis Users' Instructions, Release 2.0*, Draft SAND report. Albuquerque, NM: Sandia National Laboratories, September 2001.

6. Taylor, L. M. and D. P. Flanagan. *Pronto3D: A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912. Albuquerque, NM: Sandia National Laboratories, March 1989. pdf.

7. Attaway, S. W., K. H. Brown, F. J. Mello, M. W. Heinstein, J. W. Swegle, J. A. Ratner, and R. I. Zadoks. *PRONTO3D User's Instructions: A Transient Dynamic Code for Nonlinear Structural Analysis*, SAND98-1361. Albuquerque, NM: Sandia National Laboratories, June 1998. pdf.

8. Laursen, T. A., S. W. Attaway, and R. I. Zadoks. *SEACAS Theory Manuals: Part I. Problem Formulation in Nonlinear Solid Mechanics*, SAND98-1760/1. Albuquerque, NM: Sandia National Laboratories, August 1998. pdf.

9. Laursen, T. A., S. W. Attaway, and R. I. Zadoks. *SEACAS Theory Manuals: Part II. Nonlinear Continuum Mechanics*, SAND98-1760/2. Albuquerque, NM: Sandia National Laboratories, September 1998. pdf.

10. Laursen, T. A., S. W. Attaway, and R. I. Zadoks. *SEACAS Theory Manuals: Part III. Finite Element Analysis in Nonlinear Solid Mechanics*, SAND98-1760/3. Albuquerque, NM: Sandia National Laboratories, March 1999. pdf.

11. Larry A. Schoof, Victor R. Yarberry, *EXODUS II: A Finite Element Data Model*, SAND92-2137, Sandia National Laboratories, September 1994. pdf. See also documentation available at EXODUS II Sourceforge page. link.

12. The eXtensible Data Model and Format (XDMF). link.

13. Brown, K. H., R. M. Summers, M. W. Glass, A. S. Gullerud, M. W. Heinstein, and R. E. Jones. *ACME: Algorithms for Contact in a Multiphysics Environment, API Version, 1.0*. SAND2001-3318. Albuquerque, NM: Sandia National Laboratories, October 2001. pdf.

Chapter 2

General Commands

The commands described in this section appear in the SIERRA or procedure scope or control general functionality in Sierra/SM.

2.1 SIERRA Scope

These commands are used to set up some of the fundamentals of the Sierra/SM input. The commands are physics independent, or at least can be shared between physics. The commands lie in the SIERRA scope, not in the procedure or region scope.

2.1.1 SIERRA Command Block

```
BEGIN SIERRA <string>name
  #
  # All other command blocks and command lines
  # appear within the SIERRA scope defined by
  # begin/end sierra.
  #
END [SIERRA <string>name]
```

All input commands must occur within a `SIERRA` command block. The syntax for beginning the command block is:

```
BEGIN SIERRA <string>name
```

and for terminating the command block is as follows:

```
END [SIERRA <string>name]
```

In these input lines, `name` is a name for the `SIERRA` command block. All other commands for the analysis must be within this command block structure. The name for the `SIERRA` command block is often a descriptive name that identifies the analysis. The name is not currently used anywhere else in the file and is completely arbitrary.

2.1.2 Title

```
TITLE <string list>title
```

To permit a fuller description of the analysis, the input has a `TITLE` command line for the analysis, where `title` is a text description of the analysis. The title is transferred to the results file.

2.1.3 Restart Control

The restart capability in Sierra/SM allows a user to run an analysis up to a certain time, stop the analysis at this time, and then restart the analysis from this time. Restart can be used to break a long-running analysis into several smaller runs so that the user can examine intermediate results before proceeding with the next step. Restart can also be used in case of abnormal termination. If a restart file has been written at various intervals throughout the analysis up to the point where the abnormal termination has occurred, you can pick a restart time before the abnormal termination and restart the problem from there. Thus, users do not have to go back to the beginning of the analysis, but can continue the analysis at some time well into the analysis. With the restart capability, you will generate a sequence of restart runs. Each run can have its own set of restart, results, and history files.

When using the restart capability, you can reset a number of the parameters in the input file. However, not all parameters can be reset. Users should exercise care in resetting parameters in the input file for a restart. You will want to change parameters if you have encountered an abnormal termination. You may want to change certain parameters, hourglass control for example, to see whether you can prevent the abnormal termination and continue the analysis past the abnormal termination time you had previously encountered.

The use of the restart capability involves commands in **both the SIERRA scope and the region scope**. One of two restart command lines, `RESTART` or `RESTART TIME`, appears in the SIERRA scope. A command block in the region scope, the `RESTART DATA` command block, specifies restart file names and the frequency at which the restart files will be written. The `RESTART DATA` command block is described in Section 9.5. This section gives a brief discussion of the command lines that appear in the SIERRA scope. For a full discussion of all the command lines used for restart, consult Chapter 9. The use of some of the command lines in the `RESTART DATA` command block depends on the command line, either `RESTART` or `RESTART TIME`, you select in the SIERRA scope.

If you specify a time from a specific restart file for the restart, you will use the `RESTART TIME` command line described in Section 2.1.3.1. If you select the automatic restart option, you will use the `RESTART` command line described in Section 2.1.3.2. The command lines for both of these methods are in the **SIERRA scope**. All other commands for restart are in the **region scope** in the `RESTART DATA` command block.

For restarts specified with a restart time from a specific restart file, you will have to be concerned about overwriting information in existing files. The issue of overwriting information is discussed in Chapter 9. In general, you will want to have a restart file (or files in the case of parallel runs)

for each run in a sequence of runs you create with the restart option. You will want to preserve all restart files you have written prior to any given run in a sequence of restart runs. The easiest way to preserve prior restart information is with the use of the RESTART command line. How you preserve previous restart information is discussed in detail in Chapter 9.

The amount of data written at a restart time is quite large. The restart data written at a given time is a complete description of the state for the problem at that time. The restart data includes not only information such as displacement, velocity, and acceleration, but also information such as element stresses and all the state variables for the material model associated with each element.

2.1.3.1 Restart Time

```
RESTART TIME = <real>restart_time
```

The RESTART TIME command line is used to specify a time from a specific restart file for the restart run. This restart option will pick the restart time on the restart file that is closest to the user-specified time on the RESTART TIME command line. If the user specifies a restart time greater than the last time written to a restart file, then the last time written to the restart file is picked as the restart time. Use of this command line can result in previous restart information being overwritten. To prevent the overwriting of existing restart files, you can specify both an input restart file and an output restart file (and rename the results and history files) for the various restarts. The use of the RESTART TIME command line requires the user to be more active in the management of the file names to prevent the overwriting of restart, results, and history files. The automatic restart feature (e.g., the RESTART command line in Section 2.1.3.2) prevents the overwriting of restart, results, and history files. See Section 9.5 for a full discussion of implementing the restart capability.

2.1.3.2 Automatic Restart

```
RESTART = AUTOMATIC
```

The RESTART command line automatically selects for restart the last restart time written to the last restart file. The automatic restart feature lets the user restart runs with minimal changes to the input file. The only quantity that must be changed to move from one restart to another is the termination time. The RESTART command line manages the restart files so as not to write over any previous restart files. It also manages the results and history files so as not to write over any previous results or history files. See Section 9.5 for a full discussion of implementing the restart capability.

2.1.4 User Subroutine Identification

```
USER SUBROUTINE FILE = <string>file_name
```

This command line is a part of a set of commands that are used to implement the user subroutine functionality. The string file_name identifies the name of the file that contains the FORTRAN code of one or more user subroutines.

To understand how this command line is used, see Chapter 11.

2.1.5 Functions

```
BEGIN DEFINITION FOR FUNCTION <string>function_name
  TYPE = <string>CONSTANT|PIECEWISE LINEAR|PIECEWISE CONSTANT|
    ANALYTIC|PIECEWISE ANALYTIC
  ABSCISSA = <string>abscissa_label
    [scale = <real>abscissa_scale(1.0)]
    [offset = <real>abscissa_offset(0.0)]
  ORDINATE = <string>ordinate_label
    [scale = <real>ordinate_scale(1.0)]
    [offset = <real>ordinate_offset(0.0)]
  X SCALE = <real>x_scale(1.0)
  X OFFSET = <real>x_offset(0.0)
  Y SCALE = <real>y_scale(1.0)
  Y OFFSET = <real>y_offset(0.0)
  BEGIN VALUES
    <real>x_1   <real>y_1
    <real>x_2   <real>y_2
    ...
    <real>x_n   <real>y_n
  END [VALUES]
  BEGIN EXPRESSIONS
    <real>x_1   <string>analytic_expression_1
    <real>x_2   <string>analytic_expression_2
    ...
    <real>x_n   <string>analytic_expression_n
  END
  AT DISCONTINUITY EVALUATE TO <string>LEFT|RIGHT(LEFT)
  EVALUATE EXPRESSION = <string>analytic_expression1;
    analytic_expression2; ...
  DEBUG = ON|OFF(OFF)
END [DEFINITION FOR FUNCTION <string>function_name]
```

A number of Sierra/SM features are driven by a user-defined description of the dependence of one variable on another. For instance, the prescribed displacement boundary condition requires the definition of a time-versus-displacement relation, and the thermal strain computations require the definition of a thermal-strain-versus-temperature relation. SIERRA provides a general method of defining these relations as functions using the `DEFINITION FOR FUNCTION` command block, as shown above.

There is no limit to the number of functions that can be defined. All function definitions must appear within the SIERRA scope.

A description of the various parts of the `DEFINITION FOR FUNCTION` command block follows:

- The string `function_name` is a user-selected name for the function that is unique to the function definitions within the input file. This name is used to refer to this function in other locations in the input file.

- The `TYPE` command line has five options to define the type of function. The value of this string can be `CONSTANT`, `PIECEWISE LINEAR`, `PIECEWISE CONSTANT`, `ANALYTIC`, or `PIECEWISE ANALYTIC`.

- The `ABSCISSA` command line provides a descriptive label for the independent variable (*x*-axis) with the string `abscissa_label`. This command line is optional. The user can optionally add a scale factor and/or an offset which has the following effect: $abscissa_{scaled} = scale * (abscissa + offset)$.

- The `ORDINATE command` line provides a descriptive label for the dependent variable (*y*-axis) with the string `ordinate_label`. This command line is optional. The user can also optionally add a scale factor and/or an offset which has the following effect: $ordinate_{scaled} = scale * (ordinate + offset)$.

- The `X SCALE` command line sets the scale factor value for the abscissa and has the same effect as if the optional `SCALE` command were used in the `ABSCISSA` command line.

- The `X OFFSET` command line sets the offset value for the abscissa and has the same effect as if the optional `OFFSET` command were used in the `ABSCISSA` command line.

- The `Y SCALE` command line sets the scale factor value for the ordinate and has the same effect as if the optional `SCALE` command were used in the `ORDINATE` command line.

- The `Y OFFSET` command line sets the offset value for the ordinate and has the same effect as if the optional `OFFSET` command were used in the `ORDINATE` command line.

- The `DEBUG command` line prints functions to the log file if they were scaled and/or offset. This command line is optional. The generated function function name is the original function name concatenated with `_with_scale_and_offset_applied`. The generated function is a valid function and could be placed into the input file and used.

- The `VALUES` command block consists of the real value pairs (`x_1`,`y_1`) through (`x_n`, `x_n`), which describe the function. This command block must be used if the value on the `TYPE` command line is `CONSTANT`, `PIECEWISE LINEAR`, or `PIECEWISE CONSTANT`. For a `CONSTANT` function, only one value is needed. For a `PIECEWISE LINEAR` or `PIECEWISE CONSTANT` function, the values are (x, y) pairs of data that describe the function. The values are nested inside the `VALUES` command block.

  A `PIECEWISE LINEAR` function performs linear interpolations between the provided value pairs; a `PIECEWISE CONSTANT` function is constant valued between provided value pairs. Figure 2.1 (a) shows an example of a piecewise linear function, and Figure 2.1 (b) shows an example of a piecewise constant function.

  For functions that are based on tabular values, such as the `PIECEWISE LINEAR` and `PIECEWISE CONSTANT` functions, there is a possibility that the function may be evaluated

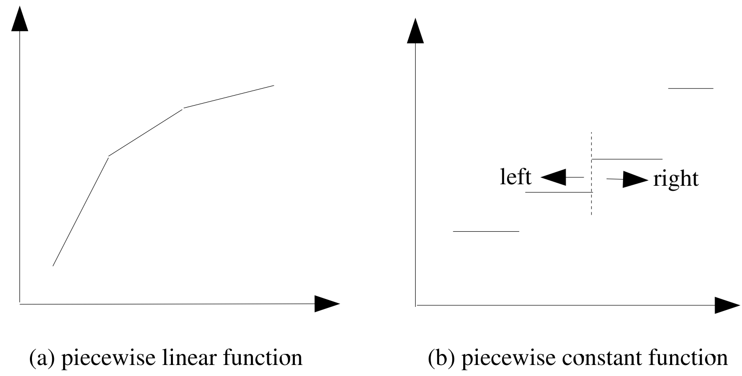


Figure 2.1: Piecewise linear and piecewise constant functions

for abscissa values that fall outside the range of the tabulated values. If the abscissa for which the function is to be evaluated is lower than the lowest tabulated abscissa, the function returns the ordinate corresponding to the smallest tabulated abscissa. Likewise, if the function is evaluated for an abscissa higher than the highest tabulated abscissa, the function returns the ordinate corresponding to the highest tabulated abscissa. For example, consider the following function:

```
begin definition for function my_func
  type = piecewise linear
  begin values
    5.0  0.0
    10.0 50000.0
  end values
end definition for function my_func
```

For values less than 5, this function returns 0, and for values greater than 10, it returns 50000.

- For a piecewise constant function, a constant valued segment ends on the left hand side of an abscissa value and a new constant value segment begins on the right hand side of the same abscissa value. (This transition from one constant value to another is indicated by the dotted line in Figure 2.1 (b).) When the value of the function is to be evaluated at a discontinuity, where there two potential values for the ordinate, the default behavior is to use the ordinate from the value pair that has the lower-valued abscissa, or in other words, to use the value on the left hand side of the discontinuity. The `AT DISCONTINUITY EVALUATE TO` command line can be used to override this default behavior at an abscissa with two ordinate values. The command line can have a value of either `LEFT` or `RIGHT`. If `LEFT` (the default) is specified, the ordinate value to the left of the abscissa is used; if `RIGHT` is specified, the ordinate value to the right of the abscissa is used.

- The `EVALUATE EXPRESSION` command line consists of one or more user-supplied algebraic expressions. This command line must be used if the value on the `TYPE` command line is `ANALYTIC`. See the rules and options for composing algebraic expressions discussed below.

Importantly, a `DEFINITION FOR FUNCTION` command block cannot contain both a `VALUES` command block and an `EVALUATE EXPRESSION` command line.

Rules and options for composing algebraic expressions. If you choose to use the `EVALUATE EXPRESSION` command line, you will need to write the algebraic expressions. The algebraic expressions are written using a C-like format. Each algebraic expression is terminated by a semicolon(;). The entire set of algebraic expressions, whether a single expression or several, is enclosed in a single set of double quotes(" ").

An expression is evaluated with `x` as the independent variable. We first provide several simple examples and then list the options available in the algebraic expressions.

Example: Return `sin(x)` as the value of the function.

```
begin definition for function fred
  type is analytic
  evaluate expression is "sin(x);"
end definition for function fred
```

In this example, the commented out table is equivalent to the evaluated expression:

```
begin definition for function pressure
  type is analytic
  evaluate expression is "x <= 0.0 ? 0.0 : (x < 0.5 ? x*200.0
    : 100.0);"
    #     begin values
    #       0.0      0.0
    #       0.5    100.0
    #       1.0    100.0
    #     end values
end definition for function pressure
```

The following functionality is currently implemented for the expressions:

Operators

```
+ - * / == != > < >= <= ! & | && || ? :
```

Parentheses

```
()
```

Math functions

```
abs(x), absolute value of x
mod(x, y), modulus of x|y
ipart(x), integer part of x
fpart(x), fractional part of x
```

Power functions

```
pow(x, y), x to the y power
pow10(x), x to the 10 power
sqrt(x), square root of x
```

Trigonometric functions

```
acos(x), arccosine of x
asin(x), arcsine of x
atan(x), arctangent of x
atan2(y, x), arctangent of y/x, signs of x and y
  determine quadrant (see atan2 man page)
cos(x), cosine of x
cosh(x), hyperbolic cosine of x
sin(x), sine of x
sinh(x), hyperbolic sine of x
tan(x), tangent of x
tanh(x), hyperbolic tangent of x
```

Logarithm functions

```
log(x), natural logarithm of x
ln(x), natural logarithm of x
log10(x), the base 10 logarithm of x
exp(x), e to the x power
```

Rounding functions

```
ceil(x), smallest integral value not less than x
floor(x), largest integral value not greater than x
```

Random functions

```
rand(), random number between 0.0 and 1.0, not including 1.0
randomize(), random number between 0.0 and 1.0, not
  including 1.0
srand(x), seeds the random number generator
```

Conversion functions

```
deg(x), converts radians to degrees
rad(x), converts degrees to radians
recttopolr(x, y), magnitude of vector x, y
recttopola(x, y), angle of vector x, y
poltorectx(r, theta), x coordinate of angle theta at
  distance r
poltorecty(r, theta), y coordinate of angle theta at
  distance r
```

Constants. There are two predefined constants that may be used in an expression. These two constants are `e` and `pi`.

```
e = e = 2.7182818284...
pi = π = 3.1415926535...
```

The `PIECEWISE ANALYTIC` function is a combination of several analytic functions that are evaluated over different ranges of x. As an example consider the following function. The function ramps up a load from time 0.0 to 1.0, holds that load constant from time 1.0 to 2.0, ramps the load back down to zero from time 2.0 to 3.0, and then holds the load constant again after time 3.0.

```
begin definition for function force_ramp
  type is piecewise analytic
  begin expressions
    0.0 "(1.0-cos(pi*x))/2.0"
    1.0 "1.0"
    2.0 "(1.0+cos(pi*(x-2.0)))/2.0"
    3.0 "0.0"
  end
end
```

Note this function is equivalent to the single compound, `ANALYTIC` function below, though the piecewise version tends to be much easier to read.

```
begin definition for function force_ramp
  type is analytic
  evaluate expression = "x <= 1.0 ?
                         (1.0-cos(pi*x))/2.0 : (x < 2.0 ?
                         1.0 : (x < 3.0 ?
                         (1.0+cos(pi*(x-2.0)))/2.0 :(0.0)))"
end
```

2.1.6 Axes, Directions, and Points

```
DEFINE POINT <string>point_name WITH COORDINATES
  <real>value_1 <real>value_2 <real>value_3
DEFINE DIRECTION <string>direction_name WITH VECTOR
  <real>value_1 <real>value_2 <real>value_3
DEFINE AXIS <string>axis_name WITH POINT
  <string>point_1 POINT <string>point_2
DEFINE AXIS <string>axis_name WITH POINT
  <string>point DIRECTION <string>direction
```

A number of Sierra/SM features require the definition of geometric entities. For instance, the prescribed displacement boundary condition requires a direction definition, and the cylindrical velocity initial condition requires an axis definition. Currently, Sierra/SM input permits the definition of points, directions, and axes. Definition of these geometric entities occurs in the SIERRA scope.

The `DEFINE POINT` command line is used to define a point:

```
DEFINE POINT <string>point_name WITH COORDINATES
  <real>value_1 <real>value_2 <real>value_3
```

where

- The string `point_name` is a name for this point. This name must be unique to all other points defined in the input file.
- The real values `value_1`, `value_2`, and `value_3` are the x, y, and z coordinates of the point.

The `DEFINE DIRECTION` command line is used to define a direction:

```
DEFINE DIRECTION <string>direction_name WITH VECTOR
  <real>value_1 <real>value_2 <real>value_3
```

where

- The string `direction_name` is a name for this direction. This name must be unique to all other directions defined in the input file.
- The real values `value_1`, `value_2`, and `value_3` are the x, y, and z magnitudes of the direction vector.

There are two command lines that can be used to define an axis. The first `DEFINE AXIS` command line uses two points:

```
DEFINE AXIS <string>axis_name WITH POINT
  <string>point_1 POINT <string>point_2
```

where

- The string `axis_name` is a name for this axis. This name must be unique to all other axes defined in the input file.
- The strings `point_1` and `point_2` are the names for two points defined in the input file via a `DEFINE POINT` command line.

The second DEFINE AXIS command line uses a point and a direction:

```
DEFINE AXIS <string>axis_name WITH POINT
  <string>point DIRECTION <string>direction
```

where

- The string axis_name is a name for this axis. This name must be unique to all other axes defined in the input file.
- The string point is the name of a point defined in the input file via a DEFINE POINT command line.
- The string direction is the name of a direction defined in the input file via a DEFINE DIRECTION command line.

2.1.7 Orientation

```
BEGIN ORIENTATION <string>orientation_name
  SYSTEM = <string>RECTANGULAR|Z RECTANGULAR|CYLINDRICAL|
    SPHERICAL(RECTANGULAR)
  #
  POINT A = <real>global_ax <real>global_ay <real>global_az
  POINT B = <real>global_bx <real>global_by <real>global_bz
  #
  ROTATION ABOUT <integer> 1|2|3(2) = <real>theta(0.0)
END [ORIENTATION <string>orientation_name]
```

The ORIENTATION command block is currently used in Sierra/SM to define a local *co-rotational* coordinate system at each shell element centroid for output of shell in-plane stresses and strains. Each BEGIN SHELL SECTION command block has an orientation that generates this local coordinate system for each element in the associated element blocks. The generated local coordinate system rotates with the shell element, thus making it *co-rotational*.

Stresses and strains for shell elements are computed in the shell element's local coordinate system, which typically varies from one element to the next, and can be seen in Figure 2.2. The local **R** axis of element 1 does not align with the local **R** axis of element 2. Thus without the use of an orientation to rotate the local coordinate system in to a consistent coordinate system, stress and strain results would be difficult to decipher. The use of an orientation generates a transformation matrix from a shell element's local **R**, **S**, **T** coordinate system to the final coordinate system, **R**′, **S**′, **T**′, by use of orientation and element-specific basis vectors and a rotation about an axis, all defined below.

There are four orientation systems available in Sierra/SM to better align the stress and strain output of shell elements. These orientations are defined via the optional SYSTEM command line within the ORIENTATION command block and are: RECTANGULAR, Z_RECTANGULAR, CYLINDRICAL and

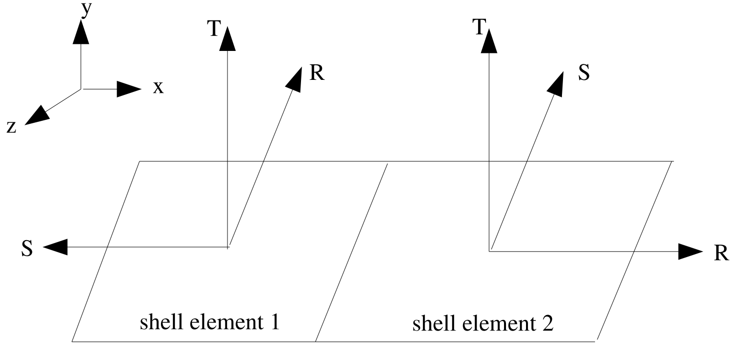


Figure 2.2: Adjacent shell elements with nonaligned local coordinate systems

SPHERICAL with the default being RECTANGULAR. If the ORIENTATION command line within a SHELL SECTION block is not used, the default orientation is a RECTANGULAR system with axes aligned with the global X,Y and Z axes.

Each of the four systems produces a unique set of bases vectors, $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3$, computed using the centroid of the shell element and the two required inputs, POINT A and POINT B. These four methods of computing $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3$ are described below.

- RECTANGULAR: POINT A defines a point, P_A, in the local coordinate system of the shell element that lies in the direction of the the first basis vector, $\mathbf{g}_1$, from the centroid, C with local coordinate of $(0, 0, 0)$, see Figure 2.3. Thus, the vector from the centroid of the element to point P_A is defined as $\mathbf{p}_A = P_A - C$. Similarly, a vector from the centroid to point P_B, defined via POINT B, can be computed as $\mathbf{p}_B = P_B - C$. Using these two vectors, the rectangular basis vectors are defined as: $\mathbf{g}_1 = \frac{\mathbf{p}_A}{\|\mathbf{p}_A\|}$, $\mathbf{g}_3 = \frac{\mathbf{g}_1 \times \mathbf{p}_B}{\|\mathbf{g}_1 \times \mathbf{p}_B\|}$, $\mathbf{g}_2 = \frac{\mathbf{g}_3 \times \mathbf{g}_1}{\|\mathbf{g}_3 \times \mathbf{g}_1\|}$.
- Z RECTANGULAR: This set of basis vectors is defined very similarly to the RECTANGULAR system, however point P_A now defines a point that lies in the direction of the third basis vector, $\mathbf{g}_3$ from the centroid of the element, see Figure 2.4. Using the same definition for $\mathbf{p}_A$ and $\mathbf{p}_B$ as above we can define the three basis vectors as: $\mathbf{g}_3 = \frac{\mathbf{p}_A}{\|\mathbf{p}_A\|}$, $\mathbf{g}_2 = \frac{\mathbf{g}_3 \times \mathbf{p}_B}{\|\mathbf{g}_3 \times \mathbf{p}_B\|}$, $\mathbf{g}_1 = \frac{\mathbf{g}_2 \times \mathbf{g}_3}{\|\mathbf{g}_2 \times \mathbf{g}_3\|}$.
- CYLINDRICAL: POINT A, P_A, and POINT B, P_B, define the direction of the third basis vector, $\mathbf{g}_3$ such that $\mathbf{g}_3 = \frac{P_B - P_A}{\|P_B - P_A\|}$, as seen in Figure 2.5. Defining a vector, $\mathbf{v}$, from P_A to the centroid of the element, C in the global coordinate system, the second basis vector, $\mathbf{g}_2$ is defined as: $\mathbf{g}_2 = \frac{\mathbf{v} \times \mathbf{g}_3}{\|\mathbf{v} \times \mathbf{g}_3\|}$. Therefore the first basis vector is defined as $\mathbf{g}_1 = \frac{\mathbf{g}_2 \times \mathbf{g}_3}{\|\mathbf{g}_2 \times \mathbf{g}_3\|}$.
- SPHERICAL: The point P_A, from the POINT A command line, defines the center of a sphere. The point P_B, from the POINT B command line, defines a polar axis for the sphere. (See Figure 2.6.) The first basis vector is defined by the vector that passes from P_A to the centroid, C in the global coordinate system, of the element: $\mathbf{g}_1 = \frac{C - P_A}{\|C - P_A\|}$. The third basis vector is

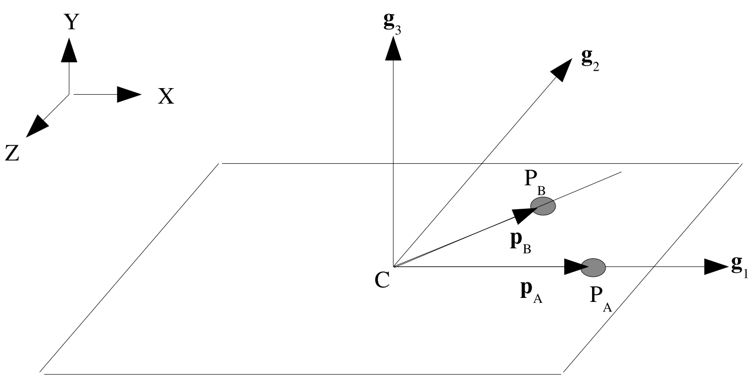


Figure 2.3: Rectangular coordinate system

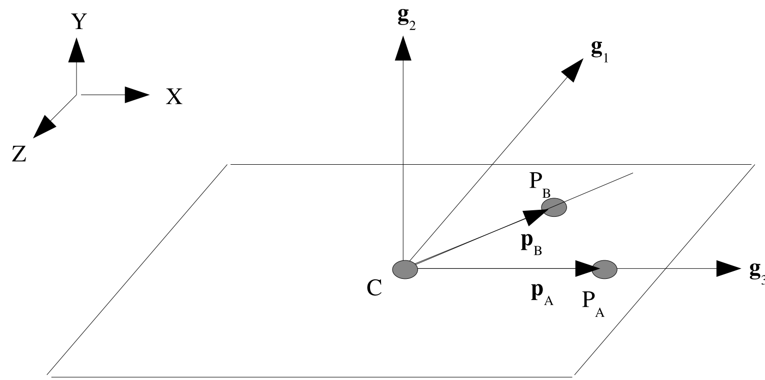


Figure 2.4: Z-Rectangular coordinate system.

parallel to the vector from P_A to P_B such that $\mathbf{g}_3 = \frac{P_B - P_A}{||P_B - P_A||}$. From these two vector, the second basis vector is defined as $\mathbf{g}_2 = \frac{\mathbf{g}_3 \times \mathbf{g}_1}{||\mathbf{g}_3 \times \mathbf{g}_1||}$. If $\mathbf{g}_1$ and $\mathbf{g}_3$ are collinear, the second basis vector is taken to be the normal of the element, $\mathbf{T}$, and the third basis is $\mathbf{g}_3 = \frac{\mathbf{g}_1 \times \mathbf{g}_2}{||\mathbf{g}_1 \times \mathbf{g}_2||}$.

It is possible, using the `ROTATION ABOUT` command line, to specify an angle and an axis to rotate the basis vectors, $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_3$ about in order to produce a second set of basis vectors, $\mathbf{g}_1'$, $\mathbf{g}_2'$ and $\mathbf{g}_3'$, that are then used to compute the final transformation matrix. The syntax for this command is as follows: `ROTATION ABOUT 1|2|3(2) = <real>theta(0.0)`

Where the 1,2,3 refers to which basis vector the rotation will occur about. The default value is 2 with an angle theta of 0.0, thus implying a rotation of 0.0 radians about $\mathbf{g}_2$. How $\mathbf{g}_1'$, $\mathbf{g}_2'$ and $\mathbf{g}_3'$ are computed for each of the three cases is described below. In each case, the second order rotation

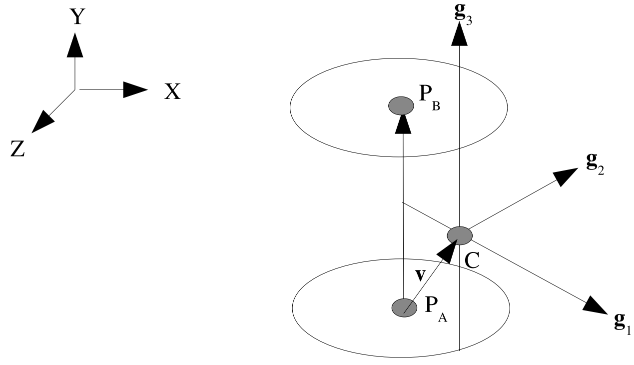


Figure 2.5: Cylindrical coordinate system.

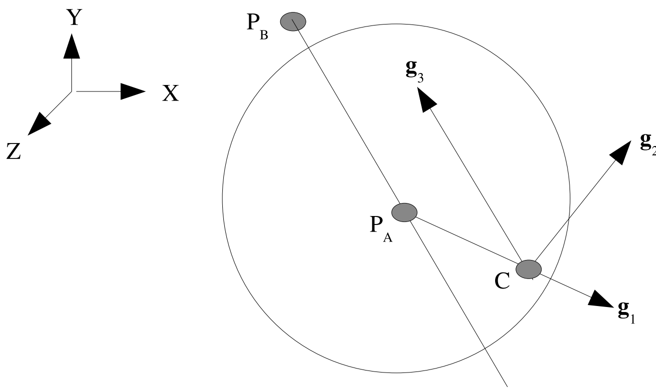


Figure 2.6: Spherical coordinate system.

tensor, **Q**, is defined as:

$$\mathbf{Q} = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 - q_0 q_3) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_o^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix}$$

Where: $q_0 = cos(\frac{\theta}{2})$, $q_1 = a_1 sin(\frac{\theta}{2})$, $q_2 = a_2 sin(\frac{\theta}{2})$, and $q_3 = a_3 sin(\frac{\theta}{2})$, where the vector $\mathbf{a} = [a_1, a_2, a_3]$ is the basis vector we are rotating about.

- Rotation about $\mathbf{g}_1$ (`ROTATION ABOUT 1`): The prime set of basis vectors are defined as: $\mathbf{g}_1' = \mathbf{Q}\mathbf{g}_3$, $\mathbf{g}_2' = \mathbf{Q}\mathbf{g}_2$, $\mathbf{g}_3' = -\mathbf{g}_1$. Which can graphically be seen in Figure 2.7
- Rotation about $\mathbf{g}_2$ (`ROTATION ABOUT 2`), default: The prime set of basis vectors are defined as: $\mathbf{g}_1' = \mathbf{Q}\mathbf{g}_1$, $\mathbf{g}_2' = \mathbf{Q}\mathbf{g}_3$, $\mathbf{g}_3' = -\mathbf{g}_2$.

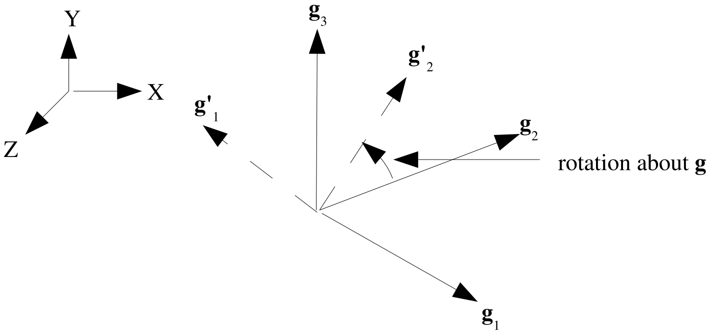


Figure 2.7: Rotation about 1

- Rotation about $\mathbf{g}_3$ (ROTATION ABOUT 3): The prime set of basis vectors are defined as: $\mathbf{g}_1' = \mathbf{Q}\mathbf{g}_2$, $\mathbf{g}_2' = \mathbf{Q}\mathbf{g}_1$, $\mathbf{g}_3' = -\mathbf{g}_3$.

The next step is to project $\mathbf{g}_1'$ onto the shell face as follows: $\mathbf{R}' = \mathbf{g}_1' - (\mathbf{g}_1' \cdot \mathbf{T})\mathbf{T}$, which defines a rotated $\mathbf{R}$ vector. If $\mathbf{R}'$ is perpendicular to the face of the element, we then redefine $\mathbf{R}'$ to be: $\mathbf{R}' = \mathbf{g}_2' - (\mathbf{g}_2' \cdot \mathbf{T})\mathbf{T}$, the projection of $\mathbf{g}_2'$ onto the element face. Once we have $\mathbf{R}'$ we can define $\mathbf{S}'$ as $\frac{\mathbf{T}\times\mathbf{R}'}{\|\mathbf{T}\times\mathbf{R}'\|}$. Hence, the final transformation matrix taking local stress and strain (in the $\mathbf{R}$, $\mathbf{S}$ coordinate system) to the $\mathbf{R}'$, $\mathbf{S}'$ coordinate system is:

$$\mathbf{Q}' = \begin{bmatrix} \mathbf{R}\cdot\mathbf{R}' & \mathbf{R}\cdot\mathbf{S}' & 0 \\ \mathbf{S}\cdot\mathbf{R}' & \mathbf{S}\cdot\mathbf{S}' & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.1.8 Coordinate System Block Command

```
BEGIN COORDINATE SYSTEM <string>coordinate_system_name
  TYPE = <string>RECTANGULAR|CYLINDRICAL|SPHERICAL (RECTANGULAR)
  #
  ORIGIN = <string>origin_point_name
  VECTOR = <string>z_vector_point_name
  POINT = <string>x_vector_point_name
  ORIGIN NODE = <string>origin_nodelist_name
  VECTOR NODE = <string>z_vector_nodelist_name
  POINT NODE = <string> x_vector_nodelist_name
  #
END [COORDINATE SYSTEM <string>coordinate_system_name]
```

The COORDINATE SYSTEM command block is used to define a local coordinate system for transforming a stress tensor from components in the global xyz coordinate system to the local system

for output. This command block cannot be used for in-plane stresses and strains of shell elements because the shell integration point stresses and strains are computed in a local element system that varies element to element and rotates with the element. For output of shell stresses, the BEGIN ORIENTATION command must be used as explained in the previous section.

The COORDINATE SYSTEM block is also used to define a local coordinate system on the elements of a block for use in a representative volume analysis. In these analyses, elements of each representative volume will be aligned with the local system defined on the parent element.

A local element coordinate system may be defined using points defined in the input file with the DEFINE POINT command or using nodelists in the mesh file that contain exactly one node each. By using nodes in the mesh file, the defined coordinate system may translate and rotate during the analysis.

The TYPE command line allows three options for constructing a local coordinate system: RECTANGULAR, CYLINDRICAL, and SPHERICAL. The type defaults to RECTANGULAR if the TYPE command line is not present.

- RECTANGULAR: The command line ORIGIN or ORIGIN NODE specifies the origin of the local Cartesian coordinate system. The local Z' axis is a vector from the origin to the point given in the VECTOR of VECTOR NODE command. The local X' axis is the component of the vector from the origin to the POINT or POINT NODE that is orthogonal to the Z' axis. Finally, the local Y' axis is obtained from the cross product of Z' and X'.

- CYLINDRICAL: The command line ORIGIN or ORIGIN NODE specifies one point on the axis of the cylinder. The local Z' axis, the axis of the cylinder, is a vector from the origin to the point given in the VECTOR or VECTOR NODE command. The local X' axis is constructed as the vector normal to the cylindrical axis and passing through the location at which the local system is desired. Depending on the context, this local point may be a node, an element centroid, or an element integration point. If this local point lies on the Z' axis, then the point defined by the POINT or POINT NODE command is instead used to define the X' axis. Finally, the Y' axis is obtained from the cross product of Z' and X'. Thus at the desired point not on the cylinder axis, the X' axis is through the cylinder thickness, Y' is tangent to the cylinder, and Z' is parallel to the cylinder axis.

- SPHERICAL: The command line ORIGIN or ORIGIN NODE specifies the location of the center of the sphere. The local X' axis is constructed as the vector from the center through the location at which the local system is desired. Depending on the context, this local point may be a node, an element centroid, or an element integration point. The local Z' axis is the component of the vector from the origin to the point given in the VECTOR or VECTOR NODE command that is normal to the X' axis. If this Z' is parallel the X' axis, then the Z' axis is defined along the vector from the point defined in POINT or POINT NODE to the origin. Finally, the Y' axis is obtained from the cross product of Z' and X'. Thus at a desired point, the X' axis is through the sphere thickness and the Y' and Z' axes lie in the tangent plane.

2.1.9 Define Coordinate System Line Command

```
DEFINE COORDINATE SYSTEM <string>coord_sys_name <string>coord_sys_type
  WITH POINT <string>point_1 POINT <string>point_2
  POINT <string>point_3
```

The line command `DEFINE COORDINATE SYSTEM` can also be used to define the axis directions of a local coordinate system to be located at nodes, element centroids, or element integration points. In this command

- The string `coord_sys_name` is a name for this coordinate system. This name must be unique to all other coordinate systems defined in the input file.
- The string `coord_sys_type` states the type of the coordinate system to be used. Three options are allowed for constructing a local coordinate system: `RECTANGULAR`, `CYLINDRICAL`, and `SPHERICAL`.
- The strings `point_1`, `point_2`, and `point_3` are the names for three points defined in the input file via `DEFINE POINT` command lines. These three points are used to define two of the coordinate system axes as described below for the different types of systems.

For a rectangular system, the local Z' axis is parallel to the vector from the `point_1` to `point_2`. The local X' axis is the component of the vector from `point_1` to `point_3` that is orthogonal to the Z' axis. Finally, the local Y' axis is obtained from the cross product of Z' and X'.

Likewise, for a cylindrical system, the local Z' axis is parallel to the vector from the `point_1` to `point_2` and defines the axis of the cylinder. The local X' axis is constructed as the vector normal to the cylindrical axis and passing through the location at which the local system is desired. This local point may be a node, an element centroid, or an element integration point. If this local point lies on the Z' axis, then the point `point_3` is instead used to define the X' axis. Finally, the Y' axis is obtained from the cross product of Z' and X'. Thus at the desired point not on the cylinder axis, the X' axis is through the cylinder thickness, Y' is tangent to the cylinder, and Z' is parallel to the cylinder axis.

For a spherical system, `point_1` specifies the center of a sphere. The local X' axis is constructed as the vector from `point_1` through the location at which the local system is desired. This local point may be a node, an element centroid, or an element integration point. The local Z' axis is the component of the vector from the `point_1` to `point_3` that is normal to the X' axis. If this Z' is parallel the X' axis, then `point_3` is used instead. Finally, the Y' axis is obtained from the cross product of Z' and X'. Thus at a desired point, the X' axis is through the sphere thickness and the Y' and Z' axes lie in the tangent plane.

2.2 Procedure and Region

Explicit dynamics analyses use a Presto procedure and region. The Presto procedure scope is nested within the SIERRA scope, and the Presto region scope is nested within the procedure scope (see Section 1.2 for more information about scope). To create the scopes for the Presto procedure and Presto region for an explicit dynamics analysis, use the following commands:

```
BEGIN PRESTO PROCEDURE <string>procedure_name

  PRINT BANNER INTERVAL = <integer>print_banner_interval(MAX_INT)

  #
  # TIME CONTROL command block
  #
  BEGIN PRESTO REGION <string>region_name
    #
    # command blocks and command lines that appear in the
    # region scope
    #
  END [PRESTO REGION <string>region_name]
END [PRESTO PROCEDURE <string>procedure_name]
```

The PRINT BANNER INTERVAL line command sets the number of PRESTO output lines to print before re-printing the column headings. The default value is set to the maximum allowed integer value. This means that the banner will be printed to the log file one time by default.

Implicit analyses follow a very similar structure, but use an Adagio procedure and region. The following example shows the structure of an input file for an implicit analysis:

```
BEGIN ADAGIO PROCEDURE <string>procedure_name
  #
  # TIME CONTROL command block
  #
  BEGIN ADAGIO REGION <string>region_name
    #
    # command blocks and command lines that appear in the
    # region scope
    #
  END [ADAGIO REGION <string>region_name]
END [ADAGIO PROCEDURE <string>region_name]
```

For both implicit and explicit analyses, the TIME CONTROL command block appears within the PRESTO PROCEDURE or ADAGIO PROCEDURE command block but outside of the PRESTO REGION or ADAGIO REGION command block. These three command blocks (procedure, time control, and region) are discussed below.

Many command blocks and command lines fall within the region scope. These command blocks and command lines are described in other sections of this document.

2.2.1 Procedure

The analysis time, from the initial time to the termination time, is controlled within the procedure scope defined by the PRESTO PROCEDURE or ADAGIO PROCEDURE command block. For explicit dynamic analyses, the command block begins with an input line of the form:

```
BEGIN PRESTO PROCEDURE <string>procedure_name
```

and is terminated with an input line of the following form:

```
END [PRESTO PROCEDURE <string>procedure_name]
```

Implicit analyses use an Adagio procedure, which starts with the following line:

```
BEGIN ADAGIO PROCEDURE <string>procedure_name
```

and ends with the following line:

```
END [ADAGIO PROCEDURE <string>procedure_name]
```

The string procedure_name is the name for the procedure.

2.2.2 Time Control

Within the procedure scope, there is a TIME CONTROL command block. This command block lets the user set the initial time and the termination time for an analysis. This block also allows the user to control the size of the time step. The time step is controlled in different ways for explicit and implicit analyses, as explained below.

2.2.2.1 Explicit Dynamic Time Control

When Sierra/SM is used to run explicit transient dynamics analyses, it must run at a time step that is less than the critical time for the problem at any given instant. Typically, this global critical time step is based on a critical time step estimate calculated for each element. With the TIME CONTROL command block, the user can set an initial time step, scale the element-based time step estimate, and control the growth of the element-based estimate for the critical time step.

In addition to the element-based method for estimating the critical time step, Sierra/SM offers other methods for estimating the critical time step. One approach for estimating the critical time step is to calculate the maximum eigenvalue for the model. There are two methods for calculating the maximum eigenvalue: the Lanczos method and the power method. A second approach for estimating the critical time step is to use a node-based method. The command blocks for implementing these various methods (maximum eigenvalue calculation and node-based) are in the region scope. There is also a mass-scaling technique that will influence the magnitude of the critical time step. If you use the mass-scaling technique, you must use the node-based method to obtain a critical time step estimate.

The estimation of the time step is a key part of any explicit Sierra/SM analysis. Time step determination and control is discussed in detail in Chapter 2.7 of this document. The TIME CONTROL

command block with its associated command lines are described in detail in Chapter 2.7. Consult Chapter 2.7 to determine how to specify command lines associated with the TIME CONTROL command block and how the TIME CONTROL command block fits into the overall scheme for time step control in Presto. Also see Chapter 2.7 to learn about the other methods for estimating the critical time step and the mass-scaling technique.

2.2.2.2 Implicit Time Control

When running implicit dynamic or quasistatic analyses with Sierra/SM a solver is used to find a converged solution at every time step. The time steps can be arbitrarily large, although errors due to time discretization are larger with larger time steps. In addition, for nonlinear problems, it is often more difficult to obtain a converged solution with large time steps. The time step size is controlled by the user, although Sierra/SM has a capability to optionally modify the time step based on the effort required by the nonlinear solver.

The TIME CONTROL block contains the basic commands used to control the time step size. To use adaptive time stepping, an ADAPTIVE TIME STEPPING command block is placed in the Adagio region scope. In addition, for all implicit analyses, commands to control the solver are required in the Adagio region scope. The details of the commands related to time stepping and the nonlinear solver are documented in Chapter 3.5.

2.2.3 Region

Individual time steps are controlled within the region scope. For explicit analyses, the region scope is defined by a PRESTO REGION command block that begins with an input line of the form

```
BEGIN PRESTO REGION <string>region_name
```

and is terminated with an input line of the following form:

```
END [PRESTO REGION <string>region_name]
```

Implicit analyses define the region scope using a ADAGIO REGION command block that begins with the following line:

```
BEGIN ADAGIO REGION <string>region_name
```

and is terminated by the following line:

```
END [ADAGIO REGION <string>region_name]
```

The string region_name is the name for this region.

The region, as indicated previously, determines what happens at each time step. In the procedure, we set the begin time and end time for the analysis. Time is incremented in the region. It is in the region where we set information about what occurs at various time steps. The output of results, for example, is set by command blocks in the region. If we want results output at certain times or certain steps in the analysis, this information is set in command blocks in the region. The region also contains command blocks for the boundary conditions. A boundary condition can have a

time-varying component. The region determines the value of the component for the current time step.

Two of the major types of command blocks, those for results output and boundary conditions, have already been mentioned. Other major types of command blocks in the region are those for restart control and contact. The region is also where the user selects the analysis model (finite element mesh). For implicit analyses, the command blocks for the solver are in the region. These blocks are discussed in Chapter 3.5.

The region makes use of information in the procedure and the SIERRA scope. For example, the specific element type used for an element block in the analysis model is defined in the SIERRA scope. This information about the element type is collected into an analysis model. The region then references this analysis model. As another example, the boundary condition command blocks can reference a function. The function will be defined in the SIERRA scope.

2.3 Use Finite Element Model

```
USE FINITE ELEMENT MODEL <string>model_name
```

The model specification occurs within the region scope. To specify the model (finite element mesh), use this command line. The string model_name must match a name used in a FINITE ELEMENT MODEL command block described in Section 6.1. If one of these command blocks uses the name penetrator in the command-block line and this is the model we wish to use in the region scope, then we would enter the command line as follows:

```
USE FINITE ELEMENT MODEL penetrator
```

2.4 Element Distortion Metrics

Sierra/SM can compute a number of element distortion metrics useful for assessing the quality of the solution as the mesh evolves over time. These metrics are computed as element variables, and can be used for output or as criteria for element death, just like any other element variable. The solution quality generally deteriorates as elements approach inversion, and if they do invert, the analysis aborts. The distortion metrics measure how close an element is to inversion. For all metrics a value of 1.0 is an ideal element and a value of 0.0 is a degenerate element. The following distortion metrics are available:

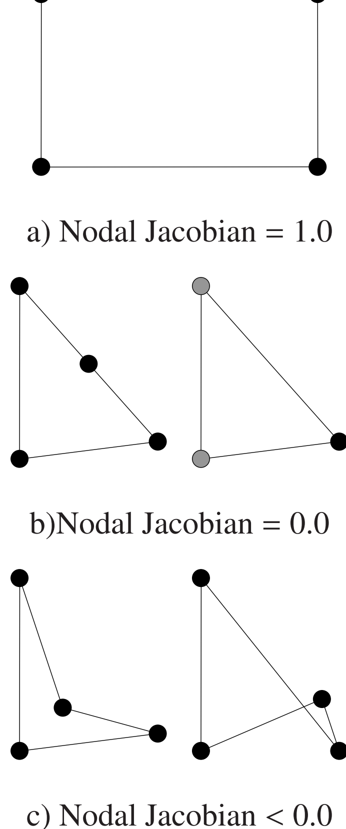
- `NODAL_JACOBIAN_RATIO` is currently available only for 8 node hexahedra. This metric evaluates the Jacobian function at each node of each element, and then computes the nodal Jacobian ratio as the smallest nodal Jacobian divided by the largest nodal Jacobian. An element having right angles between all adjacent edges has a nodal Jacobian ratio of 1.0. A negative nodal Jacobian ratio indicates that the element is becoming either convex or locally inverted. See Figure 2.8 for examples of quadrilateral elements with positive, zero, and negative nodal Jacobian ratios. The element calculations on poorly shaped elements (those with negative nodal Jacobian ratios) will generally be less accurate than those on well shaped elements (those with positive nodal Jacobian ratios). In addition, contact surfaces may become tangled and non-physical if elements start to invert.

- `ELEMENT_SHAPE` computes a general shape metric for a variety of element topologies. Is currently defined for 8 node hexahedra, 4 node tetrahedra, 4 node quads, and 3 node triangles. A value of 1.0 is optimal. A value of 0.0 represents a degenerate element. A negative value represents either a severely warped element or inverted element. For the 8 node hexahedron, the element shape is the nodal Jacobian ratio for the element. For a tetrahedron or triangle, the shape is proportional to the element aspect ratio. For the 4 node quad element, the shape is proportional to the amount of warping of the quad. If a mesh contains elements with negative `ELEMENT_SHAPE`, problems could arise in the material or contact computations.

- `ASPECT_RATIO` is available only for tetrahedral elements. A perfect equilateral tetrahedron has an aspect ratio of 1.0. A degenerate zero-volume tetrahedron has an aspect ratio of zero. An inverted tetrahedron has a negative aspect ratio. A very thin element can have a very small aspect ratio.

- `SOLID_ANGLE` computes the minimum or maximum angle between edges of an element as compared to optimal angles. The optimal solid angle for tetrahedra and triangles is 60 degrees; for hexahedra and quadrilaterals, it is 90 degrees. This error metric is 1.0 for an element in which all angles are optimal. Severely distorted or twisted elements have values of this metric near 0.0, and it is negative for inverted elements. This metric is 0 for degenerate elements. The `SOLID ANGLE` metric functions with hex, tet, triangle, and quad elements.

- `PERIMETER_RATIO` measures the the ratio of the deformed perimeter of an element to the undeformed perimeter of the element. This metric initially has a value of 1.0, and assumes values either greater than or lower than 1 as the mesh deforms. The `PERIMETER_RATIO` metric only works with triangle and quad elements.



a) Nodal Jacobian = 1.0

b)Nodal Jacobian = 0.0

c) Nodal Jacobian < 0.0

Figure 2.8: Examples of elements with varying nodal Jacobians

- DIAGONAL_RATIO measures the the ratio of the deformed max diagonal of an element to the undeformed max diagonal of the element. This metric initially has a value of 1.0, and assumes values either greater than or lower than 1 as the mesh deforms. The DIAGONAL_RATIO metric only works with hex and quad elements.

The results from any of these distortion metrics can be requested by specifying an ELEMENT VARIABLES command line (Section 9.2.1.4) in the RESULTS OUTPUT command block (Section 9.2.1) for each metric for which the results are of interest. For example, to request the ASPECT_RATIO metric to be output as an element variable named aspect in the results output file, the following syntax can be included in a RESULTS OUTPUT command block:

```
ELEMENT ASPECT_RATIO as aspect
```

The distortion metrics can also be used for element death (Section 6.5). For example, to kill all elements in which the value of the SOLID_ANGLE metric is less than 0.2, the following line can be used in an ELEMENT DEATH command block:

```
CRITERION IS ELEMENT VALUE OF SOLID_ANGLE < 0.2
```

For more information about the use of element variables for element death, see Section 6.5.2.2.

2.5 Activation/Deactivation of Functionality

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The ACTIVE PERIODS or INACTIVE PERIODS command line can be used to activate or deactivate functionality in the code at various points during an analysis. This functionality can include such things as boundary conditions, element blocks, and user subroutines. Command blocks that support this capability are documented accordingly in the sections of this manual where they are described.

In the command line, the string list period_names is a list of the time periods defined in TIME STEPPING BLOCK command blocks (see Sections 3.1 and 4.11) during which the particular functionality is considered to be active. Each such period_name must match a name used in a TIME STEPPING BLOCK command block, e.g., time_block_name. Each defined time period runs from that period's start time to the next period's start time.

Only one of ACTIVE PERIODS or INACTIVE PERIODS can be used in any given command block. If the ACTIVE PERIODS command line is present, the functionality will be treated as active for all of the named periods, and inactive for any time periods that are not listed. If the INACTIVE PERIODS command line is present, the functionality will be treated as inactive for all of the named periods and active for any period not listed. If neither of these command lines is present, by default the functionality is active during all time periods.

2.6 Error Recovery

```
ERROR TOLERANCE = ON|OFF (ON)
```

This command controls how aggressively Sierra/SM will attempt to recover from a non-physical or ill-defined condition and continue with the analysis.

Under the default setting of ON, the code may invoke methods to keep the analysis moving forward despite the presence of non-physical or ambiguous conditions. For example, if the contact algorithm determines that a facet is potentially in two contact surfaces (an ambiguous condition), Sierra/SM will assign the facet to one of the surfaces and continue with the analysis. A second example pertains to SPH particles: if the deformation gradient of an SPH particle is inverted, the particle will be ignored rather than causing the analysis to terminate. Non-physical and ambiguous conditions will generally result in a warning message being printed.

If ERROR TOLERANCE is OFF, Sierra/SM will terminate an analysis in the presence of ambiguous or non-physical conditions. This may make it easier for an analyst to address non-fatal but potentially problematic issues.

2.7 Manual Job Control

The output of a running job can be controlled externally through the use of “shutdown” and “control” files. This mechanism allows for additional output of restart, results, history, and/or heartbeat data to be requested, as well as an optional graceful shutdown of the job.

A graceful shutdown is requested by inserting a “shutdown” file in the working directory of a running Sierra/SM job. The name of this file can be any of the following:

```
sierra.shutdown
<application_name>.shutdown
<base_name>.shutdown
```

If the `<application_name>.shutdown` variant is used, `<application_name>` is the name of the Sierra application being run. For example, `presto.shutdown` would be used to shut down an analysis using the Presto executable, and `adagio.shutdown` would be used to shut down the Adagio executable. If the `<base_name>.shutdown` variant is used, `<base_name>` is the basename of the input file. For example, if the input file name is `my_analysis.i`, then the shutdown file would be `my_analysis.shutdown`.

If the code detects the existence of a shutdown file, it will dump an output step to any open restart, results, history, or heartbeat file and then gracefully terminate the job. An entry will be written to the log file specifying that a shutdown file was detected and the file will be deleted. The contents of the file are not important; the shutdown file is only checked for existence.

The control file capability is provided to give more control over output and execution than is possible with the shutdown file. The name of the control file is the same as that of the shutdown file except that the filename suffix is `.control` instead of `.shutdown`. The contents of the file consist of a single line, and are case insensitive. The syntax is:

```
DUMP [RESTART] [RESULTS] [HISTORY] [HEARTBEAT] [STEP] [TIME]
  [STOP|ABORT|SHUTDOWN|CONTINUE]
```

The optional strings `RESTART`, `RESULTS`, `HISTORY`, `HEARTBEAT`, `STEP`, and `TIME` are used to specify the type of output that is written before an action is taken. If any output blocks of the specified type were defined in the input file for the model, output will be written to the files. The `STEP` and `TIME` options result in the last step and time being written to the log file. Multiple output types may be requested. If no output type is requested, all types of output specified in the input file will be written.

In addition to controlling the type of output that occurs, the `.control` file also specifies whether the job should be terminated or allowed to continue. If `STOP`, `ABORT`, or `SHUTDOWN` is specified at the end of the line, the job will be gracefully terminated. If `CONTINUE` is specified or no option is specified, the job will continue.

Several examples of the contents of continue files are shown below. The following examples all result in output being written to all types of output file and the job continuing:

```
dump
dump continue
dump restart results history heartbeat step continue
```

In both of the following examples, the current step and time will be written to the the log file, but no additional output will be written and the job will continue:

```
dump step
dump time continue
```

This example would result in a write to all output types and a graceful shutdown:

```
dump stop
```

In the following example, no output would be written to any files, but the current step and time would be written to the log file before a graceful shutdown:

```
dump step abort
```

An alternate abbreviated syntax is also supported. The abbreviated commands are shown below along with the full commands to which they are mapped:

```
sw1          dump restart shutdown
sw2          dump step continue
sw3          dump restart continue
sw4          dump results continue
```

If either the shutdown or control files are used, a message is output to the log file listing the name of that file, and in the case of the control file, the contents of that file. The control or shutdown file is then deleted.

Explicit Only

Chapter 3

Explicit Dynamic Time Step Control

This chapter discusses time control for explicit dynamic Sierra/SM analyses. We begin with a broad overview of explicit dynamic time control and then describe the options that are available to users for time control. The commands and features documented in this chapter are applicable only for explicit dynamic analyses. For information on time stepping commands for implicit analyses, see Chapter 3.5.

The user initiates time control by setting a start time and a termination time for an analysis. The analysis is typically carried out with a large number of time steps, with each time step being much smaller than the analysis time. In explicit transient dynamics analyses, the time step must be less than a critical value. Sierra/SM has a number of methods for computing an estimate for the critical time step. These methods are discussed in detail in this chapter.

The primary time control uses a `TIME CONTROL` command block that appears in the procedure scope. Use of the `TIME CONTROL` command block gives the user, by default, access to an element-based method for estimating the critical time step. The user can access three other methods for estimating the critical time step by using specific command blocks that are placed in the region scope. These other methods tend to give better (larger) estimates for the critical time step. In addition, there is a technique for adjusting the time step known as mass scaling.

Section 3.1 describes the `TIME CONTROL` command block. In Section 3.2 we discuss the other methods for estimating the critical time step. One approach for improving this estimate is to compute the maximum eigenvalue for a problem. Two methods for computing the maximum eigenvalue are available: the Lanczos method and the power method. Section 3.2.1 discusses the Lanczos method; Section 3.2.1.5 describes the command block required to implement the Lanczos method. Section 3.2.2 discusses the power method; Section 3.2.2.5 describes the command block required to implement the power method. Another approach for improving the time step estimate relies on a node-based estimate. Section 3.2.3 discusses the node-based method; Section 3.2.3.1 describes the command block required to implement the node-based method. You should read the introductory material for the maximum eigenvalue calculation methods and the node-based method and understand this material thoroughly before you attempt to use these methods. Although these other methods give larger time step estimates than the element-based method, they may not result in a net reduction of central processor unit (CPU) time for an analysis unless they are used properly. In those sections dealing with these other methods, we discuss how to use these methods in a cost-effective manner. In Section 3.3 we describe the technique of mass scaling, which is a technique that artificially scales up nodal masses to allow larger time steps to be taken, and must be used with the node-based critical time step estimation method. Finally, in Section 3.4 we describe the explicit control modes algorithm, which uses coarse and fine meshes together to enable a

potentially significant increase in the critical time step.

3.1 Procedure Time Control

As indicated previously, the primary time control uses a TIME CONTROL command block in the procedure scope. The user sets the start time and the termination time for the analysis in the TIME CONTROL command block. For reference purposes, the general layout of the command block is as follows:

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK <string>time_block_name
    START TIME = <real>start_time_value
    BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
      INITIAL TIME STEP = <real>initial_time_step_value
      TIME STEP SCALE FACTOR =
        <real>time_step_scale_factor(1.0)
      TIME STEP INCREASE FACTOR =
        <real>time_step_increase_factor(1.1)
      STEP INTERVAL = <integer>nsteps(100)
      USER TIME STEP = <real>time_step
    END [PARAMETERS FOR PRESTO REGION <string>region_name]
  END [TIME STEPPING BLOCK <string>time_block_name]
  # Second TIME STEPPING BLOCK command block
  # would be placed here, as applicable.
  #
  # Additional TIME STEPPING BLOCK command blocks
  # would be placed here, as applicable.
  #
  # Last TIME STEPPING BLOCK command block
  # would be placed here, as applicable.
  #
  TERMINATION TIME = <real>termination_time
END [TIME CONTROL]
```

The analysis time, as demonstrated above, can be subdivided into a number of time blocks, i.e., TIME STEPPING BLOCK command blocks. If the total analysis time is from time 0 to time T and there are three blocks, then the first block is defined from time 0 to time t_1, the second block is defined from time t_1 to time t_2, and the third block is defined from time t_2 to time T. (The times t_1 and t_2 are set by the user.) If we sum all the times for each block, the sum will be T. The different time periods defined by each block can be referenced so that we can turn certain functionality on or off throughout an analysis. For example, we may want to have a certain boundary condition turned off during our first time period and activated for the second time period. (Most analyses require only one block.) Use the ACTIVE PERIODS or INACTIVE PERIODS command lines described in Section 2.5 to activate and deactivate functionality during specific time blocks.

By default, Sierra/SM relies on the element-based critical time step estimate. At every time step, an element-based calculation is performed to determine a critical time step. You have some control

over the actual time step that is used by employing one of two techniques. We discuss these two techniques in the following paragraphs. The specific command lines for using these techniques are described in Section 3.1.1.

With the `INITIAL TIME STEP` line command you can set an initial time step that is smaller than the element-based critical time step estimate. Sierra/SM will start the analysis by using your initial time step value instead of the element-based critical time step estimate (as long as your value is less than the element-based critical time step estimate). You can then control the rate at which the time step increases from your initial value. This technique is employed by using the `TIME STEP INCREASE FACTOR` command line as follows:

- If you set a time step increase factor equal to 1, then the initial value you specified will be used throughout the analysis (provided that the initial time step is never smaller than the element-based critical time step estimate throughout the computations).
- If you set a time step increase factor to some value greater than 1, the time step will grow (from the initial value) at each time step until it reaches the value of the element-based critical time step estimate. From then on, the element-based critical time step estimate will essentially control the time step.

With the second technique, you can manipulate the element-based estimate with either a scale factor or a time step increase factor. This technique is employed by using the `TIME STEP SCALE FACTOR` command line or the `TIME STEP INCREASE FACTOR` command line as follows:

- The element-based estimate for the critical time step is usually smaller than some maximum theoretical value for your model. It may therefore be possible to scale the element-based critical time step estimate by some factor greater than 1. (Your scaled value must remain below the theoretical maximum limit, however. We discuss ways to obtain a critical time step close to the theoretical maximum in later sections of this chapter.)
- If there are stability problems with a particular problem, it may be necessary to scale the element-based estimate with a factor less than 1.
- You can also control the rate at which the time step can increase for an analysis. By specifying a time step increase factor, you can limit the increase in the size of the time step so that it does not increase too rapidly from one step to the next. For certain problems, the element-based critical time step estimate may increase rapidly from one step to the next. Limiting the increase in the size of the time step may enable some problems to run in a more stable fashion.

The `USER TIME STEP` command can be used to explicitly set the integration time step, overriding the critical time step computed by any time step estimation methods. This command can be used to force the analysis to integrate at any specified time step, even a super-critical one.

Warning: The TIME STEP SCALE FACTOR and USER TIME STEP commands both provide the ability to force the code to take a time step larger than the stability limit, and as such, should be used with caution. If the time step specified with either of these commands is too large, the time integration will add energy to the system, causing the analysis to go unstable and ultimately fail.

Now that we have presented an overview of the functionality in the TIME CONTROL command block, we will discuss the actual command lines.

3.1.1 Command Blocks for Time Control and Time Stepping

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK <string>time_block_name
    START TIME = <real>start_time_value
    BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
      #
      # Time control parameters specific to PRESTO
      # are set in this command block.
      #
    END [PARAMETERS FOR PRESTO REGION <string>region_name]
  END [TIME STEPPING BLOCK <string>time_block_name]
  TERMINATION TIME = <real>termination_time
END [TIME CONTROL]
```

Time control resides in a TIME CONTROL command block. The command block begins with an input line of the form

```
BEGIN TIME CONTROL
```

and terminates with an input line of the following form:

```
END [TIME CONTROL]
```

Within the TIME CONTROL command block, a number of TIME STEPPING BLOCK command blocks can be defined. Each TIME STEPPING BLOCK command block contains the time at which the time stepping starts and a number of parameters that set time-related values for the analysis. Each TIME STEPPING BLOCK command block terminates at the start time of the following command block. The start times for the TIME STEPPING BLOCK command blocks must be in increasing order. Otherwise, an error will be generated. (The example in Section 3.1.6 shows the overall structure of the TIME CONTROL command block.)

In the above input lines, the values are as follows:

- The string time_block_name is a name for the TIME STEPPING BLOCK command block. This name must be unique to each command block of this type. The string time_block_name can be referenced in an ACTIVE PERIODS or INACTIVE PERIODS command line to activate and deactivate functionality (see Section 2.5).

- The real value `start_time_value` is the start time for this `TIME STEPPING BLOCK` command block. Values set in this block apply from the start time for this block until the next start time or the termination time. The start time may be negative.

- The string `region_name` is the name of the Presto region affected by the parameters (see Section 2.2).

The termination time for the analysis is given by the following command line:

```
TERMINATION TIME = <real>termination_time
```

Here, `termination_time` is the time at which the analysis will stop. The `TERMINATION TIME` command line appears inside the `TIME CONTROL` command block but outside of any `TIME STEPPING BLOCK` command block.

The `TERMINATION TIME` command line can appear before the first `TIME STEPPING BLOCK` command block or after the last `TIME STEPPING BLOCK` command block. Note that it is permissible to have `TIME STEPPING BLOCK` command blocks with start times after the termination time; in this case, those command blocks that have start times after the termination time are not executed. Only one `TERMINATION TIME` command line can appear in the `TIME CONTROL` command block. If more than one of these command lines appears, an error is generated.

Nested inside the `TIME STEPPING BLOCK` command block is a `PARAMETERS FOR PRESTO REGION` command block containing parameters that control the time stepping.

```
BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
  INITIAL TIME STEP = <real>initial_time_step_value
  TIME STEP SCALE FACTOR = <real>time_step_scale_factor(1.0)
  TIME STEP INCREASE FACTOR =
    <real>time_step_increase_factor(1.1)
  STEP INTERVAL = <integer>nsteps(100)
END [PARAMETERS FOR PRESTO REGION <string>region_name]
```

These parameters are specific to a Presto analysis.

The command block begins with an input line of the form

```
BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR PRESTO REGION <string>region_name]
```

As noted previously, the string `region_name` is the name of the Presto region affected by the parameters. The command lines nested inside the `PARAMETERS FOR PRESTO REGION` command block are described next. It should be noted that certain of these command lines will be ignored when either of the maximum-eigenvalue methods or the node-based method is used to estimate the critical time step. The discussions of the command blocks associated with these methods indicate whether or not these command lines are pertinent.

3.1.2 Initial Time Step

```
INITIAL TIME STEP = <real>initial_time_step_value
```

Presto computes a critical time step for the analysis and uses this value as the initial time step by default. To directly specify a smaller initial time step, use the INITIAL TIME STEP command line, where initial_time_step_value is the desired initial time step. This command line is only valid if it is in the first TIME STEPPING BLOCK command block in the problem.

The value specified for the initial time step will overwrite the calculated value for the critical time step if the specified initial time step is smaller than the critical time step. If you specify an initial time step that is larger than the critical time step, the time step is set to the value of the calculated critical time step.

3.1.3 Time Step Scale Factor

```
TIME STEP SCALE FACTOR = <real>time_step_scale_factor(1.0)
```

During the element computations, Presto computes a minimum time step required for stability of the computation (the critical time step). Using the TIME STEP SCALE FACTOR command line, you can provide a scale factor to modify the critical time step. Note that a value greater than 1.0 for time_step_scale_factor will cause the time step to be greater than the computed critical time step, and thus the problem may become unstable. By default, the scale factor is 1.0.

3.1.4 Time Step Increase Factor

```
TIME STEP INCREASE FACTOR =
  <real>time_step_increase_factor(1.1)
```

During an analysis, the computed critical time step may change as elements deform, are killed, and so forth. By using the TIME STEP INCREASE FACTOR command line, you can limit the amount that the time step can increase between two adjacent time steps. The value time_step_increase_factor is a factor that multiplies the previous time step. The current time step can be no larger than the product of the previous time step and the scale factor.

Note that an increase factor less than 1.0 will cause the time step to continuously decrease. The default value for this factor is 1.1, i.e., a time step cannot be more than 1.1 times the previous step.

3.1.5 Step Interval

```
STEP INTERVAL = <integer>nsteps(100)
```

Presto can output data about the current time step, the current internal and external energy, and the kinetic energy throughout an analysis. The STEP INTERVAL command line controls the frequency

of this output, where `nsteps` is the number of time steps between output. The default value for `nsteps` is 100.

The output at any given step (read from left to right) is

- step number,
- time,
- time step,
- global element identifier for element controlling time step
- kinetic energy,
- internal energy,
- external energy (work done on boundary),
- error in energy balance,
- hour glass energy
- cpu time, and
- wall clock time.

The time is at the current time, step n, and the time increment is the previous time step increment from step $n-1$ to step n.

The error in the energy balance is computed from the following relation:

```
energy balance error = (kinetic energy + internal energy
  - external energy) / external energy * 100
```

The above expression gives a percent error for the energy balance.

3.1.6 Example

The following is a simple example of a `TIME CONTROL` command block:

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK p1
    START TIME = 0.0
    BEGIN PARAMETERS FOR PRESTO REGION presto_region
      INITIAL TIME STEP = 1.0e-6
      STEP INTERVAL = 50
    END
```

```
  END
  BEGIN TIME STEPPING BLOCK p2
    START TIME = 0.5e-3
    BEGIN PARAMETERS FOR PRESTO REGION presto_region
      TIME STEP SCALE FACTOR = 0.9
      TIME STEP INCREASE FACTOR = 1.5
      STEP INTERVAL = 10
    END
  END
  TERMINATION TIME = 1.0e-3
END
```

The first `TIME STEPPING BLOCK`, `p1`, begins at time 0.0, the initial start time, and terminates at time 0.5×10^{-3}. The second `TIME STEPPING BLOCK`, `p2`, begins at time 0.5×10^{-3} and terminates at time 1.0×10^{-3}, which is the time listed on the `TERMINATION TIME` command line. The `TIME STEPPING BLOCK` names `p1` and `p2` can be referenced by the `ACTIVE PERIODS` or `INACTIVE PERIODS` command lines as described in Section 2.5 to activate and deactivate functionality.

3.2 Other Critical Time Step Methods

Currently, there are four methods for calculating a critical time step for Presto. First, there is the traditional element-based method. We know that, in general, the element-based time step in Presto can give a fairly conservative estimate for the time step. Second, there is a node-based method for giving a critical time step estimate. Depending on the problem, the node-based method may or may not give an estimate for the critical time step that approaches the theoretical maximum value for a particular model. Although the node-based method can give a larger critical time step estimate than the element-based method, the node-based estimate may still be significantly lower than the maximum theoretical time step for a problem. Finally, there are two methods that use an estimate for the maximum eigenvalue to obtain an estimate for the critical time step. From the maximum eigenvalue, it is possible to derive the theoretical maximum critical time step for a problem via the formula

$$\Delta t_{crit} = 2.0/\sqrt{\theta_{max}}, \tag{3.1}$$

where Δt_{crit} is the critical time step and θ_{max} is the maximum eigenvalue. The two methods employed in Presto to give an estimate for the maximum eigenvalue are the Lanczos method and the power method. The Lanczos method can give an accurate estimate of the maximum eigenvalue for a problem using a very small number of Lanczos vectors compared to the total number of degrees of freedom in a problem. The power method uses a simple iterative process to obtain an estimate for the maximum eigenvalue. The power method is not as powerful a mathematical technique as the Lanczos method for obtaining a maximum eigenvalue estimate. However, it does present another viable option for obtaining a maximum eigenvalue estimate for certain problems and has therefore been included as another method for obtaining the maximum eigenvalue estimate.

The use of a critical time step from the node-based method or a method based on the maximum eigenvalue estimate is desirable because the larger critical time steps produced by these methods (compared to the element-based method) reduce CPU time. Both methods, however, are not cost effective if they are called every time step to give a critical time step estimate. The cost of doing one node-based estimate or one maximum eigenvalue estimate for the critical time step will not offset the cost benefit of the increase to the critical time step (compared to the element-based time step estimate) over a single time step. Hence, there must be some scheme for

- calling these methods only periodically throughout a calculation and
- maintaining a larger estimate (than the element-based estimate) for the critical time step in between these calls

if we are to gain a net benefit from the increase in the critical time step these methods can produce.

If you want to use the maximum eigenvalue estimate for calculating the critical time step and your problem has long periods with a relatively stable time step estimate, the Lanczos method for calculating the maximum eigenvalue will be the preferred method to use. If you have a problem for which the changes in the time step should be monitored more frequently, the power method will be better suited for this problem than the Lanczos method. The preferred use for each of these

two different eigenvalue methods will become obvious as you read the background material for these two methods in Section 3.2.1 and Section 3.2.2. Both the Lanczos method and the power method require the use of a variety of control parameters. For some of these control parameters, techniques have been devised to automatically calculate values for these parameters in what should be a robust and reliable method for a wide range of problems. Other parameters are defaulted to values that will optimize the use of a particular eigenvalue calculation method (Lanczos method or power method) with the type of problems that best suit the method that is employed. The default values for these control parameters may change as we gain more experience in using the Lanczos method and the power method.

Detailed discussions of the Lanczos method, the power method, and the node-based method follow. There are many similarities in the implementation of the Lanczos method and the power method.

3.2.1 Lanczos Method

The Lanczos method, as implemented in Presto, is outlined here so that certain aspects of it can be referenced in subsequent parts of this chapter. In the following set of equations, $\mathbf{K}_T$ is a tangent stiffness matrix, $\mathbf{M}$ is the mass matrix, and $\mathbf{r}_0$ is an arbitrary starting vector.

Initialize

$$\mathbf{q}_0 = \mathbf{0}$$
$$\beta = ([\mathbf{r}_0]^T \mathbf{M}^{-1} \mathbf{r}_0)^{1/2}$$
$$\mathbf{q}_1 = \mathbf{r}_0 / \beta_1$$
$$\mathbf{p}_1 = \mathbf{M}^{-1} \mathbf{q}_1$$

for j = 1, n

$$\bar{\mathbf{r}}_j = \mathbf{K}_T \mathbf{p}_j$$
$$\hat{\mathbf{r}}_j = \bar{\mathbf{r}}_j - \mathbf{q}_{j-1} \beta_j$$
$$\alpha_j = [\mathbf{q}_j]^T \mathbf{M}^{-1} \hat{\mathbf{r}}_j = [\mathbf{p}_j]^T \hat{\mathbf{r}}_j$$
$$\mathbf{r}_j = \hat{\mathbf{r}}_j - q_j \alpha_j$$
$$\hat{\mathbf{p}}_j = \mathbf{M}^{-1} \mathbf{r}_j$$
$$\beta_{j+1} = ([\mathbf{r}_j]^T \mathbf{M}^{-1} \mathbf{r}_j)^{1/2} = ([\bar{\mathbf{p}}_j]^T \mathbf{r}_j)^{1/2}$$

if enough vectors, terminate loop

$$\mathbf{q}_{j+1} = \mathbf{r}_j / \beta_{j+1}$$
$$\mathbf{p}_{j+1} = \bar{\mathbf{p}}_j / \beta_{j+1}$$

end

The details for this form of the Lanczos method are described in Reference 1. Notice that the Lanczos method is an iterative method. If we use the Lanczos method in a code like Presto to compute the maximum eigenvalue for a particular finite element model, the number of iterations

required to give a good estimate for the maximum eigenvalue will depend on the size of the finite element model (the number of nodes and elements), the types of elements in the model, and the material types and material properties used in the model.

The maximum eigenvalue for a particular finite element model gives us the largest estimate we can obtain for the critical time step. Hence, the time step estimate derived from the maximum eigenvalue is our "best" estimate for the critical time step. The estimate for the critical time step based on the maximum eigenvalue can be significantly larger than an element-based time step estimate. However, computing the critical time step (for a given time step) with the Lanczos method is more expensive than computing the critical time step with element-based calculations. Over one time step, it is not possible to recoup the cost of the Lanczos calculations with the increase in the size of the time step over the element-based estimate. Using the Lanczos method for estimating the critical time step in an explicit, transient dynamics code requires a methodology that effectively addresses the computational costs. The following sections outline a cost-effective approach to using the Lanczos method in an explicit, transient dynamics code.

3.2.1.1 Lanczos Method with Constant Time Steps

To explain how to use the Lanczos method in a cost-efficient manner, we must first establish the computational cost of using the Lanczos method. As indicated previously, computing the maximum eigenvalue for a finite element model requires some number of iterations (each iteration in the Lanczos method produces a Lanczos vector) to obtain a good estimate for the maximum eigenvalue. The cost of an iteration (Lanczos vector) is approximately the cost of an internal force calculation. Notice, in preceding equations for the Lanczos method, that the Lanczos method requires the product of the tangent stiffness matrix $\mathbf{K}_T$ with a vector $\mathbf{p}$. In Presto, we do not construct a tangent stiffness matrix. Instead, we simply provide the vector $\mathbf{p}$ for the internal force calculations. The internal force calculations give us the desired matrix $\times$ vector product of $\mathbf{K}_T\mathbf{p}$.

Over a given time step, the cost of an internal force calculation is the major computational cost. (This assumes no contact. The addition of contact introduces another computationally expensive process into a time step. For our initial discussion, we ignore the cost of contact.) In order to use the Lanczos method to get a critical time step estimate, one has to call the Lanczos method for some given time step. If the Lanczos method made n iterations to get a good estimate for the maximum eigenvalue, then the overall cost of the time step would be approximately $n + 1$ times the cost of the internal force calculation. The cost of the time step would be the n internal force calculations for Lanczos and the actual internal force calculation to advance the time step. If n is 20 (a minimum for typical problems), the cost of the time step becomes 21 times the cost of the internal force calculation. The critical time step estimate based on the maximum eigenvalue would have to be at least 21 times greater than the element-based critical time step to recoup the cost of the maximum eigenvalue calculation. A typical value for the critical time step estimate based on the maximum value is more in the range of 1.1 to 2.0 times the element-based critical time step estimate. Obviously, the Lanczos method is much too expensive to call at every time step for a critical time step estimate.

To explain how the Lanczos method can effectively be used in an explicit, transient dynamics code, we begin with a simple case study. In this case study, we compute the critical time step using

the Lanczos method at some time step and then assume that this critical time step value remains constant for a subsequent number of time steps, n_L. We only call the Lanczos method once during the n_L time steps. (In reality, the critical time step in an explicit, transient dynamics code like Presto changes with each time step. We address this issue of the changing time step when we present the details for using the Lanczos method in a cost-effective manner.) For this case study, we also assume that the computational cost of an element-based estimate for the critical time step is part of the cost of an internal force calculation. The cost of the element-based time step estimate is a small part of the overall internal force calculations. Finally, for our initial discussion, we assume no contact. We address the issue of contact further in the discussion.

Assume that the Lanczos method computes a global estimate for the critical time step of Δt_L, which is the value to be used for n_L time steps. At the end of the n_L time steps, the analysis time for the code has been incremented by an amount ΔT, which is computed simply as

$$\Delta T = n_L \Delta t_L \,. \tag{3.2}$$

If the element-based estimate for the time step is Δt_e and the number of time steps required to increment the analysis time by ΔT is n_e, then, for the element-based time step, we have

$$\Delta T = n_e \Delta t_e \,. \tag{3.3}$$

Because the Lanczos estimate for the critical time step is larger than the element-based estimate, we know that $n_e > n_L$. Let us define the ratio r as

$$r = \Delta t_L / \Delta t_e = n_e / n_L \,. \tag{3.4}$$

The ratio r is greater than 1.

Now that we have determined the relation between the number of steps required for a Lanczos-based critical time step estimate versus the element-based critical time step estimate to achieve the same analysis time increment, let us examine the computational costs for these two cases in terms of CPU time. Designate the CPU cost for a time step as Δt_{IF}. If the number of Lanczos vectors required to obtain the critical time step estimate is N_L, then the total computational cost of the Lanczos method and the n_L time steps is

$$n_L \Delta t_{IF} + N_L \Delta t_{IF} \,. \tag{3.5}$$

If we use the element-based method, the total computational cost is

$$n_e \Delta t_{IF} \,. \tag{3.6}$$

Recall that we have chosen n_L and n_e so that we have the same analysis time increment ΔT even though we have different critical time steps. Now, we must determine the point at which the computational cost for the Lanczos-based critical time step calculations is the same as the cost for the element-based critical time step calculations. This is simply the point at which

$$n_e \Delta t_{IF} = n_L \Delta t_{IF} + N_L \Delta t_{IF} \,. \tag{3.7}$$

If we rearrange the above equation to eliminate Δt_{IF} and make use of the ratio r, then we obtain

$$n_e = \frac{N_L}{1 - 1/r} \,. \tag{3.8}$$

Consider the case of $r = 1.25$ and $N_L = 20$. When $n_L = 80$ and $n_e = 100$, the above equations show that the calculations with the Lanczos-based critical time step and the calculations with the element-based time step give the same analysis time for the same computational expense. If we use the Lanczos-based critical time step Δt_L for more than eighty iterations, then the Lanczos-based approach becomes cost effective. Our above equations have established the "break-even" point at which it becomes cost effective to use the Lanczos method to reduce computational costs by overcoming the initial cost of the Lanczos calculations with the larger critical time step.

We can build on what we have done thus far to account for contact. Suppose that the computational cost of contact over a time step is some multiple m of the computational cost of the internal force calculation Δt_{IF}. Then the point at which the computational cost for the Lanczos-based calculations is the same as the computational cost for the element-based calculations is

$$(1 + m) n_L \Delta t_{IF} + N_L \Delta t_{IF} = (1 + m) n_e \Delta t_{IF} \,. \tag{3.9}$$

For the case with contact,

$$n_e = \frac{N_L}{(1 + m)(1 - 1/r)} \,. \tag{3.10}$$

Again, consider the case of $r = 1.25$ and $N_L = 20$. Assume the computational cost of contact calculations is the same as an internal force calculation ($m = 1$). For these values, the break-even point is $n_L = 40$ and $n_e = 50$. The added computational cost of the contact calculations results in reaching the break-even point with a smaller number of iterations when compared to the case with no contact.

The above derivations let us calculate a break-even point based on our assumptions of a constant critical time step. Considering that a typical analysis will run for tens of thousands of time steps, something on the order of 100 steps represents a reasonable number of steps to recoup the cost of the Lanczos calculations. Whether or not the cost of the Lanczos calculations can be recouped in something on the order of 100 calculations depends heavily upon N_L. If N_L is sufficiently small, we can recoup the cost of the Lanczos calculations in a reasonable number of steps.

Some computational studies indicate that N_L is in an acceptable range for many problems. The Lanczos method computes a good estimate for the maximum eigenvalue with a small number of Lanczos vectors, N_L, compared to the number of degrees of freedom in a problem. Some component studies show that, for a problem with between 250,000 and 350,000 degrees of freedom, one can obtain a good estimate for the maximum eigenvalue with only twenty Lanczos vectors. A large-scale study of a model involving 1.7 million nodes (5.1 million degrees of freedom) showed

that only forty-five Lanczos vectors were required to obtain a good estimate of the maximum eigenvalue. These examples demonstrate that the number of Lanczos vectors required for a good maximum eigenvalue estimate is very small when compared to the number of degrees of freedom for a problem. When N_L is in the range of twenty to forty-five, Equations 3.7 and 3.10 show that, with an increase in the time step on the order of 1.2 to 1.25, we can recoup the cost of the Lanczos method in a reasonable number of time steps.

Now that we have determined we can recoup the cost of the Lanczos calculations in a reasonable number of time steps, let us look at the issue of reusing a Lanczos-based estimate in some manner.

3.2.1.2 Controls for Lanczos Method

As indicated in the above discussion, the Lanczos method can be used in a cost-effective manner in an explicit, transient dynamics code if a Lanczos calculation can be performed and the Lanczos-based estimate for the critical time step can be reused in some way over a number of subsequent time steps. This section presents an approach for reusing a Lanczos-based estimate over a number of time steps so that we maintain a critical time step estimate that is close to the theoretical maximum value in between the calls to the Lanczos method. The approach discussed here makes use of the element-based critical time step estimate at each time step.

We start our approach with a Lanczos calculation to determine the maximum eigenvalue. The Lanczos method converges to the maximum eigenvalue from below, which means that the method underestimates the maximum eigenvalue. Because the critical time step depends on the inverse of the maximum eigenvalue, we overestimate the critical time step. It is necessary, therefore, to scale back the critical time step estimate from the Lanczos method so that the calculations in the explicit time-stepping scheme do not become unstable. Our approach for determining a scaled-back value for the maximum critical time step makes use of the element-based time step estimate. Again, let Δt_L be the critical time step estimate from the Lanczos method and Δt_e be the critical time step estimate from the element-based calculations. The scaled-back estimate for the critical time step, Δt_s, is computed from the equation

$$\Delta t_s = \Delta t_e + f_s(\Delta t_L - \Delta t_e)\,, \tag{3.11}$$

where f_s is a scale factor. (A reasonable value for f_s ranges from 0.9 to 0.95 for our problems.) This value of f_s puts Δt_s close to and slightly less than a theoretical maximum critical time step.

Once Δt_s is determined, the ratio

$$t_r = \Delta t_s / \Delta t_e \tag{3.12}$$

is computed. This ratio is then used to scale subsequent element-based estimates for the critical time step. If $\Delta t_{e(n)}$ is the nth element-based critical time step after the time step where the Lanczos calculations are performed, then the nth time step after the Lanczos calculations, $\Delta t_{(n)}$, is simply

$$\Delta t_{(n)} = t_r \Delta t_{e(n)}\,. \tag{3.13}$$

The ratio t_r is used until the next call to the Lanczos method. The next call to the Lanczos method is controlled by one of two mechanisms. With the first mechanism, the user can set the frequency with which the Lanczos method is called. The user can set a parameter so that the Lanczos method is called only once every n time steps. This number remains fixed throughout an analysis. With the second mechanism, the user can control when the Lanczos method is called based on changes in the element-based critical time step. For this second mechanism, the change in the element-based critical time step estimate is tracked. Suppose the element-based critical time step at the time the Lanczos method was called is Δt_e. At the nth step after the call to the Lanczos method, the element-based critical time step is $\Delta t_{e(n)}$. If the value

$$\Delta t_{lim} = \frac{|\Delta t_{e(n)} - \Delta t_e|}{\Delta t_e} \tag{3.14}$$

is greater than some limit set by the user, then the Lanczos method will be called. If there is a small, monotonic change in the element-based critical time step over a large number of time steps, this second mechanism will result in the Lanczos method being called. Or if there is a large, monotonic change in the element-based critical time step over a few time steps, the Lanczos method will also be called.

These two mechanisms for calling the Lanczos method can be used together. For example, suppose the second mechanism (the mechanism based on a change in the element-based time step) results in a call to the Lanczos method. This resets the counter for the first mechanism (the mechanism using a set number of time steps between calls to the Lanczos method).

This approach for reusing a Lanczos-based time step estimate has been implemented in Presto, and it has been used for a number of studies. One of the component studies, as indicated previously, used the same scale factor for $n_L = 1700$ iterations. The break-even point for this problem is $n_e = 45$ time steps (not accounting for contact, which was a part of the component modeling). For this particular problem, the extended use of the Lanczos estimate reduced the computational cost to 56% of what it would have been with the element-based time step.

Not all problems will lend themselves to reusing one Lanczos-based estimate for thousands of time steps. However, if it is possible to use the Lanczos-based estimate for two to three times the number of time steps required to reach the break-even point, we begin to see a noticeable reduction in the total number of time steps required for a problem.

3.2.1.3 Scale Factor for Lanczos Method

When the Lanczos method is called for a given time step, it must appear that the calculations are using the constant tangent stiffness matrix $\mathbf{K}_T$ for all iterations. As indicated previously, we use the internal force calculations to generate the product $\mathbf{K}_T\mathbf{p}_j$ (for the jth iteration) in the Lanczos calculations. Any vector $\mathbf{p}_j$, as calculated by the Lanczos method, may be such that it represents large-strain behavior and moves the internal force calculations into a nonlinear regime. It is necessary to scale the $\mathbf{p}_j$ vectors so that the internal force calculations are in a small-strain regime, which makes it appear that we are working with a constant tangent stiffness matrix. The vectors $\mathbf{p}_j$ must be scaled so that they represent velocities associated with small strain. When properly scaled

vectors are sent to the internal force calculation, the internal force calculation effectively becomes a matrix × vector product with a constant tangent stiffness matrix for all iterations during a given call to the Lanczos method.

The scale factor for the $\mathbf{p}_j$ vectors, which we will designate as v_{sf}, must not be too small, as this will create round-off problems and give a bad estimate for the critical time step. If the scale factor is too large, we violate the above restriction of a constant tangent stiffness matrix.

There are two approaches for controlling the scale factor when the Lanczos method is used to compute the maximum eigenvalue. These approaches are discussed in Section 3.2.1.5.

3.2.1.4 Accuracy of Eigenvalue Estimate

Every time a new Lanczos vector is computed, we obtain an additional eigenvalue for our model and, in general, a better estimate for the maximum eigenvalue (and hence a better estimate for the critical time step). The Lanczos method can compute a good value for the maximum eigenvalue with a very small number of total computed eigenvalues compared to the number of degrees of freedom in a problem. There are examples, as previously indicated, of problems with 250,000 to 350,000 degrees of freedom where we have obtained a good estimate of the critical time step with twenty eigenvalues. In one problem with 5.1 million degrees of freedom, we obtained a good estimate of the critical time step with forty-five eigenvalues.

A user could, in theory, determine a reasonable number of eigenvalues necessary for obtaining a good estimate of the maximum eigenvalue based on the above information on model size and the number of eigenvalues required for a good maximum eigenvalue estimate. The user could test the validity of the choice of the number of eigenvalues by increasing the number of eigenvalues slightly and comparing the maximum eigenvalue estimate obtained with the larger number of eigenvalues to the original maximum eigenvalue estimate obtained with the smaller number of eigenvalues. If the change in the two maximum eigenvalue estimates (larger versus smaller total number of eigenvalues) is small, then the original estimate for the number of eigenvalues is reasonably accurate.

As an alternative to directly specifying the number of eigenvalues to be computed, a convergence tolerance could be set on the change in the magnitude of the maximum eigenvalue as additional eigenvalues (Lanczos vectors) are computed. Let θ_{max_n} be the maximum eigenvalue calculated corresponding to n eigenvalues (Lanczos vectors), and let θ_{max_n+1} be the maximum eigenvalue corresponding to $n + 1$ eigenvalues (Lanczos vectors). The eigenvalues would be computed until

$$\frac{\mid \theta_{max_n+1} - \theta_{max_n} \mid}{\theta_{max_n+1}} \tag{3.15}$$

is less than some tolerance. (We will now refer to the value of Equation (3.15) as the *convergence measure*.) If we calculate the convergence measure for a sequence of maximum eigenvalues computed by the Lanczos method, we will not necessarily see a monotonic decrease in the convergence measure for all time. Typically, the convergence measure will initially show a monotonically decreasing value for an increasing n, and then the convergence measure will show a monotonically increasing value for an increasing n. (The phase in which the convergence measure increases is usually very small, on the order of one to two iterations.) After the phase in which the convergence

measure monotonically increases, there is then a phase in which the convergence measure begins a long (over many iterations) monotonic decrease. This behavior (decrease of the convergence measure, slight increase, then long monotonic decrease) is typical of Krylov methods, of which the Lanczos method is one. Using the convergence measure to determine the number of eigenvalues to be computed for a problem is a reasonable option, but some care must be taken in setting the tolerance for the convergence measure.

Both of the techniques just discussed are offered as a way to set the maximum number of eigenvalues required to obtain an accurate eigenvalue estimate for a model. These techniques are discussed further in Section 3.2.1.5.

3.2.1.5 Lanczos Parameters Command Block

```
BEGIN LANCZOS PARAMETERS <string>lanczos_name
  STARTING VECTOR = <string>STRETCH_X|STRETCH_Y|STRETCH_Z|
    ISOTHERMAL(ISOTHERMAL)
  INCREASE OVER STEPS = <integer>incr_int(5)
  NUMBER EIGENVALUES = <integer>num_eig(150)
  EIGENVALUE CONVERGENCE TOLERANCE = <real>converge_tol(0.5e-3)
  SMALL STRAIN = <real>small_strain(1.0e-3)
  VECTOR SCALE = <real>vec_scale(1.0e-5)
  SCALE FACTOR = <real>time_scale(0.9)
  UPDATE ON TIME STEP CHANGE = <real>tstep_change(0.10)
  UPDATE STEP INTERVAL = <integer>step_int(500)
END [LANCZOS PARAMETERS <string>lanczos_name]
```

If you use the Lanczos method to compute a critical time step, there should be only one `LANCZOS PARAMETERS` command block, and it should appear in the region. If you have a `LANCZOS PARAMETERS` command block, you should not specify a `NODE BASED TIME STEP PARAMETERS` or a `POWER METHODS PARAMETERS` command block. If you use the Lanczos method to compute the critical time step, the time step increase factor (default or user-specified) will be used to control the increase in the time step estimate; the time step scale factor (default or user-specified) will not be used. These factors are specified in the `PARAMETERS FOR PRESTO REGION` portion of a `TIME STEPPING BLOCK` in the `TIME CONTROL` command block.

The Lanczos method requires some type of starting vector. This is determined by the `STARTING VECTOR` command line. The various options available for this command line will generate a displacement vector that stretches your model in the x-, y-, or z-direction, or in all three directions (x, y, and z) at once. (`STRETCH_X` stretches the model in the x-direction, `STRETCH_Y` stretches the model in the y-direction, `STRETCH_Z` stretches the model in the z-direction, and `ISOTHERMAL` stretches the model in all three directions at once.) The displacement vector then serves as a basis for generating the starting vector r_0 in the initialization phase of the Lanczos method. The Lanczos method appears to be fairly insensitive to the choice of a starting vector. The default starting vector is the `ISOTHERMAL` option. You may encounter cases where the use of one of the other options for a starting vector—`STRETCH_X`, `STRETCH_Y`, or `STRETCH_Z`—may result in a slight increase in accuracy for the critical time step estimate for a given number of eigenvalues. These other options

(STRETCH_X, STRETCH_Y, and STRETCH_Z) are offered for these special cases.

The INCREASE OVER STEPS command line determines how many steps, i.e., incr_int, are used to transition from an element-based critical time step estimate to the Lanczos-based estimate at the beginning of an analysis. The user may want to increase from the element-based estimate to the Lanczos-based estimate over a number of time steps if the difference between these two estimates is large. The value of incr_int for this command line defaults to 5.

As indicated in Section 3.2.1.4, some number of eigenvalues must be computed by the Lanczos method to obtain a good estimate for the maximum eigenvalue for a given finite element model. As one option, the user can simply specify the number of eigenvalues to be computed by the Lanczos method by using the NUMBER EIGENVALUES command line. The default value for num_eig in this command line is 150. As an alternative, the user can specify a tolerance on the convergence measure defined in Equation (3.15). The tolerance is set with the EIGENVALUE CONVERGENCE TOLERANCE command line. The default value for the convergence tolerance, converge_tol, is 0.5×10^{-3}. (This default value may change in the future as we gain more experience with the Lanczos method.)

By default, the convergence measure (with the default value of 0.5×10^{-3}) is used to determine the number of eigenvalues to be computed. If neither the EIGENVALUE CONVERGENCE TOLERANCE command line nor the NUMBER EIGENVALUES command line appears in the LANCZOS PARAMETERS command block, then the convergence measure will be used to determine the number of eigenvalues. If the EIGENVALUE CONVERGENCE TOLERANCE command line appears in the command block with a specified tolerance other than the default value, then the number of eigenvalues computed will be determined by the convergence measure and the user-specified tolerance. If the NUMBER EIGENVALUES command line appears in the LANCZOS PARAMETERS command block, then the user-defined number of eigenvalues will be used to set the number of eigenvalues computed by the Lanczos method.

The recommended technique for determining the number of eigenvalues is to use the convergence measure option with the default tolerance. If you use this technique, neither the NUMBER EIGENVALUES command line nor the EIGENVALUE CONVERGENCE TOLERANCE command line should be included in the LANCZOS PARAMETERS command block.

As indicated in Section 3.2.1.3, it is necessary to scale one of the vectors (with a scale factor v_{sf}) in the Lanczos calculations so that it appears we are always working with a constant tangent stiffness each time we call the Lanczos method. There are two approaches for determining the scale factor. With the first approach, the user can specify the value of a small strain for a particular model by using the SMALL STRAIN command line. (As long as the internal force calculation inside the Lanczos method is working with a vector that approximates small-strain behavior, it will appear that we are working with a constant tangent stiffness inside the Lanczos method.) The value for v_{sf} is then computed based on your specification for what constitutes a small strain in the model. (The default value for small_strain is 1.0×10^{-3}.) With the second approach for determining the scale factor, the user can specify a value for v_{sf} directly by using the VECTOR SCALE command line. The default value for v_{sf}, the vec_scale parameter in that command line, is 1.0×10^{-5}. A number of tests have established that this value for the scale factor, 1.0×10^{-5}, works well for a range of models encountered at Sandia National Laboratories. If the user wants to use the option of directly specifying the scale factor, it is possible to determine whether a particular scale factor

is suitable for a particular problem. Take a scale factor v_{sf}, plus values on either side of it, say, $0.9 \times v_{sf}$ and $1.1 \times v_{sf}$. If all three of these scale factor values produce almost the same estimate for the critical time step for a particular model, then the value for v_{sf} meets the criterion for an acceptable scale factor.

By default, the small-strain approach (with the default value of 1.0×10^{-3}) is used to determine a scale factor. If neither the `SMALL STRAIN` command line nor the `VECTOR SCALE` command line appears in the `LANCZOS PARAMETERS` command block, then the small-strain value will be used to calculate the scale factor. If the `SMALL STRAIN` command line appears in the command block with any value other than the default value *and* the `VECTOR SCALE` command line does not appear in the command block, then the vector scale will be determined by the user-specified value for the small strain. If the `VECTOR SCALE` command line does appear in the `LANCZOS PARAMETERS` command block, then the vector scale will be determined by the user-specified value for the vector scale.

The recommended way of determining the scale factor is to use the small-strain approach with the default small-strain value. If you follow this advice, neither the `SMALL STRAIN` command line nor the `VECTOR SCALE` command line should be included in the `LANCZOS PARAMETERS` command block.

As indicated previously, the scale factor must not be too small, as this will create round-off problems and give a bad estimate for the critical time step. If the scale factor is too large, we violate the above restriction of a constant tangent stiffness matrix.

The `SCALE FACTOR` command line sets the factor f_s in Equation (3.11). The value for f_s, i.e., `time_scale`, is set to 0.9. More experience with the Lanczos method will help us determine whether a slightly larger value of f_s can be used as a default.

The `UPDATE ON TIME STEP CHANGE` command line sets the value for Δt_{lim} in Equation (3.14). If the change in the element-based critical time step estimate as given by Equation (3.14) exceeds the value for Δt_{lim}, then the Lanczos method is called for a new estimate for the critical time step. The default value for Δt_{lim}, i.e., `tstep_change`, is 0.10.

The `UPDATE STEP INTERVAL` command line sets the number of step intervals at which the Lanczos method is called. If `step_int` is set to 1000, the Lanczos method will be called every one thousand steps to compute an estimate for the critical time step. (The default value of `step_int` is 500.) This control mechanism interacts with the control established by the `UPDATE ON TIME STEP CHANGE` command line. Suppose we have set `step_int` to 1000, and we have computed 800 steps since the last call to the Lanczos method. If Δt_{lim} has been set to 0.15 and we exceed this value at step 800, then the change in the element-based time step will result in the Lanczos method being called. The counter for keeping track of the number of step intervals since the last Lanczos computation will be reset to zero. The next call to the Lanczos method will occur again in one thousand steps, unless we again exceed the change in the element-based time step.

3.2.2 Power Method

The power method is a simple iterative technique that gives an estimate for the maximum eigenvalue. Consider the following form of the eigenvalue problem, which is useful for our explicit transient dynamics problem.

$$\mathbf{K}_T \mathbf{x}_i - \theta_i \mathbf{M} \mathbf{x}_i = 0 \tag{3.16}$$

Reformulate the problem as

$$[\mathbf{M}]^{-1} \mathbf{K}_T \mathbf{x}_i - \theta_i \mathbf{x}_i = 0\,. \tag{3.17}$$

Denote the matrix product $\mathbf{M}^{-1}\mathbf{K}_T$ as the matrix operator $\mathbf{A}$. The eigenvalue problem is now

$$\mathbf{A}\mathbf{x}_i - \theta_i \mathbf{x}_i = 0\,. \tag{3.18}$$

The maximum eigenvalue for the problem in Equation (3.18) can be computed with the following iterative process:

$$
\begin{aligned}
&\textit{for } i = 1, n \\
&\qquad \mathbf{x}_i = \mathbf{A}\mathbf{x}_{i-1} \\
&\qquad \mathbf{x}_i = \mathbf{x}_i / \parallel \mathbf{x}_i \parallel \\
&\qquad \theta_i = [\mathbf{x}_i]^T \mathbf{A}\mathbf{x}_i \\
&\textit{end}
\end{aligned}
$$

The value θ_i is the ith estimate for the maximum eigenvalue. As n increases, the power method should yield a more accurate estimate for the maximum eigenvalue. So, just as with the Lanczos method, the power method must iterate at some given time step to obtain an estimate for the maximum eigenvalue at that time step. The details for the above form of the power method are described in Reference 2.

Like the Lanczos method, the power method requires the use of an internal force calculation. The power method, however, is somewhat simpler than the Lanczos method. The power method requires that one vector and one scalar be computed. The Lanczos method requires the computation of five vectors and two scalars. Further, the power method yields a direct estimate for the maximum eigenvalue, whereas the Lanczos method requires that the maximum eigenvalue be extracted from a tridiagonal matrix.

If the sequence of eigenvalues for an analysis problem is

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|, \tag{3.19}$$

then the rate of convergence to the maximum eigenvalue depends on the ratio

$$\frac{|\lambda_1|}{|\lambda_2|}. \tag{3.20}$$

Depending on the magnitudes of $|\lambda_1|$ and $|\lambda_2|$, the rate of convergence could be rather slow. The power method, for a given problem, may not converge to as accurate an estimate for the maximum eigenvalue as quickly as the Lanczos method. We should be careful, however, in how

we interpret this convergence property for the power method. For many of our problems, we probably have a large number of eigenvalues clustered just below the maximum eigenvalue. The power method may give a reasonable estimate for the maximum eigenvalue in a relatively small number of iterations. The power method may converge quickly to some eigenvalue estimate that is just below the maximum eigenvalue. Convergence to the maximum eigenvalue, in terms of accuracy to many digits, might then be very slow. However, one would still have an adequate estimate for the maximum eigenvalue to set a value for the critical time step.

One of the advantages of the power method is that we can use the last value for $\mathbf{x}_i$ from a previous call to the power method as a starting vector for our current call to the power method and converge quickly (for some problems) to an accurate estimate for the maximum eigenvalue. Consider the following scenario. We make an initial call to the power method using some arbitrary $\mathbf{x}_0$ starting vector. We obtain a good estimate for the maximum eigenvalue after n iterations. After some number of time steps, we again call the power method and we use $\mathbf{x}_n$ from the previous call to the power method as our current starting vector. Because we are solving nonlinear problems, the matrix operator, $\mathbf{A}$, has most likely changed from the time when we made the initial call to the power method. If the changes in the matrix operator are small, then $\mathbf{x}_n$ should be an excellent choice for the starting vector, and we should converge quickly to a good estimate for the maximum eigenvalue. This behavior, quick convergence to a good eigenvalue estimate using a prior value of $\mathbf{x}_n$ as a starting vector, has been observed in some sample problems. We discuss this issue of restarting the power method using a previous $\mathbf{x}_n$ in more detail in Section 3.2.2.1.

Obtaining a cost-effective and user-friendly implementation of the power method requires addressing the same issues discussed above for an automated implementation of the Lanczos method. First, we must determine some scheme to call the power method on an intermittent basis so that it is cost effective. The control mechanism for calling the Lanczos method on an intermittent basis can also be used for the power method. Second, the $\mathbf{x}$ vector must be scaled appropriately before it is used in the internal force calculations. (For the Lanczos method, we must scale the vector $\mathbf{p}$.) Finally, we need to terminate the power method. A tolerance on the value in Equation (3.15) can be set to terminate the method. (Setting a tolerance on Equation (3.15) is the same approach used for the Lanczos method.)

3.2.2.1 Power Method with Constant Time Steps

The cost of making one iteration for the power method is approximately the cost of an internal force calculation. As indicated previously, the cost of one iteration for the Lanczos method (the calculation of a Lanczos vector) is also approximately the cost of an internal force calculation. Therefore, the equations developed in Section 3.2.1.1 are also applicable for determining when the power method can be used in a cost-effective manner. Recall that the equation

$$n_e = \frac{N_L}{1 - 1/r} \tag{3.21}$$

in Section 3.2.1.1 gives the number of time steps for the break-even point at which it becomes cost effective for the case of no contact to use the Lanczos method to reduce computational costs by overcoming the initial cost of the Lanczos calculations with the larger critical time step. In the

above equation, N_L is the number of Lanczos iterations, and r is the ratio of the Lanczos-based critical time to the element-based time step estimate. This equation is directly applicable to the power method because the cost of one iteration in the power method is similar to the cost of one iteration in the Lanczos method. (The variable N_L becomes the number of iterations for the power method.)

Now let us return to the scenario in Section 3.2.2 where we look at restarting the power method with the last value for $\mathbf{x}_i$ from a previous call to the power method. We make an initial call to the power method using some arbitrary $\mathbf{x}_0$ starting vector. We obtain a good estimate for the maximum eigenvalue after n iterations. Our estimate for the critical time step based on the maximum eigenvalue gives us a value of 1.25 for r. After some number of time steps, we again call the power method, and we use $\mathbf{x}_n$ from the previous call to the power method as our current starting vector. In our second call to the power method, the power method converges to a good estimate for the maximum eigenvalue in two iterations. The break-even point where we recoup the cost of the power method (in our second call) is ten time steps. If the power method converges to a good estimate for the maximum eigenvalue in three iterations, then the break-even point where we recoup the cost of the power method (in our second call) is fifteen time steps. There are some problems where we can obtain a good estimate for the maximum eigenvalue in two or three iterations when we use the last value for $\mathbf{x}_n$ from a previous call to the power method as our starting vector for the current call to the power method. For the case where we can converge to a good estimate for the maximum eigenvalue in two or three iterations, we can easily call the power method once every forty or fifty time steps and use the power method to improve the time step estimate in a cost-effective manner. Over a time span of forty or fifty time steps, matrix operator $\mathbf{A}$, as defined in Equation (3.18) may change by only a small amount for a large class of problems. If matrix operator $\mathbf{A}$ changes by a small amount, then using the last value of $\mathbf{x}_n$ from a previous call to the power method provides an excellent starting vector and gives us quick convergence. This ability of the power method to converge quickly by using previous information makes it a viable scheme for calculating the maximum eigenvalue and producing a critical time step estimate.

For a problem with contact, the equations derived for the Lanczos method (with contact) in Section 3.2.1.1 are also applicable to the power method. For the power method, as in the case of the Lanczos method, the added computational cost of the contact calculations results in reaching a break-even point with a smaller number of iterations when compared to a case where there is no contact.

3.2.2.2 Controls for Power Method

Using the power method is similar to using the Lanczos method in that we must be able to perform a power method calculation and reuse the power method estimate for the critical time step in some way over a number of subsequent time steps. The approach for reusing a power method estimate for the critical time step over a number of time steps is the same as the approach for reusing a Lanczos-based estimate over a number of time steps as described in Section 3.2.1.2. The approach discussed in Section 3.2.1.2 presents a way to reuse a maximum eigenvalue time step estimate over a number of time steps so that we maintain a critical time step estimate that is close to the theoretical maximum value in between the calls to the maximum eigenvalue calculations. The

approach discussed in in Section 3.2.1.2 makes use of the element-based critical time step estimate at each time step.

3.2.2.3 Scale Factor for Power Method

When the power method is called for a given time step, it must appear that the calculations are using a constant matrix operator, $\mathbf{A}$, for all iterations. The $\mathbf{A}$ operator is the product of the inverse of the mass matrix $\mathbf{M}^{-1}$ and the tangent stiffness matrix $\mathbf{K}_T$ for the problem. The mass matrix is a constant. Therefore, we must assure that we are working with what is effectively a constant tangent stiffness matrix for all iterations during a call to the power method. This is the same requirement that we encounter in the Lanczos method (see Section 3.2.1.3). For the power method, we use the internal force calculations to generate the product $\mathbf{A}\mathbf{x}_i$ by first computing the internal forces using $\mathbf{x}_i$, which gives us the product $\mathbf{K}_T\mathbf{x}_i$, and then multiplying the internal force results by the inverse of the mass matrix. Any vector $\mathbf{x}_i$, as calculated by the power method, may be such that it represents large-strain behavior and moves the internal force calculations into a nonlinear regime. It is necessary to scale the $\mathbf{x}_i$ vectors so that the internal force calculations are in a small-strain regime, which makes it appear that we are working with a constant tangent stiffness matrix. The vectors $\mathbf{x}_i$ must be scaled so that they represent velocities associated with small-strain calculations. When properly scaled vectors are sent to the internal force calculation, the internal force calculation becomes a matrix $\times$ vector product with a constant tangent stiffness matrix for all iterations during a given call to the power method.

The scale factor for the $\mathbf{x}_i$ vectors, which we will designate as v_{sf}, must not be too small, as this will create round-off problems and give a bad estimate for the critical time step. If the scale factor is too large, we violate the above restriction of a constant tangent stiffness matrix.

There are two approaches for controlling the scale factor when the power method is used to compute the maximum eigenvalue. These approaches are discussed in Section 3.2.2.5.

3.2.2.4 Accuracy of Eigenvalue Estimate

To use the power method efficiently, it is best not to specify a set number of iterations for every instance when the power method is called. Instead, some tolerance should be set on the convergence measure defined in Equation (3.15). The initial call to the power method may take a significant number of iterations to produce a good estimate for the maximum eigenvalue. On subsequent calls to the power method, the number of iterations to converge to a good estimate for the maximum eigenvalue should be quite small when we use previous values of $\mathbf{x}$ as a starting vector.

For the power method, we rely on the convergence measure defined by Equation (3.15) to control the number of iterations. The user can set an upper limit on the number of iterations for the power method. Control of the number of iterations for the power method is discussed in more detail in Section 3.2.2.5.

3.2.2.5 Power Method Parameters Command Block

```
BEGIN POWER METHOD PARAMETERS <string>powermethod_name
  STARTING VECTOR = <string>STRETCH_X|STRETCH_Y|STRETCH_Z|
    ISOTHERMAL(ISOTHERMAL)
  INCREASE OVER STEPS = <integer>incr_int(5)
  NUMBER ITERATIONS = <integer>num_iter(150)
  EIGENVALUE CONVERGENCE TOLERANCE =
    <real>converge_tol(0.5e-3)
  SMALL STRAIN = <real>small_strain(1.0e-3)
  VECTOR SCALE = <real>vec_scale(1.0e-5)
  SCALE FACTOR = <real>time_scale(0.9)
  UPDATE ON TIME STEP CHANGE = <real>tstep_change(0.10)
  UPDATE STEP INTERVAL = <integer>step_int(50)
END [POWER METHOD PARAMETERS <string>powermethod_name]
```

If you use the power method to compute a critical time step, there should be only one `POWER METHOD PARAMETERS` command block, and it should appear in the region. If you have a `POWER METHOD PARAMETERS` command block, you should not specify a command block for the node-based method or the Lanczos method. If you use the power method to compute the critical time step, the time step increase factor (default or user-specified) set in the `TIME CONTROL` command block will be used to control the increase in the time step estimate; the time step scale factor (default or user-specified) set in the `TIME CONTROL` command block will not be used. These factors are specified in the `PARAMETERS FOR PRESTO REGION` portion of a `TIME STEPPING BLOCK` in the `TIME CONTROL` command block.

The power method requires some type of starting vector. This is determined by the `STARTING VECTOR` command line. The various options available in this command line will generate a displacement vector that stretches your model in the x-, y-, or z-direction, or in all three directions (x, y, and z) at once. (`STRETCH_X` stretches the model in the x-direction, `STRETCH_Y` stretches the model in the y-direction, `STRETCH_Z` stretches the model in the z-direction, and `ISOTHERMAL` stretches the model in all three directions at once.) The displacement vector then serves as a basis for generating the starting vector x_0 in the initialization phase of the Lanczos method. The default starting vector is the `ISOTHERMAL` option. You may encounter cases where use of one of the other options for a starting vector—`STRETCH_X`, `STRETCH_Y`, or `STRETCH_Z`—may result in a slight increase in accuracy for the critical time step estimate for a given number of iterations. These other options (`STRETCH_X`, `STRETCH_Y`, and `STRETCH_Z`) are offered for these special cases.

The `INCREASE OVER STEPS` command line determines how many steps, `incr_int`, are used to transition from an element-based critical time step estimate to the power method estimate at the beginning of an analysis. The user may want to increase from the element-based estimate to the power method estimate over a number of time steps if the difference between these two estimates is large. The value of `incr_int` defaults to 5.

As indicated in Section 3.2.2.4, some number of iterations are required by the power method to obtain a good estimate for the maximum eigenvalue for a given model. The number of iterations is controlled by the tolerance set for the convergence measure given in Equation (3.15). The tolerance

is set with the EIGENVALUE CONVERGENCE TOLERANCE command line. The default value for the convergence tolerance, converge_tol is 0.5×10^{-3}. (This default value may change in the future as we gain more experience with the power method.)

The user can set an upper limit on the number of iterations for the power method by using the NUMBER ITERATIONS command line. If the number of iterations in the power method exceeds the value set by num_iter, then the power method is terminated. Otherwise, the iterative process in the power method is terminated by the tolerance set on the convergence measure. The default value for num_iter is 150.

As indicated in Section 3.2.2.3, it is necessary to scale the vectors $\mathbf{x}_i$ (with a scale factor v_{sf}) in the power method calculations so that it appears we are always working with a constant tangent stiffness each time we call the power method. There are two approaches for determining the scale factor. With the first approach, the user can specify the value of a small strain for a particular model by using the SMALL STRAIN command line. (As long as the internal force calculation inside the power method is working with a vector that approximates small-strain behavior, it will appear that we are working with a constant tangent stiffness inside the power method.) The value for v_{sf} is then computed based on your specification for what constitutes a small strain in the model. (The default value for small_strain is 1.0×10^{-3}.) With the second approach for determining the scale factor, the user can specify a value for v_{sf} directly by using the VECTOR SCALE command line. The default value for v_{sf}, the vec_scale parameter in that command line, is 1.0×10^{-5}. A number of tests have established that this value for the scale factor, 1.0×10^{-5}, works well for a range of models encountered at Sandia National Laboratories. If the user wants to use the option of directly specifying the scale factor, it is possible to determine whether a particular scale factor is suitable for a particular problem. Take a scale factor v_{sf}, plus values on either side of it, say, $0.9 \times v_{sf}$ and $1.1 \times v_{sf}$. If all three of these scale factor values produce almost the same estimate for the critical time step for a particular model, then the value for v_{sf} meets the criterion for an acceptable scale factor.

By default, the small-strain approach (with the default value of 1.0×10^{-3}) is used to determine a scale factor. If neither the SMALL STRAIN command line nor the VECTOR SCALE command line appears in the POWER METHOD PARAMETERS command block, then the default small-strain value will be used to calculate the scale factor. If the SMALL STRAIN command appears in the command block with any value other than the default value *and* the VECTOR SCALE command line does not appear in the command block, then the vector scale will be determined by the user-specified value for the small strain. If the VECTOR SCALE command line does appear in the POWER METHOD PARAMETERS command block, then the vector scale will be determined by the user-specified value for the vector scale.

The recommended way of determining the scale factor is to use the small-strain approach with the default small-strain value. Thus, if you follow this advice, neither the SMALL STRAIN command line nor the VECTOR SCALE command line should be included in the POWER METHOD PARAMETERS command block.

As indicated previously, the scale factor must not be too small, as this will create round-off problems and give a bad estimate for the critical time step. If the scale factor is too large, we violate the above restriction of a constant tangent stiffness matrix.

The `SCALE FACTOR` command line sets the factor f_s in Equation (3.11). The value for f_s, i.e., `time_scale` is set to 0.9. More experience with the power method will help us determine whether a slightly larger value of f_s can be used as a default.

The `UPDATE ON TIME STEP CHANGE` command line sets the value for Δt_{lim} in Equation (3.14). If the change in the element-based critical time step estimate as given by Equation (3.14) exceeds the value for Δt_{lim}, then the power method is called for a new estimate for the critical time step. The default value for Δt_{lim}, i.e., `tstep_change`, is 0.10.

The `UPDATE STEP INTERVAL` command line sets the number of step intervals at which the power method is called. If `step_int` is set to 40, the power method will be called every forty steps to compute an estimate for the critical time step. (The default value of `step_int` is 50.) This control mechanism interacts with the control established by the `UPDATE ON TIME STEP CHANGE` command line. Suppose we have set `step_int` to 40, and we have computed thirty-five steps since the last call to the power method. If Δt_{lim} has been set to 0.15 and we exceed this value at step 35, then the change in the element-based time step will result in the power method being called. The counter for keeping track of the number of step intervals since the last power method computation will be reset to zero. The next call to the power method will then occur again in forty steps, unless we again exceed the change in the element-based time step.

3.2.3 Node-Based Method

Now that we have developed schemes to make the Lanczos method and power method cost-effective tools for estimation of the critical time step, let us examine the node-based scheme.

The node-based method in Presto will give an estimate for the critical time step that is greater than or equal to the element-based estimate. It may or may not give an estimate for the time step that is close to the maximum value associated with the maximum eigenvalue for the problem. In general, we assume the node-based method will give us an estimate larger than the element-based estimate, but not significantly larger.

If the node-based scheme is used to determine the critical time step for a block of uniform elements (all the same material), then the estimate from the node-based method will be the same as that for the element-based estimate. The node-based estimate begins to diverge from (become larger than) the element-based estimate as the differences in aspect ratios of the elements attached to a node become larger. We can assume, therefore, that if the stiffest part of our structure has a relatively uniform mesh, the node-based method will not give a significantly larger estimate than the element-based method.

The node-based method costs only a fraction of an internal force calculation. However, the node-based method makes use of element time step estimates. Therefore, in order to do the node-based method, one must also do the element-based method. The cost of the node-based method is in addition to the element-based method. The modest increase in the critical step estimate from the node-based method is unlikely to offset the added cost of the node-based method.

To make the node-based method cost-effective, we can use the same methodology that is employed for the Lanczos method. Let t_b be the estimate for the critical time step from the node-based

method. We define the ratio of the node-based estimate to the element-based estimate as

$$t_r = \frac{\Delta t_b}{\Delta t_e}. \tag{3.22}$$

This ratio is then used to scale subsequent element-based estimates for the critical time step. If $\Delta t_{e(n)}$ is the nth element-based critical time step after the time step where the node-based calculations are performed, then the nth time step after the node-based calculations, $\Delta t_{(n)}$, is simply

$$\Delta t_{(n)} = t_r \Delta t_{e(n)}. \tag{3.23}$$

The ratio t_r is used until the next call to the node-based method. The next call to the node-based method is controlled by one of two mechanisms described in the Lanczos discussion. The node-based method is called after a set number of times or a significant change in the element-based estimate for the critical time step.

3.2.3.1 Node-Based Parameters Command Block

```
BEGIN NODE BASED TIME STEP PARAMETERS <string>nbased_name
  INCREMENT INTERVAL = <integer>incr_int(5)
  STEP INTERVAL = <integer>step_int(1)
  TIME STEP LIMIT = <real>step_lim(0.10)
END [NODE BASED TIME STEP PARAMETERS <string>nbased_name]
```

If you use the node-based method to compute a critical time step, there should be only one `NODE BASED TIME STEP PARAMETERS` command block, and it should appear in the region. If you use the node-based method to compute the critical time step, you should set the time step scale factor to 1.0 and the time step increase factor to a large number (5.0 is an acceptable value for the time step increase factor). These factors are specified in the `PARAMETERS FOR PRESTO REGION` portion of a `TIME STEPPING BLOCK` in the `TIME CONTROL` command block. If you have a `NODE BASED TIME STEP PARAMETERS` command block, you should not specify a `LANCZOS PARAMETERS` command block.

The `INCREMENT INTERVAL` command line determines how many steps, `incr_int`, are used to transition from an element-based critical time step estimate to the node-based estimate at the beginning of an analysis. The user may want to increase from the element-based estimate to the node-based estimate over a number of time steps if the difference between these two estimates is large. The value of `incr_int` defaults to 0 so the node based time step is used immediately at the beginning of the analysis.

The `STEP INTERVAL` command line sets the number of step intervals at which the node-based method is called. If `step_int` is set to 1000, the node-based method will be called every one thousand steps to compute an estimate for the critical time step. (The default value of `step_int` is 1.) Increasing the step interval can improve performance, but depending on the problem could run the risk of sending the time step super-critical. This control mechanism interacts with the control established by the `TIME STEP LIMIT` command line. Suppose we have set `step_int` to

1000, and we have computed 800 steps since the last call to the node-based method. If Δt_{lim} has been set to 0.15 and we exceed this value at step 800, then the change in the element-based time step will result in the node-based method being called. The counter for keeping track of the number of step intervals since the last node-based computation will be reset to zero. The next call to the node-based method will occur again in one thousand steps, unless we again exceed the change in the element-based time step.

The `TIME STEP LIMIT` command line sets the value for Δt_{lim} in Equation (3.14). If the change in the element-based critical time step estimate as given by Equation (3.14) exceeds the value for Δt_{lim}, then the node-based method is called for a new estimate for the critical time step. The default value for Δt_{lim}, i.e., `step_lim`, is 0.10.

3.3 Mass Scaling

3.3.1 What is Mass Scaling?

Mass scaling allows for arbitrarily increasing the mass of certain nodes in order to increase the global estimate for the critical time step. The nodes where the mass is increased must be associated with those elements that have the minimum time step. By increasing the mass at any node for an element, we have effectively raised the critical time step estimate for that element.

Note that mass scaling does not adjust the value for the density used in the element calculations. Mass scaling only adjusts the mass at the nodes. The net effect of the mass scaling makes it appear, however, as if we have modified the density of selected elements (even though no adjustment has been made to element densities).

Warning: Use of mass scaling will introduce error into your analysis. The amount of error incurred is unbounded and can be unpredictable. It is entirely up to the analyst to decide whether mass scaling can be used in a way that does not distort the results of interest.

Mass scaling can be useful in a number of circumstances, as listed below. However, in all of these circumstances, error will be introduced into the calculations. The user must be extremely careful not to introduce excessive error.

- Quasi-static or rigid-body motion: If the model or part of a model is undergoing what is basically quasi-static or rigid-body motion, then adding mass may have little effect on the end result.
- Disparate sizes in element geometry: A model may contain elements for some portion of the model that are much smaller than the majority of elements in the rest of the model. For example, a model might include screws or gears that are modeled in detail. The elements for the screw threads or gear teeth could be much smaller than elements in other portions of the model. If the dynamics of these parts modeled with small elements (compared to the rest of

the mesh) are relatively unimportant, adding mass to them might not affect the quantities of interest.

- Increasing time step for “unimportant" sections of the mesh: For some problems, you may not want part of the mesh to control the time step. Consider a car-crash problem in which the bumper is the first part to strike an object. The crumpling of the bumper could greatly reduce the time step in some elements of the bumper, and these elements would control the time step for the problem. At a later time in the analysis, the effect of the bumper on the overall crash dynamics may not be significant. Mass scaling could be applied to ensure that this now noncritical part (the bumper) is no longer controlling the global time step.

3.3.2 Mass Scaling Command Block

```
BEGIN MASS SCALING
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list> nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  TARGET TIME STEP = <real>target_time_step
  ALLOWABLE MASS INCREASE RATIO = <real>mass_increase_ratio
  #
  # additional command
  ACTIVE PERIODS = <string list>periods
  INACTIVE PERIODS = <string list>periods
END MASS SCALING
```

The `MASS SCALING` command block controls mass scaling for a specified set of nodes. This command block contains one or more command lines to specify the node set. It also contains two command lines that determine how the actual mass scaling will be applied to the nodes in the node set. In addition to the command lines in the two command groups, there are two additional command lines: `ACTIVE PERIODS` and `INACTIVE PERIODS`. These command lines are used to activate or deactivate the mass scaling for specified time periods.

Multiple `MASS SCALING` command blocks can exist to apply different criteria to different portions of the mesh at different times. For any given set of `MASS SCALING` command blocks, mass will only be added to a node if doing so will allow increasing the global time step. The amount of artificial mass added to a node will vary in time as the mesh deforms and moves. The added mass computation is recomputed every time the node-based time step estimate is recomputed.

NOTE: Mass scaling must be used in conjunction with the node-based time step estimation method.

See the preceding sections for a description of the node-based method for estimating the critical time step.

Following are descriptions of the different command groups:

3.3.3 Node Set Commands

The `node set commands` portion of the `MASS SCALING` command block specifies the nodes associated with the boundary condition. This portion of the command block can include some combination of the following lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list> nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes for mass scaling. There must be at least one `NODE SET`, `SURFACE`, `BLOCK`, or `INCLUDE ALL BLOCKS` command line in the command block.

3.3.3.1 Mass Scaling Commands

The `MASS SCALING` command block may contain either a

```
TARGET TIME STEP = <real>target_time_step
```

command line or an

```
ALLOWABLE MASS INCREASE RATIO = <real>mass_increase_ratio
```

command line, or both of these command lines can appear in the input block.

The `TARGET TIME STEP` command lines sets the maximum time step for a set of nodes. The parameter `target_time_step` is the maximum time step for all the nodes specified in the command block.

The `ALLOWABLE MASS INCREASE RATIO` command line sets an upper limit on the mass scaling at a node. The value specified for `mass_increase_ratio` limits the ratio of the mass at a node, as set by mass scaling, to the original mass at the node. (The original mass of the node is determined only by the element contributions.) This ratio must be a factor greater than or equal to 1. If m_s is the scaled mass at a node and m_0 is the original mass at the node due only to element contributions, then the ratio $m_s \div m_0$ will not exceed the value of `mass_increase_ratio`.

Mass scaling will add mass to nodes until the target time step is reached, the mass added to some node reaches the allowable mass increase ratio, or the current set of nodes no longer controls the

global analysis time step.

The amount of mass added due to mass scaling is stored in the nodal variable `mass_scaling_added_mass`. This variable can be output and post-processed to determine how much mass is being added at a given time. See Section 9.2.1 regarding the output of nodal variables to a results file.

3.3.3.2 Additional Commands

The `ACTIVE PERIODS` or `INACTIVE PERIODS` command lines can optionally appear in the `MASS SCALING` command block:

```
ACTIVE PERIODS = <string list>periods
INACTIVE PERIODS = <string list>periods
```

These command lines determine when mass scaling is active. See Section 2.5 for more information about these command lines.

3.4 Explicit Control Modes

Presto provides an algorithm known as "explicit control modes" that uses a coarse mesh overlaying the actual problem mesh, referred to as the reference mesh. The name "control modes" comes from the implicit multigrid solution algorithm in Adagio with that name. The explicit control modes algorithm uses the coarse and reference meshes together to enable a potentially significant increase in the critical time step in explicit dynamic calculations. Reference 3 provides an in-depth discussion of this algorithm.

Warning: Explicit Control Modes is a new and experimental analysis technique. While it has been shown to be an extremely useful technique on some problems, it has not undergone rigorous testing. It also does not interoperate with some features, such as rigid bodies.

In the explicit dynamic algorithm, the nodal accelerations are computed by dividing the residual (external force minus internal force) by the nodal mass. In the explicit control modes algorithm, the reference mesh residual is mapped to the coarse mesh, and accelerations are computed on the coarse mesh. These accelerations are then interpolated back to the reference mesh. The portion of the residual with higher frequency content that cannot be represented by the basis functions of the coarse mesh is left on the reference mesh, and is used to compute an acceleration that is added to the acceleration that is interpolated from the coarse mesh.

By computing the acceleration on the coarse mesh, the explicit control modes algorithm allows for the critical time step to be computed based on the size of the coarse mesh rather than the size of the reference mesh. A critical time step is estimated based on the coarse mesh, and mass scaling is applied to the high frequency component of the acceleration that is computed on the reference

mesh. While some error is introduced due to this mass scaling, this error only applies to the high frequency part of the response, which is often well beyond the frequency range of interest. This is in contrast to traditional mass scaling, which affects the full spectrum of structural response.

The choice of the degree of refinement in the coarse and reference meshes has a large influence on the effectiveness of the explicit control modes algorithm. The reference mesh should be created to give a discretization that is appropriate to capture the geometry of the problem with sufficient refinement to adequately represent gradients in the discretized solution. The coarse mesh should completely overlay the reference mesh, and it should be coarser than the reference mesh at every location in the model. All coarse elements need not contain elements in the reference mesh, so it is possible to use a coarse mesh that extends significantly beyond the domain of the reference mesh.

To maximize solution efficiency by increasing the critical time step, the coarse mesh should be significantly coarser than the reference mesh. The code user has the freedom to create a coarse mesh that gives an acceptable critical time step without using an excessively crude discretization. It is important to remember that the reference mesh controls the spatial discretization, while the coarse mesh controls the temporal discretization of the model.

To use explicit control modes, the user should set up the reference mesh file and the input file as usual, except that the following additional items must be provided:

- A coarse mesh must be generated, as discussed above. The coarse mesh must be in a separate file from the reference mesh, which is the real model.

- A second `FINITE ELEMENT MODEL` command block must be provided in addition to the standard definition for the reference finite element model in the input file. This command block is set up exactly as it normally would be (see Section 6.1), except that the mesh file referenced is the coarse mesh instead of the reference mesh. Although the coarse mesh does not use any material models, each block in the coarse mesh must still be assigned a material model.

- A `CONTROL MODES REGION` command block must appear alongside the standard `PRESTO REGION` command block within the `PRESTO PROCEDURE` command block. The presence of the `CONTROL MODES REGION` command block instructs Presto to use the explicit control modes algorithm. The `CONTROL MODES REGION` command block is documented in Section 3.4.1. It contains the same commands used within the standard `PRESTO REGION` command block, except that the commands in the `CONTROL MODES REGION` command block are used to control the control modes algorithm and the boundary conditions on the coarse mesh.

3.4.1 Control Modes Region

```
BEGIN CONTROL MODES REGION
  #
  # model setup
  USE FINITE ELEMENT MODEL <string>model_name
  CONTROL BLOCKS [WITH <string>coarse_block] =
```

```
    <string list>control_blocks
  #
  # time step control
  TIME STEP RATIO SCALING = <real>cm_time_scale(1.0)
  TIME STEP RATIO FUNCTION = <string>cm_time_func
  LANCZOS TIME STEP INTERVAL = <integer>lanczos_interval
  POWER METHOD TIME STEP INTERVAL = <integer>pm_interval
  #
  # mass scaling
  HIGH FREQUENCY MASS SCALING = <real>cm_mass_scale(1.0)
  #
  # stiffness damping
  HIGH FREQUENCY STIFFNESS DAMPING COEFFICIENT =
    <real>cm_stiff_damp(0.0)
  #
  # kinematic boundary condition commands
  BEGIN FIXED DISPLACEMENT
    #
    # Parameters for fixed displacement
    #
  END [FIXED DISPLACEMENT]
  #
  # output commands
  BEGIN RESULTS OUTPUT <string> results_name
    #
    # Parameters for results output
    #
  END RESULTS OUTPUT <string> results_name
END [CONTROL MODES REGION]
```

The CONTROL MODES REGION command block controls the behavior of the control modes algorithm, and is placed alongside a standard PRESTO REGION command block within the PRESTO PROCEDURE scope. With the exception of the CONTROL BLOCKS command line, all the commands that can be used in this block are standard commands that appear in the Presto region. These commands have the same meaning in either context; they simply apply to the coarse mesh or to the reference mesh, depending on the region block in which they appear. Sections 3.4.1.1 through 3.4.1.3 describe the components of the CONTROL MODES REGION command block.

3.4.1.1 Model Setup Commands

```
USE FINITE ELEMENT MODEL <string>model_name
CONTROL BLOCKS [WITH <string>coarse_block] =
  <string list>control_blocks
```

The command lines listed above must appear in the CONTROL MODES REGION command block if explicit control modes is used. The USE FINITE ELEMENT MODEL command line should refer-

ence the finite element model for the coarse mesh. This command line is used in the same way that the command line is used for the reference mesh (see Section 2.3).

The CONTROL BLOCKS command line provides a list of blocks in the reference mesh that will be controlled by the coarse mesh. The block names are listed using the standard method of referencing mesh entities (see Section 1.5). For example, the block with an ID of 1 would be listed as block_1 in this command. Multiple CONTROL BLOCKS command lines may be used.

The CONTROL BLOCKS command line does not require the coarse blocks used to control the fine blocks to be listed. In the following example, blocks 10 and 11 are controlled by the coarse mesh, but the element blocks in the coarse mesh that control those blocks are not listed:

```
CONTROL BLOCKS = block_10 block_11
```

If the CONTROL BLOCKS command line is used in this manner, the search for fine nodes contained within coarse elements will be conducted for all elements in the coarse mesh. The coarse block used to control a given set of fine blocks can optionally be specified by using the CONTROL BLOCKS WITH coarse_block variant of the command. For example, the command:

```
CONTROL BLOCKS WITH block_1 = block_10 block_11
```

would use block 1 on the coarse mesh to control blocks 10 and 11 on the fine mesh. This variant of the command is necessary when the coarse blocks overlap. It removes any ambiguity about which coarse elements control which fine nodes. This is particularly useful for contact problems where the fine block on one side of an interface should be controlled by one block, and the fine block on the other side of the interface should be controlled by a different block. Only one coarse block can be listed in a given instance of this command, so if there are multiple coarse blocks, they must be listed in separate commands.

3.4.1.2 Time Step Control Commands

```
TIME STEP RATIO SCALING = <real>cm_time_scale(1.0)
TIME STEP RATIO FUNCTION = <string>cm_time_func
LANCZOS TIME STEP INTERVAL = <integer>lanczos_interval
POWER METHOD TIME STEP INTERVAL = <integer>pm_interval
```

The control modes algorithm computes a node-based time step for the coarse mesh at each time step, and uses this as the default time step. This time step is typically much larger than the critical time step for the fine mesh.

The TIME STEP RATIO SCALING and TIME STEP RATIO FUNCTION command lines allow the user to control the time step used with explicit control modes. The TIME STEP RATIO SCALING command is used to specify a scale factor cm_time_scale, which has a default value of 1.0. The TIME STEP RATIO FUNCTION command is used to specify a function cm_time_func that is used to control the scale factor as a function of time. At any given time, a scale factor, f_{ts}, is computed by multiplying cm_time_scale by the current value of the function. Both of these commands are optional and one can be used without the other.

The time step Δt, is computed as a function of f_{ts}, as well as of the time step of the fine mesh, Δt_f

and the time step of the coarse mesh, Δt_c.

$$\Delta t = \Delta t_f + f_{ts}(\Delta t_c - \Delta t_f) \tag{3.24}$$

Thus, if the scale factor is zero, the time step of the fine mesh is used, and if it is one, the time step of the coarse mesh is used.

The nodal time step estimator for the coarse mesh typically works well on problems where the fine mesh overlaid by the coarse mesh is essentially isotropic. In cases where it is not, such as when there are significant voids covered by the coarse mesh, the nodal time step can be non-conservative, resulting in stability problems. The time step control command lines described above can be used to manually scale down the time step in such scenarios.

Alternatively, the Lanczos or power method global time step estimators can be applied to the coarse mesh to give an improved estimate of the stability limit. These are invoked using the `LANCZOS TIME STEP INTERVAL` or `POWER METHOD TIME STEP INTERVAL` command lines, respectively. Only one of these command lines can be used at a time, and both commands specify an interval at which the global time step estimate is calculated. When the global time step estimate is calculated, a ratio of the global estimate to the nodal estimate is calculated, and this ratio is used to scale the nodal estimate in subsequent time steps in which the global estimate is not computed.

Experience has shown that the time step predicted by the global time step estimators is typically slightly higher than the actual stability limit. For this reason, it is recommended that a scale factor of 0.9 be used in conjunction with these estimators. This can be set using the `TIME STEP SCALE FACTOR` command line in the `TIME CONTROL` block as described in Section 3.1.

Known Issue: The Lanczos and power method time step estimators can not yet be used with problems that have contact, rigid bodies, blocks in the fine mesh that are not controlled by the coarse mesh, or coarse elements that contain no fine nodes.

3.4.1.3 Mass Scaling Commands

```
HIGH FREQUENCY MASS SCALING = <real>cm_mass_scale(1.0)
```

The `HIGH FREQUENCY MASS SCALING` command line allows the user to control the mass scaling applied to the high frequency component of the response. The mass scaling factor required to stably integrate the high frequency response at the time step being used is computed at every node on the fine mesh. The parameter `cm_mass_scale` that can optionally be supplied with this command line is applied as a multiplier to that mass scaling. If that mass scaling (multiplied by `cm_mass_scale`) is greater than 1.0, then the scaled mass is used at that node. If not, the original nodal mass is used.

It may be useful for some models to use this command line to set `cm_mass_scale` to a value greater than 1.0 to stabilize the high frequency response. Experience has shown, however, that this is rarely needed, so it is recommended that this command line not be used in most cases.

3.4.1.4 Damping Commands

```
HIGH FREQUENCY STIFFNESS DAMPING COEFFICIENT =
  <real>cm_stiff_damp(0.0)
```

The HIGH FREQUENCY STIFFNESS DAMPING COEFFICIENT command is used to apply stiffness-proportional damping on the high frequency portion of the response in explicit control modes. This may help reduce high frequency noise in problems that have abrupt loading such as that caused by contact. The default value of cm_stiff_damp is 0.0. The value specified for cm_stiff_damp can be between 0 and 1. It is recommended that small values (around 0.001) be specified if this option is used.

3.4.1.5 Kinematic Boundary Condition Commands

```
BEGIN FIXED DISPLACEMENT
  #
  # Parameters for fixed displacement
  #
END FIXED DISPLACEMENT
```

All types of kinematic boundary conditions can be applied to the coarse mesh. This is done by inserting a kinematic boundary condition command block in the CONTROL MODES REGION command block. The mesh entity (node set, surface, or block) to which the boundary condition is applied must exist on the coarse mesh.

This capability is potentially useful to ensure better enforcement of kinematic boundary conditions on the fine mesh by applying the same type of boundary condition on the portion of the coarse mesh that overlays the portion of the fine mesh to which boundary conditions are applied. For example, if there is a node set on the fine mesh that has a fixed displacement boundary condition, a node set can be created on the coarse mesh that covers the same physical domain. The same fixed displacement boundary condition could then be applied to the coarse mesh.

Although the capability to enforce boundary conditions on the coarse mesh is provided, it is not necessary to do so. It is also often very difficult to create a node set on the coarse mesh that matches the discretization of the node set on the fine mesh. Users are advised to initially prescribe kinematic boundary conditions only on the fine mesh and only prescribe boundary conditions on the coarse if the initial results appear questionable.

3.4.1.6 Output Commands

```
BEGIN RESULTS OUTPUT <string> results_name
  #
  # Parameters for results output
  #
END RESULTS OUTPUT <string> results_name
```

Variables can be output from the coarse mesh just as they can from the fine mesh with explicit control modes. Because the actual results of interest for the model all reside on the fine mesh, it is typically not necessary to output results on the coarse mesh, but this can be helpful for debugging purposes.

The syntax for the results output for the coarse mesh is identical to that used for output from the fine mesh, and is documented in Section 9.2. The only thing that differentiates the `RESULTS OUTPUT` command block for the coarse mesh from that of the fine mesh is that the results output block for the coarse mesh is put in the `CONTROL MODES REGION` command block instead of in the `PRESTO REGION` command block. The output files for the coarse and fine mesh must be different from each other, so different output file names must be used within the output blocks for the coarse and fine meshes.

One of the most useful variables to output from the coarse mesh is the nodal `timestep`. This variable is similar in nature to the element `timestep`, which exists on the fine mesh, but is a nodal rather than an element variable, and exists on the coarse mesh. It reports the critical time step calculated for each node on the coarse mesh. If the coarse time step is higher than expected, the output from `nodal_time_step` can be examined to see which region of the coarse mesh is controlling the time step.

Central difference time integration is performed on the coarse mesh in addition to the fine mesh, so the `displacement`, `velocity`, and `acceleration` variables can be requested for visualization on the coarse mesh.

3.5 References

1. Hughes, T. J. R. *The Finite Element Method:Linear Static and Dynamic Finite Element Analysis*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

2. Golub, G. H., and C. F. Van Loan *Matrix Computations, Third Edition*. Baltimore, MD: The John Hopkins University Press, 1996.

3. Spencer, B. W., M. W. Heinstein, J. D. Hales, K. H. Pierson and J. R. Overfelt. *Multi-Length Scale Algorithms for Failure Modeling in Solid Mechanics*, SAND2008-6499. Albuquerque, NM: Sandia National Laboratories, October 2008.

Implicit Only

Chapter 4

Implicit Solver, Time Step, and Dynamics

This chapter discusses the commands used to control the solver, time stepping, and dynamics parameters for implicit calculations. The commands and features documented in this chapter are applicable only for implicit quasistatic and dynamic analyses. For information on time stepping commands for explicit dynamic analyses, see Chapter 2.7.

Nonlinear solvers are a core part of any implicit finite element code. The solution strategy used in Sierra/SM is based on iterative solution techniques, and builds on the capabilities pioneered in the JAS3D code [1]. Sierra/SM uses a nonlinear preconditioned conjugate gradient (CG) algorithm [2] to iteratively find a solution that satisfies equilibrium within a user-specified error tolerance at each load step.

The nonlinear CG algorithm iterates to find a solution by computing a series of search direction vectors that are successively added to the trial velocity vector. In each iteration, the residual is multiplied by a preconditioning matrix to obtain a gradient direction. The gradient direction is used to compute the next search direction in a way that ensures that each new search direction is orthogonal to the previous search directions. A line search is used to compute a scaling factor that when applied to the search direction results in a minimized residual.

The performance of the CG algorithm is highly dependent on the conditioning of the problem, which is affected by the choice of preconditioner. Using the inverse of the full tangent stiffness matrix as a preconditioner minimizes the number of CG iterations required, but this can be computationally expensive. An alternative is to use the inverse of a diagonalized stiffness matrix as a preconditioner. This requires much lower computational and memory resources, per iteration, but may require many more iterations.

Sierra/SM provides several preconditioner options that can be categorized in two general types. These enable the efficient solution of a wide variety of problems using available resources. The first type of preconditioner is the nodal preconditioner. There are several different variants of nodal preconditioners available. These include the diagonal preconditioners that were available in JAS3D, in addition to other options. The second type of preconditioner is the full tangent preconditioner. This option is typically used with the FETI parallel scalable linear solver to solve for the action of the full tangent matrix.

In addition to using a preconditioner, conditioning of a problem can be improved by using a multi-level solver. Rather than directly solving the potentially ill-posed full problem with the CG solver, Sierra/SM forms a series of “model problems” and solves them with the core CG solver. Features of the model that make it ill-posed are either adjusted or held constant for a model problem. After each model problem is solved, the multilevel solver updates the quantities that were controlled. The process of forming model problems and updating is iterative, as with the core solver, and

convergence must be achieved both in the core solver and in the multilevel controls.

Three types of controls are available for the multilevel solver: "control contact," "control stiffness," and "control failure". Control contact must be used for all problems that have sliding contact, but is not needed for tied contact. Control stiffness is used to solve problems that are difficult because of extreme differences in the stiffness of various modes of material response. For instance, control stiffness is beneficial for nearly incompressible materials where the bulk response is much stiffer than the shear response. Control stiffness is also useful for oriented materials where the material response is much stiffer in some directions than it is in others. In addition, using control stiffness can greatly speed the iterative solution process when a model contains several materials that have extreme differences in stiffness. Control failure is used on problems that involve element death.

Sierra/SM also provides a geometric multigrid algorithm called "control modes," which uses a coarse grid to solve for the low mode response. To use the control modes algorithm, a user must supply a coarse mesh that overlays the actual problem mesh. This algorithm works well for solving problems that are dominated by bending of slender members.

This chapter discusses the commands for specifying solver options, for controlling time stepping, and for solving implicit dynamic problems. The CG solver can either be used alone or as the core solver within a multilevel solver. Section 4.1 presents the characteristics and structure of the `SOLVER` command block, which contains all solver-related commands within the `ADAGIO REGION` command block. Section 4.2 discusses the `CG` command block, which controls the nonlinear preconditioned CG solver. An option in the `CG` command block is to use a full tangent preconditioner. The command block to control the behavior of this option is discussed in Section 4.3. The recommended linear solver for use with the full tangent preconditioner is FETI, whose command block is presented in Section 4.4. The three types of controls, control contact, control stiffness, and control failure, that can be used within the multilevel solver are presented in Sections 4.5, 4.6, and 4.7. Section 4.8 explains how to activate the multigrid control modes solution method. This feature greatly increases the effectiveness of the CG algorithm for solving slender, bending-dominated problems. Section 4.9 describes the predictors, which are used to generate an initial trial solution for a load step or for a multilevel-solver model problem. As explained in Section 4.10, Sierra/SM also has an option that allows it to use the same algorithms as the JAS3D solver. The techniques to control time stepping and to perform implicit solutions on quasi-static and dynamic problems are described in Sections 4.11 and 4.12, respectively. Finally, Section 4.13 provides references for this chapter.

4.1 Multilevel Solver

Sierra/SM allows the conjugate gradient (CG) solver to be used either by itself or as the core solver within a multilevel solver. The multilevel solver improves the ability of the core solver to solve poorly conditioned problems. This is done by holding fixed some variables that would ordinarily be free to change in the fully nonlinear iteration. The multilevel solver algorithm can be used recursively to treat multiple variables by assigning them to multiple solution levels. CG iterations are performed while a variable is held fixed. After convergence is obtained with the controlled variable fixed, a new residual calculation is performed, and the controlled variable is updated. Convergence on a particular level requires that all levels leading up to that level must be converged.

Once all control variables have been established, CG iterations are performed until convergence is obtained on the core “model problem.” At this point, the variable controlled at level 1 is adjusted to minimize its residual. This process is known as a “level 1 update.” After the level 1 update, the model problem is solved again, and this process is repeated until the core solver and the variable controlled at level 1 have converged. Once a solution has been obtained at level 1, the variable controlled at level 2 is updated. After each level 2 update, the level 1 problem must be solved again. This process can be repeated recursively for many levels. For convergence in the multilevel solver, all updates starting with the lowest level and proceeding to the highest level must be in equilibrium. These concepts are illustrated in Figure 4.1.

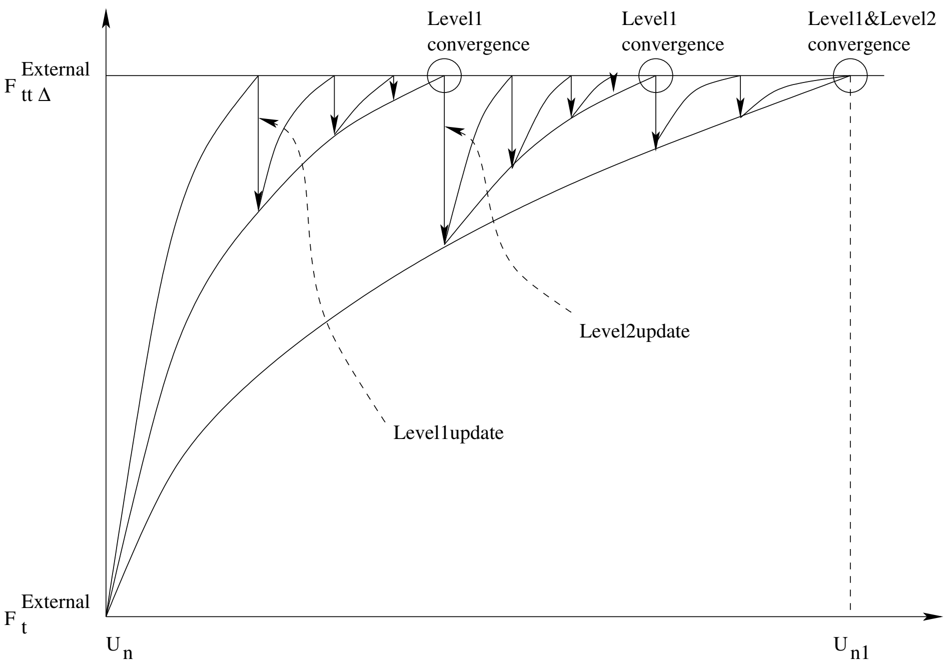


Figure 4.1: Reaching convergence with controls and the multilevel solver.

Sierra/SM has three types of controls that can be used within the multilevel solver: control contact, control stiffness, and control failure. Control contact is used to keep the set of nodes in contact constant during the solution of a model problem. Contact searches are performed during the mul-

tilevel solver updates. Control contact must be used for all problems that have sliding contact. It is not required or beneficial for models that only have tied contact. Control contact is documented in detail in Section 4.5.

Control stiffness is used to improve the conditioning of models that contain widely varying stiffnesses in different modes of material response. Such differences naturally exist for models with nearly incompressible materials where the bulk behavior is much stiffer than the shear behavior, for models with orthotropic and/or anisotropic materials that have much higher stiffnesses in preferred material directions, and for models containing several materials that are vastly different in their overall stiffnesses. Control stiffness works by scaling a given material response up (stiffening) or down (softening) to create a set of better conditioned model problems. For nearly incompressible materials, for instance, the bulk stress increments may be softened and/or the shear stress increments may be stiffened. For oriented materials, the stress increments in certain material directions are softened to create a material response that is more isotropic in nature. Finally, for the case where a material is much stiffer than the surrounding materials, both the bulk and shear stress increments are softened to create a response that has a stiffness much closer to that of the adjacent materials. Ultimately, as the control stiffness model problems progress, the true material response is recovered by building up stresses that correspond to the true material response. There are a number of special material models that are designed to work with this capability.

Control failure is used to improve the conditioning of models that involve element death due to failure defined within a material model or due to a user-defined global variable failure criteria. Currently, failure due to element or nodal variables is not supported in implicit analyses. This control limits the failure only to the single most critical element during each control failure iteration. Iterations of the control failure continue until no additional elements fail.

The following is the basic structure of the SOLVER command block:

```
BEGIN SOLVER
  #
  # cg solver commands
  BEGIN CG
    #
    # Parameters for cg
    #
  END [CG]
  #
  # control contact commands
  BEGIN CONTROL CONTACT [<string>contact_name]
    #
    # Parameters for control contact
    #
  END [CONTROL CONTACT <string>contact_name]
  #
  # control stiffness commands
  BEGIN CONTROL STIFFNESS [<string>stiffness_name]
    #
    # Parameters for control stiffness
```

```
    #
  END [CONTROL STIFFNESS <string>stiffness_name]
  #
  # control failure commands
  BEGIN CONTROL FAILURE [<string>failure_name]
    #
    # Parameters for control failure
    #
  END [CONTROL FAILURE <string>failure_name]
  #
  # predictor commands
  BEGIN LOADSTEP PREDICTOR
    #
    # Parameters for predictor
    #
  END [LOADSTEP PREDICTOR]
  LEVEL 1 PREDICTOR = <string>NONE|DEFAULT(DEFAULT)
END [SOLVER]
```

The `SOLVER` command block contains all solver-related commands, and must be present in the `ADAGIO REGION` command block. It is used regardless of whether the CG solver should be used alone or as a core solver in the multilevel solver. The `CG` command block (described in Section 4.2) is required for all models. If no multilevel controls are desired, the commands related to those controls are simply omitted from this block.

In addition to the `CG` command block, the `SOLVER` command block contains command blocks that describe the controls that will be used. The `SOLVER` command block can contain any or all of `CONTROL CONTACT` command block, `CONTROL STIFFNESS` command block, and `CONTROL FAILURE` command blocks described in Sections 4.5, 4.6, and 4.7.

Aside from the command blocks for the core solver and the controls, the `SOLVER` block contains the `LOADSTEP PREDICTOR` command block, which is described in Section 4.9.1, and the `LEVEL 1 PREDICTOR` command line, which is described in Section 4.9.2.

Convergence tolerances for the `CONTROL CONTACT` and `CONTROL STIFFNESS` solver levels are set within those command blocks while the tolerance of the core CG solver is set within the `CG` command block. Rather than use a convergence tolerance, the `CONTROL FAILURE` block is considered converged when no new elements fail during an iteration or when the maximum iterations is reached.

The commands within the `CG` command block are used in the same way whether or not the multilevel solver is used. It is, however, important to consider the role of tolerances within the multilevel solver. The convergence of the problem is controlled by the convergence of the highest solver level. For the multilevel solver to converge well, it is important for the core solver (or lower level of the solver, in the case of multiple levels) to reduce the residual by a certain amount during each model problem solution. For this reason, it is recommended that the solution tolerances be set an order of magnitude tighter for each lower level solver. The `MINIMUM RESIDUAL IMPROVEMENT` command in the `CG` command block is helpful for ensuring that the residual is reduced by the core

solver during model problems, even though the residual might already be within the convergence tolerances (see Section 4.2.1).

4.2 Conjugate Gradient Solver

The core nonlinear preconditioned conjugate gradient solver is controlled through the `CG` command block, which must be nested within a `SOLVER` command block, regardless of whether multilevel controls are to be used if it is to be used without multilevel controls.

```
BEGIN CG
  #
  # convergence commands
  # Default is not defined.
  TARGET RESIDUAL = <real>target_resid
    [DURING <string list>period_names]
  TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-4)
    [DURING <string list>period_names]
  # Default is 10 times target residual
  ACCEPTABLE RESIDUAL = <real>accept_resid
    [DURING <string list>period_names]
  # Default is 10 times target relative residual
  ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid(1.0e-3)
    [DURING <string list>period_names]
  REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
    [DURING <string list>period_names]
  # Default is not defined.
  MINIMUM RESIDUAL IMPROVEMENT = <real>resid_improvement
    [DURING <string list>period_names]
  MINIMUM ITERATIONS = <integer>min_iter(0)
    [DURING <string list>period_names]
  RESIDUAL ROUNDOFF TOLERANCE = <real>(1.0e-15)
  #
  # default = 100 for tangent preconditioner.
  # default = max(NumNodes,1000) for all other preconditioners.
  MAXIMUM ITERATIONS = <integer>max_iter
    [DURING <string list>period_names]
  #
  # preconditioner commands
  PRECONDITIONER = BLOCK|BLOCK_INITIAL|DIAGONAL|
   DIAGSCALING|ELASTIC|IDENTITY|PROBE|TANGENT(ELASTIC) [<real>scaling_factor]
  PRECONDITIONER ITERATION UPDATE = <integer>iter_update
  BALANCE PROBE = <integer>balance_probe(1.0e-6)
  NODAL PROBE FACTOR = <real>probe_factor(1.0e-6)
  NODAL DIAGONAL SCALE = <real>nodal_diag_scale(0.0)
  NODAL DIAGONAL SHIFT = <real>nodal_diag_shift(0.0)
  BEGIN FULL TANGENT PRECONDITIONER
    #
    # Parameters for full tangent preconditioner
    #
  END [FULL TANGENT PRECONDITIONER]
```

```
  #
  # line search command, default is secant
  LINE SEARCH ACTUAL|TANGENT [DURING <string list>period_names]
  LINE SEARCH SECANT [<real>scale_factor]
  #
  # diagnostic output commands
  ITERATION PRINT = <integer>iprint(25) [DURING <string list>period_names]
  # off by default.
  ITERATION PLOT = <integer>iplot [DURING <string list>period_names]
  ITERATION PLOT OUTPUT BLOCKS = <string list>plot_blocks
  #
  # cg algorithm commands
  ITERATION RESET = <integer>iter_reset(10000)
    [DURING <string list>period_names]
  ORTHOGONALITY MEASURE FOR RESET = <real>ortho_reset(0.5)
    [DURING <string list>period_names]
  RESET LIMITS <integer>iter_start <integer>iter_reset
    <real>reset_growth <real>reset_orthogonality
    [DURING <string list>period_names]
  BETA METHOD = FletcherReeves|PolakRibiere|
    PolakRibierePlus(FletcherReeves)
    [DURING <string list>period_names]
  # line search step length bounds
  MINIMUM STEP LENGTH = <real>min_step(-infinity)
  MAXIMUM STEP LENGTH = <real>max_step(infinity)
END [CG]
```

Sections 4.2.1 through 4.2.5 describe the components of the CG command block.

4.2.1 Convergence Commands

```
# Default is not defined.
TARGET RESIDUAL = <real>target_resid
  [DURING <string list>period_names]
TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-4)
  [DURING <string list>period_names]
ACCEPTABLE RESIDUAL = <real>accept_resid(1.0e-3)
  [DURING <string list>period_names]
# Default is 10 times target relative residual
ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid
  [DURING <string list>period_names]
REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
  [DURING <string list>period_names]
# Default is not defined.
MINIMUM RESIDUAL IMPROVEMENT = <real>resid_improvement
  [DURING <string list>period_names]
MINIMUM ITERATIONS = <integer>min_iter(0)
  [DURING <string list>period_names]
# default = 100 for tangent preconditioner
# default = max(NumNodes,1000) for all other preconditioners.
MAXIMUM ITERATIONS = <integer>max_iter
  [DURING <string list>period_names]
RESIDUAL ROUNDOFF TOLERANCE = <real>1.0e-15)
  [DURING <string list>period_names]
```

The nonlinear preconditioned CG solver iterates to decrease the residual until the residual is deemed to be sufficiently small, based on user-specified convergence criteria. The command lines listed above are placed in the CG command block and used to control these convergence criteria.

Solver convergence is measured by computing the L^2 norm of the residual and comparing that residual norm with target convergence criteria specified by the user. Convergence can be monitored either directly in terms of the residual norm, or in terms of a relative residual, which is the residual norm divided by a reference quantity that is indicative of the current loading conditions on a model. Basing convergence on the relative residual is often helpful because doing so ensures that the convergence target is meaningful for the model. There are some situations, however, when it is better to check for convergence based on the actual residual rather than on the relative residual. This approach is especially helpful when the model passes through a stage during which there are no loads. Note that the actual residual, the relative residual, or both of these residuals can be used for checking convergence at the same time.

All the convergence command lines in the CG command block can have different values for different time periods by using the DURING specification. This specification consists of the key word DURING followed by a list of time periods, and is appended to the end of the command line. Each of the time periods included in the list must correspond to the name of a TIME STEPPING command

block (i.e., time_block_name) specified in the TIME CONTROL command block. If the DURING specification is omitted from the command line, the value of the parameter in the command line is applicable for the entire analysis. This default value can be overwritten for specific periods by repeating the same command line and using the DURING specification on each repeated line. In other words, multiple occurrences of the same command line can be included in the command block, one without a DURING specification and each of the others with a unique DURING specification for its applicable time periods.

The TARGET RESIDUAL command line specifies the target convergence criterion in terms of the actual residual norm. The TARGET RELATIVE RESIDUAL command line specifies a convergence criterion in terms of the relative residual. If both command lines are included in the CG command block, the solver will accept the solution as converged if either the target residual or the target relative residual is below the specified values. Note that use of the term "target" in this discussion means "desired."

The solver also allows the user to define acceptable convergence criteria. If the solution has not converged to the specified targets before the maximum number of iterations of the solver is reached, the residual is checked against the acceptable convergence criteria. These criteria are specified via the ACCEPTABLE RESIDUAL and ACCEPTABLE RELATIVE RESIDUAL command lines. The concepts of residual and relative residual are the same as those used for the target limits, i.e., in the TARGET RESIDUAL and TARGET RELATIVE RESIDUAL command lines. If the solution has not met the target criteria but has met the acceptable criteria, the solution is allowed to proceed. The default value for each acceptable criterion command line is 10 times the value of its corresponding target criterion command line, e.g., if the value of target_resid was 1.0e-6, the value of accept_resid would be 1.0e-5. Solutions that meet only the acceptable criteria are noted in the log file. If the acceptable criteria are not met, the solution at that load step has failed to converge, and the solver exits. If adaptive time stepping, as discussed in Section 4.11.2, is active, a solution of the load step may be attempted with a smaller time step. Otherwise, the code will exit at this point with an error.

If relative residuals are given in the convergence criteria, the REFERENCE command line can be used to select the method for computing the reference load. This command line has several options: EXTERNAL, INTERNAL, BELYTSCHKO, RESIDUAL and ENERGY. When the EXTERNAL option, which is the default, is selected, the reference load is computed by taking the L^2 norm of the current external load. If the model has force boundary conditions, the external force vector used for this norm is composed of the nodal forces resulting from those boundary conditions. If there are no prescribed forces, the reaction forces at all prescribed kinematic boundary conditions are used instead. The INTERNAL option uses the norm of the internal forces as the reference load. This option is helpful in situations where a structure has no externally applied boundary conditions, such as in the case of a thermally loaded structure. The BELYTSCHKO option uses the maximum of the norm of the external load, the norm of the reactions, and the norm of the internal forces. The RESIDUAL option denotes that the residual from the initial residual should be used as the reference quantity. The initial residual is computed from the first iteration of the solver. The ENERGY residual is the energy conjugate norm of the residual and reference vectors. The energy norm of the residual is given by $R.K^{-1}.R$ where K is the diagonal of the stiffness matrix. The primary advantage of the energy norm is that forces and moments are converted to consistent and compatible energy units.

The MINIMUM RESIDUAL IMPROVEMENT command line stipulates that the CG solver must reduce the initial residual by a specified amount to obtain convergence. If this command line is included in the CG command block, the improvement condition must be met in addition to the standard target residual criteria. The parameter resid_improvement is a real number between 0.0 and 1.0 that specifies the amount by which the residual must be improved as a fraction of the initial residual in the current model problem. Thus, to stipulate that the residual must be smaller than 10% of the initial residual, one would set the value of resid_improvement equal to 0.9. The MINIMUM RESIDUAL IMPROVEMENT command line is primarily useful in the context of the multilevel solver. The model problems presented to the core solver sometimes begin with very low residuals and require very few iterations to converge. This command line forces the core solver to further improve the residual, which can accelerate the convergence of the other solver levels.

The MAXIMUM ITERATIONS and MINIMUM ITERATIONS command lines are used to specify the maximum and minimum number of core nonlinear solver iterations, respectively. These limits apply for a load step if the CG solver is used in the stand-alone mode, or for a model problem if the CG solver is used as the core solver in the multilevel solver. The default maximum number of iterations is set to $max(N_{nodes}, 1000)$ where N_{nodes} is the global number of nodes in the mesh. The linear CG algorithm is guaranteed to converge to the solution in a number of iterations equal to the number of equations in the matrix. The number of iterations can be larger than the number of equations since we are solving a nonlinear system of equations using a nonlinear CG algorithm. The default value for the minimum number of iterations, min_iter, is zero. If a number greater than zero is specified, the solver will iterate at least that many times, regardless of whether the convergence criteria are met.

The RESIDUAL ROUNDOFF TOLERANCE command line specifies a tolerance used to compute an approximately zero residual. Due to finite precision arithmetic the solver can only drop the residual so far before additional corrections to the nodal coordinate field cannot be numerically represented. The approximately zero residual is the residual that would be produced by a perturbation of the displacement by the smallest machine representable perturbation. The default value for this command is the accuracy of double precision arithmetic (1.0e-15). This tolerance number can be set to zero to disable this feature or set higher to converge to larger, but still near machine limit, residuals.

The log file annotates which convergence criterion was met in each CG iteration using markings shown in Table refconvergenceMarkTable.

4.2.2 Preconditioner Commands

```
PRECONDITIONER = BLOCK|BLOCK_INITIAL|DIAGONAL|
  DIAGSCALING|ELASTIC|IDENTITY|PROBE|TANGENT
  [<real>scaling_factor]
PRECONDITIONER ITERATION UPDATE = <integer>iter_update
BALANCE PROBE = <integer>balance_probe
NODAL PROBE FACTOR = <real>probe_factor(1.0e-6)
NODAL DIAGONAL SCALE = <real>nodal_diag_scale(0.0)
NODAL DIAGONAL SHIFT = <real>nodal_diag_shift(0.0)
BEGIN FULL TANGENT PRECONDITIONER
```

Table 4.1: Log File Convergence Markings

Mark	Convergence Property
>M	Convergence is reached, but still have not iterated the minimum number of specified iterations
>A	Reached maximum iterations and still not converged
<T	Residual converged to target tolerance of either relative or absolute criteria
<A	Reached maximum iterations and converged to acceptable tolerance of either relative or absolute criteria
:C	Intermediate convergence
~0	Residual is converged due to being approximately zero to within machine precision for the dimensions and units used in the analysis. This approximately zero value is effected by the RESIDUAL ROUNDOFF TOLERANCE command

```
#
#  Parameters for full tangent preconditioner
#
END [FULL TANGENT PRECONDITIONER]
```

The command lines listed above are used to select the type of preconditioner used by the CG algorithm, as well as the method used to form the preconditioner for some types of preconditioners. The preconditioner is applied to the residual vector to obtain a gradient direction, which in turn is used to find a search direction. The most effective preconditioner is one that closely approximates the effect of multiplying by the inverse of the tangent stiffness matrix.

There are two basic types of preconditioners available: nodal and full tangent. The nodal preconditioners provide only the diagonal terms of the full tangent stiffness matrix or 3-by-3-block diagonal matrices along the diagonal of the stiffness matrix. The diagonal nature of a nodal preconditioner allows it to be inverted inexpensively and also use very little memory. Nodal preconditioners are so termed because they only account for the stiffness at each node in isolation and ignore the coupling between nodes that is accounted for in the off-diagonal terms of a full stiffness matrix. The result of this approximation is that with a nodal preconditioner, in a single iteration, the equilibration of a residual at a node can only cause movement in nodes that are directly connected by an element to that node. Thus, as the model size increases, the number of iterations also increases. Nodal preconditioners require many iterations, but they are often efficient because the iterations are inexpensive, especially if the problem is “blocky” in nature.

With the PRECONDITIONER command line, the user selects the nodal preconditioner. As defined below, this command line has several options. Some options are synonyms for other options.

- The BLOCK and ELASTIC options, which are synonyms, specify the default preconditioner. These options provide a nodal preconditioner with 3-by-3-block matrices that are computed based on the elastic material properties. These matrices are updated at every model problem to account for the current geometry.

- The BLOCK_INITIAL option, which is a variant of the BLOCK preconditioner, forms the preconditioner at the beginning of the analysis but never recomputes it for efficiency.
- The DIAGONAL preconditioner is formed in the same way as the BLOCK preconditioner, but only uses the terms on the diagonal.
- The PROBE option forms a 3-by-3-block diagonal preconditioner by probing the stiffness of nodes rather than by using the elastic stiffness.
- The IDENTITY option uses an identity matrix for the preconditioner, meaning that the residual is used as the gradient direction. This option is only of academic interest and is included for completeness.
- The DIAGSCALING option uses the identity matrix multiplied by the parameter scaling_factor. This is the only option for which the scaling_factor parameter is applicable. This option is also only of academic interest.

The PRECONDITIONER ITERATION UPDATE command line controls the frequency at which the full tangent and nodal probe preconditioner are updated. By default, the preconditioner is only updated at the first iteration of each model problem. If this command line is used, updates will occur at the specified frequency iter_update. If a model experiences highly nonlinear behavior over the course of a model problem, it may be useful to use this command to cause more frequent preconditioner updates. Such problems may converge better when iter_update is set to about 10 for the full tangent case. It is important to realize that frequent updates may reduce the number of CG iterations but dramatically increase the computational cost.

The BALANCE PROBE and NODAL PROBE FACTOR command lines control the behavior of the probing algorithm that is used to obtain the stiffness for the probe preconditioner (selected via the PROBE option in the PRECONDITIONER command line). For the probe preconditioner, the tangent stiffness K is approximated using finite differences:

$$K_{ij} = \frac{\partial F_i^{int}}{\partial x_j}, \tag{4.1}$$

where F^{int} is the internal force and x is the displacement. The nodal probe preconditioner only computes the 3-by-3-block diagonal version of the stiffness matrix. The NODAL PROBE FACTOR command line controls the probing distance, with the value of probe_factor defaulting to 1.0e-6. The probe factor controls the probing distance relative to element size. Smaller values may give a better tangent approximation, but these values could result in the generation of round-off errors.

The BALANCE PROBE command line selects the type of finite differencing scheme used to probe for the stiffness. The value of balance_probe can be set to 0, 1, or 2, as explained below.

- Setting balance_probe to 0, the default for the nodal probe preconditioner, causes the preconditioner to be formed by forward finite differencing. Probing occurs in only one direction, resulting in zero-order accuracy in the preconditioner. The stiffness computed when balance_probe is set to 0 appears as

$$K_{ij} = \frac{\partial F_i^{int}(x + \delta e_j) - F_i^{int}(x)}{\delta}, \tag{4.2}$$

where δ is the probing distance, controlled by the NODAL PROBE FACTOR command line, and e_j is a unit vector in the jth equation direction.

- Setting balance_probe to 1, the default for the full tangent preconditioner, causes the preconditioner to be formed with central differencing. Probing at each element degree of freedom is performed in both positive and negative directions, leading to a first-order accurate estimate of the tangent stiffness. This approach can be necessary in problems with material nonlinearity, where a probe that stretches rather than compresses the element can result in orders of magnitude differences in estimated stiffness. The stiffness computed when balance_probe is set to 1 appears as

$$K_{ij} = \frac{\partial F_i^{int}(x + \delta e_j) - F_i^{int}(x - \delta e_j)}{2\delta}. \tag{4.3}$$

- The value of balance_probe may also be set to 2, which allows the full tangent preconditioner to obtain central finite differencing with fourth-order error. This approach requires twice the work as central differencing but is more accurate. The stiffness computed when balance_probe is set to 2 appears as

$$K_{ij} = \frac{\partial F_i^{int}(x + 2\delta e_j) + 8F^{int}(x + \delta e_j) - 8F^{int}(x + \delta e_j) - F_i^{int}(x - 2\delta e_j)}{12\delta}. \tag{4.4}$$

The NODAL DIAGONAL SCALE command line is used to specify a nodal diagonal scale factor, which is used to scale the diagonal entries of the nodal preconditioner. This is useful in some situations where a model may be softening due to material effects. The diagonal terms of the nodal preconditioner are scaled by (1.0 + nodal_diagonal_scale). The default value of nodal_diagonal_scale is 0.0, which means the diagonals are not scaled. A value of 0.1 scales up all the diagonal terms by 10 percent.

The NODAL DIAGONAL SHIFT command line is used to specify a nodal diagonal shift, which is summed into the diagonal entries of the nodal preconditioner. This is useful in some situations where a model may be softening due to material effects. The diagonal terms of the nodal preconditioner are shifted by (num_nodes * tangent_diagonal_shift). The default value of nodal_diagonal_shift is 0.0, which means the diagonals are not shifted.

The FULL TANGENT PRECONDITIONER command block enables the full tangent preconditioner and specifies options that apply specifically to that type of preconditioner. See Section 4.3 for a description of the options available. The FULL TANGENT PRECONDITIONER command block cannot be used with the PRECONDITIONER command line in a CG command block. The user must select only one type of preconditioner for an analysis.

4.2.3 Line Search Command

```
LINE SEARCH ACTUAL|TANGENT [DURING <string list>period_names]
LINE SEARCH SECANT [<real>scale_factor]
```

During each CG iteration, after the search direction is computed, a line search is used to find an optimal scaling factor that when applied to the search vector will result in a minimized residual. The line search algorithm is controlled with the `LINE SEARCH` command line. Three line search types are available: `ACTUAL`, `SECANT`, and `TANGENT`. The `SECANT` option is the default, and is used if the `LINE SEARCH` command line is not present.

- The `ACTUAL` line search is a single-step quadratic line search that is equivalent to the corresponding JAS3D line search option. The line search requires an additional evaluation for each iteration of the element internal forces and thus the material response. For a given search direction s, residual r, and velocity v, the step length α is computed as

$$\alpha = -\frac{s^T r(v)}{s^T (F_{int}(v+s) - F_{int}(v))}, \tag{4.5}$$

  where the residual is computed based on the internal force F_{int} and on the external force F_{ext} as $r = F_{int}(v) - F_{ext}(v)$.

- The `SECANT` line search is a slight variation of the `ACTUAL` line search described above. Like the `ACTUAL` line search, this line search requires an additional internal force evaluation for each iteration. The step length α is computed as

$$\alpha = -\frac{s^T r(v)}{s^T (r(v+s) - r(v))}. \tag{4.6}$$

  The optional `scale_factor` can be used with the secant line search to scale the the search vector s in the equation above. If `scale_factor` is omitted, no scaling is done. If `scale_factor` is provided on the command line, s is scaled by the product of `scale_factor` and the dimension of the smallest element. Setting the scale factor to a small value can be helpful to avoid inverting elements during the evaluation of $r(v + s)$. The `SECANT` line search is recommended for problems in which the external force is highly nonlinear (e.g., a beam-bending problem with a pressure distribution).

- The `TANGENT` line search avoids the additional internal force calculation that is inherent in the other line search types by using the tangent modulus computed in the last call to the material subroutine for each element. The step length α is computed as

$$\alpha = -\frac{s^T r(v)}{s^T K s}, \tag{4.7}$$

  where K is the tangent stiffness.

4.2.4 Diagnostic Output Commands

```
ITERATION PRINT = <integer>iter_print
  [DURING <string list>period_names]
ITERATION PLOT = <integer>iter_plot
  [DURING <string list>period_names]
ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
```

The command lines listed above can be used to control the output of diagnostic information from the CG solver. The ITERATION PRINT and ITERATION PLOT command lines can be appended with the DURING specification, as discussed in Section 4.2.1.

The ITERATION PRINT command line controls the frequency at which the convergence information line is printed to the log file. The default value of iter_print is 25 if a nodal preconditioner is used, and it is 1 if a full tangent preconditioner is used.

The ITERATION PLOT command line allows plots of the current state of the model to be written to the output database during the CG iterations. The value supplied in iter_plot specifies the frequency of such plots. The default behavior is not to produce such plots. The plots are generated with a fictitious time step. Every time such a plot is generated, a message is sent to the log file. This feature is useful for debugging, but it should be used with care. Note that the ITERATION PLOT command line can be appended with the DURING specification, as discussed in Section 4.2.1.

The output produced by iteration plots is sent by default to every active results output file, as specified by a RESULTS OUTPUT command block. The ITERATION PLOT OUTPUT BLOCKS command line can optionally be used to supply a list of output block names to which iteration plots should be written. Each of the output block names listed in plot_blocks, must match the name of a RESULTS OUTPUT command block (see Section 9.2.1). The ITERATION PLOT OUTPUT BLOCKS command line can be useful for directing iteration plots to separate output files so that these plots are separated from the converged solution output. It can also be used to send output plots for different levels of the multilevel solver to different files.

4.2.5 CG Algorithm Commands

```
ITERATION RESET = <integer>iter_reset(10000)
  [DURING <string list>period_names]
ORTHOGONALITY MEASURE FOR RESET = <real>ortho_reset(0.5)
  [DURING <string list>period_names]
RESET LIMITS <integer>iter_start <integer>iter_reset
  <real>reset_growth <real>reset_orthogonality
  [DURING <string list>period_names]
BETA METHOD = FletcherReeves|PolakRibiere|
  PolakRibierePlus(FletcherReeves)
  [DURING <string list>period_names]
MINIMUM STEP LENGTH = <real>min_step(-infinity)
MAXIMUM STEP LENGTH = <real>max_step(infinity)
```

The behavior of the CG algorithm can be changed using the command lines listed above. The CG algorithm orthogonalizes the current search direction and the previous search direction at each iteration. During an iteration where a reset occurs, the orthogonalization is skipped, resulting in a search in the steepest descent direction. Note that all of these command lines can be appended with the DURING specification, as discussed in Section 4.2.1.

The ITERATION RESET command line specifies that the CG algorithm be reset after every iter_reset iterations. The default is to reset the algorithm after every 10,000 iterations.

For the CG algorithm to work well, each new search direction must be sufficiently orthogonal to the previous search directions. If the current search direction has a significant component in one of the previous search directions, error can be introduced that will not be corrected in future iterations. At every iteration, the relative lack of orthogonality between the current gradient direction and the previous search direction is computed. The `ORTHOGONALITY MEASURE FOR RESET` command line specifies that a reset will occur if the lack of orthogonality exceeds the value of `ortho_reset`. The default value of 0.5 results in few resets. Setting `ortho_reset` to a value as tight as 0.01 can result in many resets and potentially improved convergence on some problems.

The `RESET LIMITS` command line is also used to control CG resets. This command line is provided to achieve compatibility with JAS3D, and it behaves exactly the same as its JAS3D counterpart. The command line, however, should not be used when either the `ITERATION RESET` command line or the `ORTHOGONALITY MEASURE FOR RESET` command line is included in the `CG` command block. Up to four parameters can optionally be specified in the `RESET LIMITS` command line. The first, `iter_start`, sets the number of iterations to wait before looking for a minimum residual. The second, `iter_reset`, sets the number of iterations to allow between finding a minimum and restarting the iterative algorithm. The third, `reset_growth`, sets the amount of growth in the residual norm that would indicate divergence and thus trigger a reset. Finally, `reset_orthogonality` sets the relative lack of orthogonality between the current gradient direction and the previous search direction that would trigger a reset.

The `BETA METHOD` command line allows different formulas to be used in computing the β scalar in the CG algorithm. If set to `FletcherReeves`, the β scalar is computed as

$$\beta_k = \frac{r_k^T g_k}{r_{k-1}^T g_{k-1}}. \tag{4.8}$$

If set to `PolakRibiere` (the default), β is computed as

$$\beta_k = \frac{r_k^T (g_k - g_{k-1})}{r_{k-1}^T g_{k-1}}. \tag{4.9}$$

With the `PolakRibierePlus` option, β is computed as

$$\beta_k = \max(\beta_k^{PR}, 0). \tag{4.10}$$

In the above equations, r_k is the current iteration's residual vector, r_{k-1} is the previous iteration's residual vector, g_k is the current iteration's gradient vector, and g_{k-1} is the previous iteration's gradient vector. The `PolakRibierePlus` formula forces beta to be positive. A steepest descent direction is taken when beta becomes negative.

The `MINIMUM STEP LENGTH` and `MAXIMUM STEP LENGTH` commands sets the minimum step length and maximum step length that the CG algorithm can take along a search direction. The default values are set so that any positive or negative step length will be accepted. This option can be used to limit motion of the model along any search direction and can help improve convergence in some cases.

4.3 Full Tangent Preconditioner

In addition to the nodal preconditioners, Sierra/SM provides the capability to use a full tangent preconditioner in the CG algorithm. The full tangent preconditioner employs a scalable parallel linear solver to solve for the gradient direction using the full tangent stiffness matrix. Although the full tangent preconditioner is significantly more costly per CG iteration than are the nodal preconditioners, it can solve problems in a very small number of iterations. Full tangent preconditioners are especially effective in solving poorly conditioned problems that are often very difficult to solve using the CG algorithm with nodal preconditioners, such as those problems that involve the bending response of long, slender members.

To use a full tangent preconditioner generally two command blocks are added to the input file, one required and one optional. First, the `FULL TANGENT PRECONDITIONER` command block, described in this section, must be added to the `CG` command block. The `FULL TANGENT PRECONDITIONER` command block is used instead of the `PRECONDITIONER` command line in the `CG` command block. Second, a command block for a linear solver may be defined in the SIERRA scope, as described in Section 4.4. That linear solver command block may then be referenced from within the `FULL TANGENT PRECONDITIONER` command block, instructing the CG algorithm to use the specified linear solver. If no linear solver command block is given and/or no linear solver is referenced in the `FULL TANGENT PRECONDITIONER` command block, the FETI linear solver will be employed with default settings. The command block to enable the full tangent preconditioner is as follows:

Known Issue: Deactivation of element blocks (see Section 6.1.5.9) in conjunction with the full tangent preconditioner is available but not yet fully tested. Use caution when combining these capabilities.

```
BEGIN FULL TANGENT PRECONDITIONER
  #
  # solver selection commands
  LINEAR SOLVER = <string>linear_solver_name(FETI)
  NODAL PRECONDITIONER METHOD = ELASTIC|PROBE|DIAGONAL|BLOCK
    (ELASTIC)
  #
  # tangent matrix formation commands
  PROBE FACTOR = <real>probe_factor(1.0e-6)
  BALANCE PROBE = <integer>balance_probe(1)
  CONSTRAINT ENFORCEMENT = SOLVER|PENALTY(PENALTY)
  PENALTY FACTOR = <real>penalty_factor(100.0)
  TANGENT DIAGONAL SCALE = <real>tangent_diag_scale(0.0)
  TANGENT DIAGONAL SHIFT = <real>tangent_diag_shift(0.0)
  CONDITIONING = NO_CHECK|CHECK|AUTO_REGULARIZATION(CHECK)
  #
  # reset and iteration commands
  MAXIMUM RESETS FOR MODELPROBLEM = <integer>max_mp_resets
```

```
    (100000) [DURING <string list>period_names]
  MAXIMUM RESETS FOR LOADSTEP = <integer>max_ls_resets
    (100000) [DURING <string list>period_names]
  MAXIMUM ITERATIONS FOR MODELPROBLEM
    = <integer>max_mp_iter(100000)
    [DURING <string list>period_names]
  MAXIMUM ITERATIONS FOR LOADSTEP = <integer>max_ls_iter
    (100000) [DURING <string list>period_names]
  ITERATION UPDATE = <integer>iter_update
    [DURING <string list>period_names]
  SMALL NUMBER OF ITERATIONS = <integer>small_num_iter
    [DURING <string list>period_names]
  MINIMUM SMOOTHING ITERATIONS = <integer>min_smooth_iter(0)
    [DURING <string list>period_names]
  MAXIMUM SMOOTHING ITERATIONS = <integer>max_smooth_iter(0)
    [DURING <string list>period_names]
  TARGET SMOOTHING RESIDUAL = <integer>tgt_smooth_resid(0)
    [DURING <string list>period_names]
  TARGET SMOOTHING RELATIVE RESIDUAL
    = <integer>tgt_smooth_rel_resid(0)
    [DURING <string list>period_names]
  #
  # fall-back strategy commands
  STAGNATION THRESHOLD = <real>stagnation(1.0e-12)
    [DURING <string list>period_names]
  MINIMUM CONVERGENCE RATE = <real>min_conv_rate(1.0e-4)
    [DURING <string list>period_names]
  ADAPTIVE STRATEGY = SWITCH|UPDATE(SWITCH)
    [DURING <string list>period_names]
END [FULL TANGENT PRECONDITIONER]
```

The `FULL TANGENT PRECONDITIONER` command block contains all command lines related to the interaction with the linear solver. There are command lines to control selection of the linear solver, formation of the matrices passed into the linear solver, CG iteration strategies, updating strategies, and strategies for dealing with poor convergence. Reasonable defaults have been set for most problems. The command lines in this command block are described in Sections 4.3.1 through 4.3.4.

4.3.1 Solver Selection Commands

```
LINEAR SOLVER = <string>linear_solver_name
NODAL PRECONDITIONER METHOD = ELASTIC|PROBE|DIAGONAL|BLOCK(ELASTIC)
```

The command lines listed above are used in selecting a linear solver for use with the full tangent preconditioner and in selecting a nodal preconditioner. The `LINEAR SOLVER` command line specifies the name of the solver that will be used to compute the action of the preconditioner by solving

the linear system. If no linear solver is specified, Sierra/SM will default to using the FETI linear solver with its default settings. Linear solvers are defined in the SIERRA scope. Although several linear-solver packages are available for use as preconditioners in Sierra/SM, we recommend that the FETI equation solver be used in production analyses. The FETI equation solver is actively maintained and tested by the Sierra/SM development team. The FETI EQUATION SOLVER command block is documented in Section 4.4.

An arbitrary number of equation solver command blocks can be included in the input file to define various sets of solver options. Each equation solver command block must be given a name, which is referenced in the linear_solver_name parameter of the LINEAR SOLVER command line. This name instructs Sierra/SM to use the specified linear solver as a preconditioner for the CG solver.

When Sierra/SM uses a full tangent preconditioner, it also forms a nodal preconditioner. This nodal preconditioner is used in contact calculations, as a fall-back preconditioner if the full tangent preconditioner does not perform well, and optionally for performing smoothing iterations prior to use of the full tangent preconditioner.

The nodal preconditioner is selected with the NODAL PRECONDITIONER METHOD command line. Four options are available: ELASTIC, PROBE, DIAGONAL, and BLOCK. The ELASTIC option is the default. These options have the same meaning as the corresponding options given for the nodal preconditioner in the CG command block, as documented in Section 4.2.2.

4.3.2 Matrix Formation Commands

```
PROBE FACTOR = <real>probe_factor(1.0e-6)
BALANCE PROBE = <integer>balance_probe(1)
CONSTRAINT ENFORCEMENT = SOLVER|PENALTY(PENALTY)
PENALTY FACTOR = <real>penalty_factor(100.0)
TANGENT DIAGONAL SCALE = <real>tangent_diag_scale(0.0)
TANGENT DIAGONAL SHIFT = <real>tangent_diag_shift(0.0)
CONDITIONING = NO_CHECK|CHECK|AUTO_REGULARIZATION(CHECK)
```

The command lines listed above can be used to optionally control the way the stiffness matrix is formed. This matrix is formed by assembling contributions from all active elements. These element stiffness matrices are formed by finite differencing.

The PROBE FACTOR command line controls the probing distance relative to element size The default value of probe_factor is 1.0e-6. Smaller values may give a better tangent approximation, but they may also generate round-off errors.

The BALANCE PROBE command line selects the type of finite-differencing scheme used to probe for the stiffness. The value of balance_probe can be set to 0, 1, or 2. Setting balance_probe to 0 causes the preconditioner to be formed by forward finite differencing. Probing occurs in only one direction, resulting in zero-order accuracy in the preconditioner. Setting balance_probe to 1, the default for the full tangent preconditioner, causes the preconditioner to be formed with central differencing. Probing at each element degree of freedom is performed in both positive and

negative directions, leading to a first-order accurate estimate of the tangent stiffness. The value of balance_probe may also be set to 2, which allows the full tangent preconditioner to obtain central finite differencing with fourth-order error. This scheme requires twice the work as the central-differencing scheme but is more accurate. See Section 4.2.2 for a more detailed discussion of these options.

The CONSTRAINT ENFORCEMENT command line controls the way multipoint constraints (i.e. contact, rigid body, and periodic BC) are enforced in the linear system. Two options are available: SOLVER and PENALTY. Specifying SOLVER results in the constraints being passed to the solver, thus allowing the solver to handle constraint enforcement internally. Specifying PENALTY results in the constraints being converted to penalty "elements" whose stiffness is contributed to the linear system before it is passed to the linear solver. If the PENALTY option is used, the penalty stiffness is controlled by the PENALTY FACTOR command line.

Although some of the available linear-solver packages support internal constraint enforcement, these packages often perform poorly in parallel. Thus, we recommend that CONSTRAINT ENFORCEMENT be set to PENALTY for all analyses. The errors caused by penalty compliance are corrected in each CG iteration, so the converged solution will not have errors that are due to penalty compliance. This strategy has proven to be very effective for dealing with constraints.

The PENALTY FACTOR command line is used to specify the penalty stiffness. At each constraint, the diagonal tangent stiffness of the slave degree of freedom is multiplied by the value of the penalty_factor parameter. The default value for this parameter is 100.0, which is reasonable for most problems. Assigning a high value (greater than about 1.0e+8) can cause poor conditioning of the linear system and also cause the linear solver to perform poorly. Assigning a low value (less than about 10) could require more CG iterations due to the correction of errors resulting from penalty compliance.

The TANGENT DIAGONAL SCALE command line is used to specify a tangent matrix diagonal scale factor, which is used to scale the diagonal entries of the tangent matrix. This is useful in some situations where a model does not have sufficient boundary conditions to remove all rigid body modes. This command has also been used as a last resort to get convergence when all else fails. The diagonal terms of the tangent matrix are scaled by (1.0 + tangent_diagonal_scale). The default value of tangent_diagonal_scale is 0.0, which means the tangent matrix diagonals are not scaled. A value of 0.1 scales up all tangent matrix diagonal terms by 10 percent.

The TANGENT DIAGONAL SHIFT command line is used to specify a tangent matrix diagonal shift, which is summed into the diagonal entries of the tangent matrix. This is useful in some situations where a model does not have sufficient boundary conditions to remove all rigid body modes. This command has also been used as a last resort to get convergence when all else fails. The diagonal terms of the tangent matrix are shifted by (num_nodes * tangent_diagonal_shift). The default value of tangent_diagonal_shift is 0.0, which means the tangent matrix diagonals are not shifted.

The CONDITIONING command line controls the linear system conditioning inside the FETI solver. It has no effect on any other linear system solver. By default when used with the full tangent preconditioner FETI checks the conditioning of its coarse level matrix and reports a warning to the log file if it finds ill-conditioning (defined as a condition number greater than 1e5). This default

behavior corresponds to the CHECK option. For performance optimization this check may be turned off using the NO_CHECK option. The third option, AUTO_REGULARIZATION, instructs FETI to add a small shift to its coarse matrix to regularize the linear system and artificially reduce the condition number. In the context of the CG solver, this will affect convergence but not the residual calculations. Coarse level matrix ill-conditioning often indicates an issue with the problem at hand, such as the presence of zero energy rigid body modes either inherently or temporarily as a result of a CG iteration, and AUTO_REGULARIZATION may improve CG convergence in the presence of ill-conditioning.

4.3.3 Reset and Iteration Commands

```
MAXIMUM RESETS FOR MODELPROBLEM = <integer>max_mp_resets
  (100000) [DURING <string list>period_names]
MAXIMUM RESETS FOR LOADSTEP = <integer>max_ls_resets
  (100000) [DURING <string list>period_names]
MAXIMUM ITERATIONS FOR MODELPROBLEM = <integer>max_mp_iter
  (100000) [DURING <string list>period_names]
MAXIMUM ITERATIONS FOR LOADSTEP = <integer>max_ls_iter
  (100000) [DURING <string list>period_names]
ITERATION UPDATE = <integer>iter_update
  [DURING <string list>period_names]
SMALL NUMBER OF ITERATIONS = <integer>small_num_iter(0)
  [DURING <string list>period_names]
MINIMUM SMOOTHING ITERATIONS = <integer>min_smooth_iter(0)
  [DURING <string list>period_names]
MAXIMUM SMOOTHING ITERATIONS = <integer>max_smooth_iter(0)
  [DURING <string list>period_names]
TARGET SMOOTHING RESIDUAL = <integer>tgt_smooth_resid(0)
  [DURING <string list>period_names]
TARGET SMOOTHING RELATIVE RESIDUAL
  = <integer>tgt_smooth_rel_resid(0)
  [DURING <string list>period_names]
```

When a linear solver is used as a preconditioner, the efficiency of the solution can often be dramatically affected by the frequency at which the preconditioner is updated. The computational expense of updating the preconditioner, which involves forming and factorizing the stiffness matrix, is often much higher than the cost of applying the preconditioner, which involves only a back-solve, during a CG iteration. If the stiffness does not dramatically change from iteration to iteration, it is often much more efficient to re-use the preconditioner for a number of iterations, or even load steps, than it is to update it at every iteration.

The command lines listed above can be used to control when the preconditioner is updated. As Sierra/SM allows for the nodal preconditioner to be used in place of the full tangent preconditioner in some situations, these command lines also control this behavior. Note that all of these command lines can be appended with the DURING specification, as discussed in Section 4.2.1.

The MAXIMUM RESETS FOR MODELPROBLEM command line limits the number of times that the preconditioner can be reset (updated) during a model problem. By default, there is no limit on the number of updates that can occur during a model problem. The MAXIMUM RESETS FOR LOADSTEP command line limits the updates that can occur during a load step. By default, there is also no limit on the number of updates that can occur during a load step.

The MAXIMUM ITERATIONS FOR LOADSTEP and MAXIMUM ITERATIONS FOR MODELPROBLEM command lines can be used to limit the number of CG iterations that will be performed using the full tangent preconditioner per model problem or load step, respectively. The values of the parameters max_mp_iter and max_ls_iter are both unlimited by default. If one of these iteration limits is reached, the CG solver will continue to iterate using the fall-back nodal preconditioner until either the CG solver iteration limit is reached or convergence is achieved. It can be useful to limit the iterations taken by the full tangent preconditioner. If the full tangent preconditioner is performing poorly, the nodal preconditioner may work better in some cases.

The ITERATION UPDATE command line controls the frequency at which the full tangent preconditioner is updated. By default, it is only updated at the first iteration of each model problem. If this command line is used, the preconditioner will be updated at the frequency specified by iter_update. If a model experiences highly nonlinear behavior over the course of a model problem, it may be useful to use this command line to cause more frequent preconditioner updates. Such problems may converge better when iter_update is set to about 10. It is important to realize that frequent updates may reduce the number of CG iterations but dramatically increase the computational cost.

It can often be efficient to update the preconditioner very infrequently. The SMALL NUMBER OF ITERATIONS command line is used to “freeze” the preconditioner, or prevent updates, until a model problem requires more iterations than the number specified in small_num_iter. The default value of small_num_iter is 0, meaning that the preconditioner is never re-used for the next model problem unless the SMALL NUMBER OF ITERATIONS command line is used. It is often beneficial to set small_num_iter to 10. This can often result in the re-use of the preconditioner over many load steps. When this command line is used, the CG solver often converges in a very small number of iterations immediately following a preconditioner update; however, over the course of several load steps, the iteration counts slowly increase because the preconditioner is out of date. At a certain point, the iteration count exceeds small_num_iter, and the preconditioner is reset. This solution strategy is very efficient because it allows re-use of the preconditioner while it is still relatively effective.

In some cases it is beneficial to perform a number of CG iterations using the nodal preconditioner prior to applying the full tangent preconditioner. The iterations with the nodal preconditioner are used as a “smoother” to put the model in a better state for the full tangent preconditioner. The smoothing iterations are controlled using the MINIMUM SMOOTHING ITERATIONS, MAXIMUM SMOOTHING ITERATIONS, TARGET SMOOTHING RESIDUAL, and TARGET SMOOTHING RELATIVE RESIDUAL command lines. By default, no smoothing iterations are taken. Using these commands, the user can specify that smoothing iterations should be taken, and control how many are taken.

The number of smoothing iterations taken can vary based on the convergence of the solver dur-

ing those iterations. The smoothing iteration commands allow the user to control the desired convergence of the smoother before switching to the full tangent preconditioner. These commands behave the same way as the standard solver convergence commands, except that they apply only to the smoothing iterations. The standard convergence criteria still must be met to obtain a converged solution. The MINIMUM SMOOTHING ITERATIONS command controls the minimum smoothing iterations taken, and the MAXIMUM SMOOTHING ITERATIONS command controls the maximum smoothing iterations. The TARGET SMOOTHING RESIDUAL and TARGET SMOOTHING RELATIVE RESIDUAL control the residual and relative residual, respectively, that must be obtained in the smoothing iterations before switching to the full tangent preconditioner. Once tgt_smooth_resid has been reached, the solver switches from using the nodal preconditioner to the tangent preconditioner. If max_smooth_iter is reached before the target smoothing convergence is reached, the solver simply switches to using the full tangent preconditioner.

This feature can also be used in cases where the nodal preconditioner may be effective enough to achieve convergence with relatively few iterations, but occasionally the nodal preconditioner is not effective. If max_smooth_iter is set to a relatively high number, many load steps can be solved without ever using the full tangent preconditioner, but this preconditioner is used on the more difficult steps that require more iterations.

Another use of the smoothing iteration control commands is for switching between the nodal preconditioner and the full tangent preconditioner for certain time periods. These command lines, like all the other command lines discussed above, can be set on a period-specific basis by appending them with the DURING keyword, followed by a list of periods. To use the nodal preconditioner on some periods, the user can set max_smooth_iter to a high number for those periods and leave the default value of 0 for the periods in which the full tangent preconditioner is desired.

4.3.4 Fall-Back Strategy Commands

```
STAGNATION THRESHOLD = <real>stagnation(1.0e-12)
MINIMUM CONVERGENCE RATE = <real>min_conv_rate(1.0e-4)
ADAPTIVE STRATEGY = SWITCH|UPDATE(SWITCH)
```

Occasionally, the full tangent preconditioner may stagnate, failing to significantly reduce the residual from one CG iteration to the next. At each iteration, Sierra/SM checks for slow convergence, and attempts to remedy poor convergence if such is detected. The commands listed above can optionally be added to the FULL TANGENT PRECONDITIONER command block.

The STAGNATION THRESHOLD and MINIMUM CONVERGENCE RATE command lines are used to set the stagnation and min_conv_rate parameters, respectively. These parameters are used in a strategy that attempts to remedy slow convergence. The method for remedying slow convergence is controlled with the ADAPTIVE STRATEGY command line, which can have a value of SWITCH or UPDATE. The default strategy is SWITCH. The convergence rate is computed as the absolute value of the relative difference between residuals after successive CG iterations. In the SWITCH strategy, if the convergence rate is less than the value of min_conv_rate but greater than the value of stagnation, the CG solver will switch to using the fall-back nodal preconditioner. In

the `UPDATE` strategy, the full tangent preconditioner would be updated instead. In either case, if the convergence rate is below the value of `stagnation`, the solver will switch to the nodal preconditioner.

4.4 FETI Equation Solver

FETI is a domain-decomposition-based parallel iterative linear solver that can be used to compute the action of the full tangent preconditioner for the nonlinear CG solver [3, 4]. FETI uses a direct solver on each domain and iteratively solves for Lagrange multiplier fields at the domain boundaries. Under typical usage, the FETI domains correspond to the portions of the model owned by each processor. If a model is run on a single processor, FETI simply behaves as a direct solver. Because large models are typically run using many processors, FETI uses significantly lower computing resources than a direct solution of the whole problem would require because a large number of small direct solutions are performed in parallel.

Although a number of other linear solvers are available for use with the full tangent preconditioner in the CG solver, it is recommended that FETI be used. FETI is actively maintained and tested by the Sierra/SM development team; its effectiveness as a robust parallel solver has been demonstrated on a wide range of production analyses. The command block for the FETI equation solver is as follows:

```
BEGIN FETI EQUATION SOLVER <string>name
  #
  # convergence commands
  MAXIMUM ITERATIONS = <integer>max_iter(500)
  RESIDUAL NORM TOLERANCE = <real>resid_tol(1.0e-6)
  #
  # diagnostic commands, off by default.
  # solver turns on print statements from FETI
  # matrix dumps the matrix to a matlab file (in serial)
  PARAM-STRING "debugMask" VALUE <string>"solver"|"matrix"
  #
  # memory usage commands
  PARAM-STRING "precision" VALUE <string>"single"|"double"
    ("double")
  PRECONDITIONING METHOD = NONE|LUMPED|DIRICHLET(DIRICHLET)
  MAXIMUM ORTHOGONALIZATION = <integer>max_orthog(500)
  #
  # solver commands
  LOCAL SOLVER  = SKYLINE|SPARSE|ITERATIVE(SPARSE)
  COARSE SOLVER = SKYLINE|SPARSE|ITERATIVE(SPARSE)
  #
  # This command is only used in conjunction with
  # local solver = iterative.
  NUM LOCAL SUBDOMAINS = <integer>num_local_subdomains
END [FETI EQUATION SOLVER <string>name]
```

The command lines used to control FETI all reside in the `FETI EQUATION SOLVER` command block, where `name` identifies the particular command block. This command block, as with all equation solver command blocks, is placed in the SIERRA scope and referenced by `name` when it

is used. Although a number of command lines are available to control the behavior of FETI, the default settings generally work well for the vast majority of problems. Thus, it is recommended that all default settings be used unless special behavior is desired because of the unique features of a specific model. The command lines in the FETI EQUATION SOLVER command block are described in Sections 4.4.1 through 4.4.3.

4.4.1 Convergence Commands

```
MAXIMUM ITERATIONS = <integer>max_iter(500)
RESIDUAL NORM TOLERANCE = <real>resid_tol(1.0e-6)
```

The command lines listed above provide controls on the convergence of the FETI iterative solver, and belong in the FETI EQUATION SOLVER command block.

The MAXIMUM ITERATIONS command line sets the maximum number of iterations allowed per FETI solution. The default value of the parameter max_iter is 500. A FETI solution occurs for every CG iteration in which FETI is used with the full tangent preconditioner. The RESIDUAL NORM TOLERANCE command line sets the convergence criterion for the FETI solver. The default value of the parameter resid_tol is 1.0e-6. If convergence is not reached before the iteration count exceeds max_iter, FETI will simply return the current gradient direction to the CG solver, which will continue iterating. The code will not exit with an error. The default settings for both of these command lines are reasonable for most models and typically should not be modified.

4.4.2 Memory Usage Commands

```
PARAM-STRING "precision" VALUE <string>"single"|"double"
  ("double")
PRECONDITIONING METHOD = NONE|LUMPED|DIRICHLET(DIRICHLET)
MAXIMUM ORTHOGONALIZATION = <integer>max_orthog(500)
```

The command lines listed above can be placed in the FETI EQUATION SOLVER command block to enable optional memory-saving features of the FETI solver. All these features will adversely affect the performance of this solver to some degree, but they can be useful if the memory requirements of a model exceed the capacity of the machine on which the model is run. Before using these features, it is important to consider that on a distributed memory cluster, spreading the model out over more processors can reduce the memory requirements on each processor. For this reason, it is often better to use more processors rather than use the options described here.

FETI has the option of using either single or double precision for storage of internal variables. The default behavior is to use double precision, and this is typically recommended. To select single precision, the user would specify the command line as follows: PARAM-STRING "precision" VALUE "single". Using single-precision variables within FETI can dramatically reduce the memory requirements. This may, however, slightly degrade the performance of the solver, requiring more iterations within FETI or more CG iterations. Using single precision in FETI does not

affect the Sierra/SM data structures, which are always double precision, and therefore does not adversely affect solution accuracy.

The PRECONDITIONING METHOD command line selects the preconditioning method that is used internally within FETI. The default option, DIRICHLET, typically results in the best convergence rate. The LUMPED option uses less memory, but it usually results in more iterations within FETI. This option should only be used if there are constraints on memory usage. The NONE option uses no preconditioner and is included only for completeness. This option is not of practical interest.

Like the CG solver, the FETI equation solver stores a set of search directions to ensure that the search direction used in each iteration is orthogonal to previous search directions. The number of search directions stored is controllable with the MAXIMUM ORTHOGONALIZATION command line. The default value of 500 for max_orthog provides optimal convergence. Setting max_orthog to a lower number can decrease memory usage but, as with the other options discussed above, may require more iterations for convergence.

4.4.3 Solver Commands

```
LOCAL SOLVER  = SKYLINE|SPARSE|ITERATIVE(SPARSE)
COARSE SOLVER = SKYLINE|SPARSE|ITERATIVE(SPARSE)
NUM LOCAL SUBDOMAINS = <integer>num_local_subdomains
```

The command lines listed above control the type of solver used by FETI for solving the linear system that arises from the coarse grid and for the local linear system on the actual solution mesh. The default behavior is for FETI to use a sparse direct solver for both of these systems, an approach that works well for most problems. The LOCAL SOLVER command line is used to select the solver for the local subdomains. Similarly, the solver for the coarse grid can be selected with the COARSE SOLVER command lines. Both of these command lines allow the same three options: SPARSE, SKYLINE, and ITERATIVE, where SPARSE is the default.

- The SPARSE option uses a sparse matrix storage direct solver. The sparse matrix is factored into an $A = LDL^T$ decomposition, but the code implementation takes advantage of equation orderings to reduce matrix fill-in. The default equation ordering is done by calling METIS's sparse matrix ordering algorithm. This solver is recommended for both speed and memory efficiency.

- The SKYLINE option uses a skyline (profile) matrix storage direct solver. The skyline matrix is factored into an $A = LDL^T$ decomposition. This method is very robust, detects rigid-body modes effectively, and includes the reverse Cuthill-McKee (RCM) and Sloan equation orderings. This solver uses much more memory than does the sparse solver.

- The ITERATIVE option implements a multiple domain FETI algorithm for the local solver or the coarse solver. The the number of domains for the local solver is either computed based on internal heuristics or input by the user. The number of domains for the coarse solver is defaulted to 2 which means that the iterative coarse solver will run on 2 processors. This

option is recommended in special circumstances where memory becomes an issue with the sparse direct solver.

The `NUM LOCAL SUBDOMAINS` command line sets the number of local subdomains for the `ITERATIVE` local solver. This command line is not used for the other local solvers. As the number of local subdomains increases, the size of the local subdomains decreases, thus reducing the memory requirements and the time it takes for the local matrix factorizations. As the number of subdomains increases, the size of the coarse grid increases, thus requiring increased factorization time and memory. A good rule of thumb is to set the number of local subdomains to the total number of elements in the mesh divided by 400. For a serial run, if the `NUM LOCAL SUBDOMAINS` command line is not specified, FETI sets the number of domains to the greater of 2 or the number of elements divided by 400. For parallel runs, FETI uses the same number of domains as that used by Sierra/SM.

4.5 Control Contact

The multilevel solution control scheme used for contact is referred to as control contact. After a set of nodes in contact (a constraint set) is established, a model problem is solved using the CG solver with this constraint set held constant. For frictional contact, slave nodes are fixed to master faces during the CG iterations. For frictionless contact, the slave nodes are fixed in the normal direction but are allowed to slide during the CG iterations. After model problem convergence, the constraint set is updated to reflect the changing contact conditions. This update gives rise to a force imbalance and to another model problem.

Changing or updating the constraint set is referred to as a contact update. Multiple contact updates are typically required before equilibrium is achieved. The contact update consists of a search, a gap removal, an equilibrium query, and a slip calculation if equilibrium is not satisfied.

New constraints are detected in the search phase based on the current deformed configuration of the model. If a node is found to penetrate a master surface, the node is added to the set of constraints. The slave node is moved to the master surface by moving it along the push-back vector. The penetration may be removed at once (the default behavior), or it may be incrementally removed over a number of model problems.

The gap removal phase involves creating or destroying constraints and calculating push-back vectors for slave nodes. For types of surface mechanics in which contacting surfaces are free to separate, a slave node constraint continues to exist as long as there is a compressive force between the slave node and the master surface. The constraint changes during the gap removal phase to reflect changes in the shape and orientation of the master surface. Constraints are destroyed when a tensile force exceeding a known tolerance exists at the master/slave interface.

During the slip portion of the contact update, a residual force is calculated at each node and resolved into normal and tangential components. This force reflects changes in residual due to gap removal that occurred earlier in the update. For frictional contact, the friction coefficient is used to determine the tangential load capacity. If the tangential loads exceed this capacity, nodes will slip.

The procedure for control contact is illustrated in the following sequence of figures. It is assumed that control contact is on level 1, so the model problems are directly solved by the core solver. In this example (Figure 4.2), the model problem has three constraints that are enforced during iterations of the core solver. Nothing prevents other nodes from penetrating surfaces during solution of the model problem, and in this example, one node does penetrate.

Once the model problem has converged, a contact update is performed. In the gap removal portion of the update (Figure 4.3), one of the constraints accumulated a tensile force large enough that surfaces should separate, thus eliminating this constraint. During the search, one node penetrated the surface, resulting in a new constraint for this node. A push-back vector is calculated for this node to remove the penetration.

In the second phase of the contact update, slip is allowed to occur along the master/slave interface. After the gaps are removed, the external and internal force vectors are recomputed, and forces are partitioned along normal and tangential directions of the master surface. This process is illustrated in Figure 4.4.

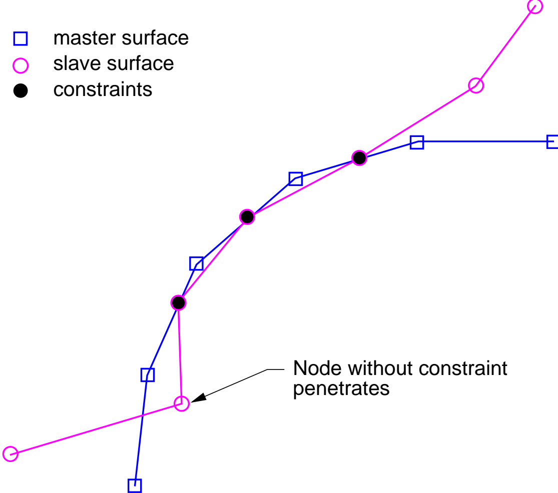


Figure 4.2: Contact configuration at the beginning of the contact update.

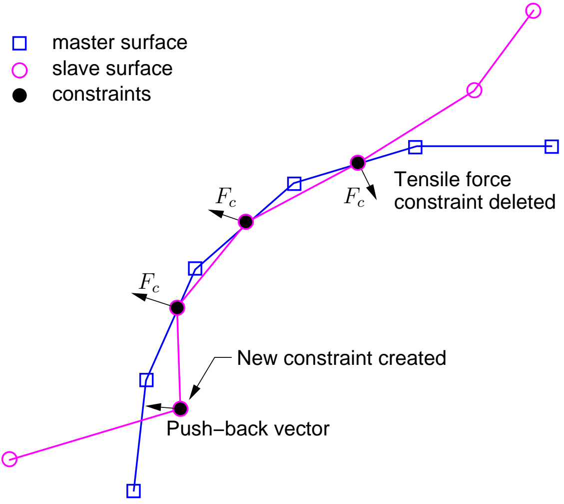


Figure 4.3: Contact gap removal (after contact search).

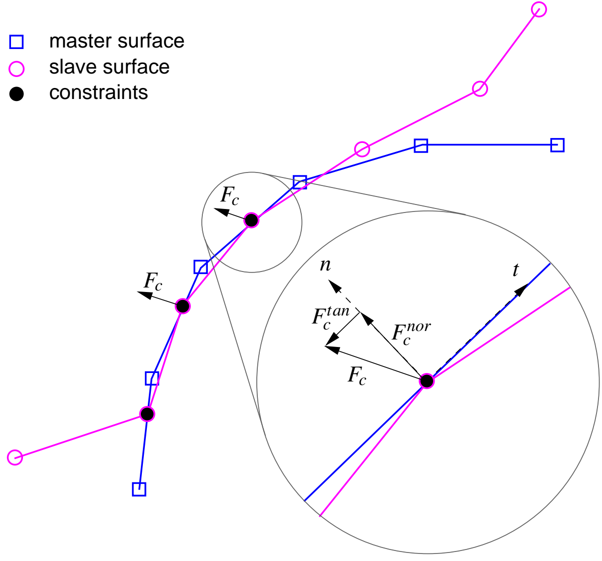


Figure 4.4: Contact slip calculations.

The command block for control contact is as follows:

```
BEGIN CONTROL CONTACT
  #
  # Convergence commands
  TARGET RESIDUAL = <real>target_resid
    [DURING <string list>period_names]
  TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-3)
    [DURING <string list>period_names]
  TARGET RELATIVE CONTACT RESIDUAL = <real>target_rel_cont_resid
    [DURING <string list>period_names]
  ACCEPTABLE RESIDUAL = <real>accept_resid
    [DURING <string list>period_names]
  ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid(1.0e-2)
    [DURING <string list>period_names]
  ACCEPTABLE RELATIVE CONTACT RESIDUAL =
    <real>accept_rel_cont_resid(1.0e-1) [DURING <string list>period_names]
  REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
    [DURING <string list>period_names]
  MINIMUM ITERATIONS  = <integer>min_iter(0)
    [DURING <string list>period_names]
  MAXIMUM ITERATIONS  = <integer>max_iter(100)
    [DURING <string list>period_names]
  #
  # Level selection command
  LEVEL = <integer>contact_level(1)
  #
  # Diagnostic output commands, off by default.
```

```
  ITERATION PLOT = <integer>iter_plot
    [DURING <string list>period_names]
  ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
  #
  # Augmented Lagrange enforcement commands used with
  # enforcement = al defined in the contact definition
  #
  # Adaptive Penalty Options
  LAGRANGE ADAPTIVE PENALTY = OFF|SEPARATE|UNIFORM(OFF)
    [DURING <string list>period_names]
  LAGRANGE ADAPTIVE PENALTY GROWTH FACTOR = <real>(2.0)
    [DURING <string list>period_names]
  LAGRANGE ADAPTIVE PENALTY REDUCTION FACTOR = <real>(0.5)
    [DURING <string list>period_names]
  LAGRANGE ADAPTIVE PENALTY THRESHOLD = <real>(0.25)
    [DURING <string list>period_names]
  LAGRANGE MAXIMUM PENALTY MULTIPLIER = <real>(100.0)
    [DURING <string list>period_names]
  #
  # Tolerance Options
  LAGRANGE TARGET GAP = <real>(unused)
    [DURING <string list>period_names]
  LAGRANGE TARGET RELATIVE GAP = <real>(unused)
    [DURING <string list>period_names]
  LAGRANGE TOLERANCE = <real>(0.0)
    [DURING <string list>period_names]
  #
  # Miscellaneous Options
  LAGRANGE FLOATING CONSTRAINT ITERATIONS = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE INITIALIZE = BOTH|MULTIPLIER|NONE|PENALTY(MULTIPLIER)
    [DURING <string list>period_names]
  LAGRANGE LIMIT UPDATE = OFF|ON(OFF)
    [DURING <string list>period_names]
  LAGRANGE MAXIMUM UPDATES = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE SEARCH UPDATE = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE NODAL STIFFNESS MULTIPLIER = <real>(0.0)
    [DURING <string list>period_names]
END [CONTROL CONTACT]
```

To enable control contact, a CONTROL CONTACT command block must exist in the SOLVER command block. The line commands within the CONTROL CONTACT command block are used to control convergence during the contact updates, select the level for the contact control within the multilevel solver, and output diagnostic information. These commands are described in detail in Section 4.5.1 through Section 4.5.3.

4.5.1 Convergence Commands

The command lines listed in this section are placed in the CONTROL CONTACT command block to control convergence criteria for contact within the multilevel solver.

```
TARGET RESIDUAL = <real>target_resid
  [DURING <string list>period_names]
TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-3)
  [DURING <string list>period_names]
TARGET RELATIVE CONTACT RESIDUAL = <real>target_rel_cont_resid(1.0e-3)
  [DURING <string list>period_names]
ACCEPTABLE RESIDUAL = <real>accept_resid
  [DURING <string list>period_names]
ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid(1.0e-2)
  [DURING <string list>period_names]
ACCEPTABLE RELATIVE CONTACT RESIDUAL =
  <real>accept_rel_cont_resid(1.0e-1)[DURING <string list>period_names]
REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
  [DURING <string list>period_names]
MINIMUM ITERATIONS = <integer>min_iter(0)
  [DURING <string list>period_names]
MAXIMUM ITERATIONS = <integer>max_iter(100)
  [DURING <string list>period_names]
```

Solver convergence is monitored for control contact in much the same way as it is done for the core CG solver. The command lines listed above are used for specifying the convergence criteria used in the level 1 updates. With the exception of two additional command lines used for controlling the residual due to contact, these command lines are the same as those used for the core CG solver (described in Section 4.2.1) and have the same meaning. Here, however, these command lines are applied to the contact control. Note that all of these command lines can be appended with the DURING specification, as discussed in Section 4.2.1.

Contact convergence is measured by computing the L^2 norm of the residual, and comparing that residual norm with target convergence criteria specified by the user. There are two residual norms used as convergence metrics for contact: the norm of the residual on all nodes, and the norm of the residual on the nodes currently in contact. In the discussion here, the residual norm for all nodes is referred to as the residual, while the residual norm for the contact nodes is referred to as the contact residual. Convergence can be monitored directly in terms of the residual norm, the relative residual, or the relative contact residual. The relative residual and relative contact residual are computed by dividing the residual and contact residual, respectively, by a reference quantity that is indicative of the current loading conditions on a model. Either the residual, the relative residual, or both can be used for checking convergence at the same time. In addition, the contact relative residual must be below specified convergence limits.

The TARGET RESIDUAL command line specifies the target convergence criterion in terms of the actual residual norm. The TARGET RELATIVE RESIDUAL command line specifies a convergence criterion in terms of the relative residual. If both command lines are specified, the multilevel

solver will accept the contact solution as converged if either the target residual or the target relative residual is below the specified values. The TARGET RELATIVE CONTACT RESIDUAL command line specifies the target convergence criterion for the relative contact residual. This criterion must be satisfied in addition to the target residual or relative residual for convergence. The default value for the target relative contact residual is the target relative residual.

The multilevel solver also allows for acceptable convergence criteria to be input for contact convergence. If the solution has not converged to the specified targets before the maximum number of iterations is reached, the residual is checked against the acceptable convergence criteria. These criteria are specified via the ACCEPTABLE RESIDUAL, ACCEPTABLE RELATIVE RESIDUAL, and ACCEPTABLE RELATIVE CONTACT RESIDUAL command lines. The concepts of residual, relative residual, and relative contact residual are the same as those used for the target limits. If the solution has not met the target criteria but has met the acceptable criteria, the solution is allowed to proceed. The defaults for each of these acceptable criteria are 10 times the corresponding target criteria.

If relative residuals are given in the convergence criteria, the REFERENCE command line can be used to select the method for computing the reference load. This command line has several options: EXTERNAL, INTERNAL, BELYTSCHKO, RESIDUAL and ENERGY. When the EXTERNAL option, which is the default, is selected, the reference load is computed by taking the L^2 norm of the current external load. The INTERNAL option uses the norm of the internal forces as the reference load. The BELYTSCHKO option uses the maximum of the norm of the external load, the norm of the reactions, and the norm of the internal forces. The RESIDUAL option denotes that the residual from the initial residual should be used as the reference quantity. The ENERGY residual is the energy conjugate norm of the residual and reference vectors. These criteria are further explained in Section 4.2.1.

The MAXIMUM ITERATIONS and MINIMUM ITERATIONS command lines specify the maximum and minimum number of contact updates, respectively. The default minimum number of iterations, min_iter, is 0. If a number greater than 0 is specified, the multilevel solver will update contact at least that many times, regardless of whether the convergence criteria have been met.

4.5.2 Level Selection Command

```
LEVEL = <integer>contact_level(1)
```

The LEVEL command line in the CONTROL CONTACT command block is used to specify the level in the multilevel solver at which contact is controlled. The contact_level parameter can have a value of either 1 or 2, with 1 being the default.

This command is used when multiple controls (i.e. control contact and control stiffness) are active in the multilevel solver. It is permissible for multiple controls to exist at a given level. The default behavior is for all controls to be at level 1. To have a control at level 2, another control must be active at level 1 because a level in the multilevel solver cannot be skipped. The level of other controls is specified using the LEVEL command line in the command blocks associated with those controls.

4.5.3 Diagnostic Output Commands

```
ITERATION PLOT = <integer>iter_plot
  [DURING <string list>period_names]
ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
```

The command lines listed above can be used to control the output of diagnostic information from the contact control within the multilevel solver. The ITERATION PLOT command line allows plots of the current state of the model to be written to the output database during the contact updates. The value supplied in iter_plot specifies the frequency of such plots. The default behavior is not to produce such plots. The plots are generated with a fictitious time step. Every time such a plot is generated, a message is sent to the log file. This feature is useful for debugging, but it should be used with care. Note that the ITERATION PLOT command line can be appended with the DURING specification, as discussed in Section 4.2.1.

The output produced by iteration plots is sent by default to every active results output file, as specified by a RESULTS OUTPUT command block. The ITERATION PLOT OUTPUT BLOCKS command line can optionally be used to supply a list of output block names to which iteration plots should be written. Each of the output block names listed in plot_blocks, must match the name of a RESULTS OUTPUT command block (see Section 9.2.1). The ITERATION PLOT OUTPUT BLOCKS command line can be useful for directing iteration plots to separate output files so that these plots are separated from the converged solution output. It can also be used to send output plots for different levels of the multilevel solver to different files.

4.5.4 Augmented Lagrange Enforcement Commands

```
# Adaptive Penalty Options
LAGRANGE ADAPTIVE PENALTY = OFF|SEPARATE|UNIFORM(OFF)
  [DURING <string list>period_names]
LAGRANGE ADAPTIVE PENALTY GROWTH FACTOR = <real>(2.0)
  [DURING <string list>period_names]
LAGRANGE ADAPTIVE PENALTY REDUCTION FACTOR = <real>(0.5)
  [DURING <string list>period_names]
LAGRANGE ADAPTIVE PENALTY THRESHOLD = <real>(0.25)
  [DURING <string list>period_names]
LAGRANGE MAXIMUM PENALTY MULTIPLIER = <real>(100.0)
  [DURING <string list>period_names]
#
# Tolerance Options
LAGRANGE TARGET GAP = <real>(unused)
  [DURING <string list>period_names]
LAGRANGE TARGET RELATIVE GAP = <real>(unused)
  [DURING <string list>period_names]
LAGRANGE TOLERANCE = <real>(0.0)
  [DURING <string list>period_names]
#
```

```
# Miscellaneous Options
LAGRANGE FLOATING CONSTRAINT ITERATIONS = <integer>(infinity)
  [DURING <string list>period_names]
LAGRANGE INITIALIZE = BOTH|MULTIPLIER|NONE|PENALTY(MULTIPLIER)
  [DURING <string list>period_names]
LAGRANGE LIMIT UPDATE = OFF|ON(OFF)
  [DURING <string list>period_names]
LAGRANGE MAXIMUM UPDATES = <integer>(infinity)
  [DURING <string list>period_names]
LAGRANGE SEARCH UPDATE = <integer>(infinity)
  [DURING <string list>period_names]
LAGRANGE NODAL STIFFNESS MULTIPLIER = <real>(0.0)
  [DURING <string list>period_names]
```

The command lines listed above can be used to control augmented Lagrange contact enforcement, which is activated when `enforcement = al` is set in the contact definition block.

4.5.4.1 Augmented Lagrange Adaptive Penalty Commands

The `LAGRANGE ADAPTIVE PENALTY` command indicates how the penalty multiplier will be adaptively adjusted as the problem progresses. The option `SEPARATE` scales the penalty on an interaction by interaction basis. The option `UNIFORM` scales all interactions equally. The default option is `OFF`.

`LAGRANGE ADAPTIVE PENALTY GROWTH FACTOR` command sets the adaptive penalty growth factor. `LAGRANGE ADAPTIVE PENALTY REDUCTION FACTOR` command sets the adaptive penalty reduction factor. The `LAGRANGE ADAPTIVE PENALTY THRESHOLD` sets the adaptive penalty threshold. The `LAGRANGE MAXIMUM PENALTY MULTIPLIER` sets the maximum penalty multiplier, which is the upper bound on the penalty multiplier when running augmented Lagrange contact.

The penalty multiplier adaptation is as follows. If the current gap magnitude is greater than the previous gap magnitude, the penalty multiplier is reduced by multiplying it by the penalty reduction factor. If the current gap magnitude is in between the previous gap magnitude and previous gap magnitude times the penalty threshold the penalty multiplier is increased by multiplying it by the penalty growth factor. The lower bound for the penalty multiplier is one and the upper bound is the maximum penalty multiplier.

4.5.4.2 Augmented Lagrange Tolerance and Convergence Options

The `LAGRANGE TARGET GAP` sets the absolute target gap. If the target gap is not reached the contact iteration will not converge. The default sets no requirement on the absolute gap. When the target gap is reached no Lagrange multiplier update is performed. The `LAGRANGE TARGET RELATIVE GAP` sets the relative target gap. The relative target gap functions similar to the target gap.

The LAGRANGE TOLERANCE sets the the Lagrange multiplier tolerance for updating. Once the change to the Lagrange multiplier is less than the tolerance, the Lagrange multiplier will not be updated. The default is 0.0 which always updates the Lagrange multiplier.

4.5.4.3 Augmented Lagrange Miscellaneous Options

The LAGRANGE FLOATING CONSTRAINT ITERATIONS sets the number of model problems to perform before locking (fixing) the set of contact constraints. The default is to never lock the set of constraints. The contact interaction history decides whether the interaction will be CAPTURED or RELEASED. Contact interactions that are CAPTURED remain CAPTURED, interactions that are RELEASED remain RELEASED, and interactions that are ACTIVE become CAPTURED if they are penetrating more than they are gapping and RELEASED if they are gapping more than penetrating.

The LAGRANGE INITIALIZE command sets the algorithm used to initialize the Lagrange multiplier and penalty multiplier. The option NONE does not initialize the Lagrange multiplier. The option MULTIPLIER initializes the Lagrange multiplier from the previous contact force. The option PENALTY attempts to scale the penalty based off of the previous contact force and current gap. The option BOTH initializes both the penalty scale and Lagrange multiplier from the previous contact force.

The LAGRANGE LIMIT UPDATE command indicates whether to limit the updating of the Lagrange multiplier to the previous magnitude of the Lagrange multiplier.

The LAGRANGE MAXIMUM UPDATES command sets the maximum number of augmented Lagrange updates performed during a loadstep. The default is to always update the Lagrange multiplier. Setting this to zero makes contact a pure penalty formulation.

The LAGRANGE SEARCH UPDATE sets the increment in model problem iterations between successive contact search updates. The default is to search once per loadstep.

The LAGRANGE NODAL STIFFNESS MULTIPLIER sets a scale factor on the amount of contact stiffness that gets included in the nodal preconditioner. The default value is zero and so no contact stiffness is present in the nodal preconditioner. A value of one would distribute the full contact stiffness to the contact nodes. A value much greater than one would effectively pin the contact nodes in the nodal preconditioner.

4.6 Control Stiffness

Control stiffness is an augmented Lagrange iterative solution strategy for solving models that have widely varying stiffnesses in various modes of material response. These differences in stiffness can be between individual materials in a problem or even between the different responses of a single material model. For instance, nearly incompressible materials have a bulk/volumetric response that is much stiffer than the corresponding shear/deviatoric response. A similar case arises for materials that are orthotropic or anisotropic in nature and exhibit widely varying stiffnesses in the principal material directions.

Using control stiffness allows part of the constitutive response of a material to be softened or stiffened to give a tangent response that results in model problems that can be solved more easily by the core solver. This is accomplished by scaling a chosen stress increment up or down in a given model problem and adding it to a saved reference stress to yield a scaled stress that is used for equilibrium evaluations. The reference stress accumulates the model problem stress increments so that the true material response for a time step is obtained after solving a sequence of model problems.

For the case of nearly incompressible materials, a sequence of model problems can be constructed in which the bulk behavior is softened (scaled down) and/or the shear behavior is stiffened (scaled up). For the case of anisotropic materials, the anisotropic part of the material response is scaled down to create a model problem where the material has a more nearly isotropic response. In addition, the isotropic part of the response of some of these orthotropic materials can have its bulk stress increments scaled down and/or its shear/deviatoric stress increments scaled up. Finally, if an isotropic material is much stiffer than its neighbors, all of that material's stress increments can be scaled down.

Scaling a stress increment up or down is akin to using scaled moduli in a model problem. It should be noted that this is only precisely true for the case in which the true material response is linear. For the case of nonlinear materials, the effective scaled moduli are based solely on the difference between the unscaled stresses and the reference stresses. As a result, these effective scaled moduli vary over the course of model problem and core solver iterations. Nevertheless, the control stiffness algorithms are completely general and do not rely on linearity in the true material response.

The degree to which components of a material's response are softened or stiffened to achieve overall minimum computational time is problem dependent and involves a trade-off between the difficulty of solving a model problem and the number of model problems that must be used to achieve the final solution.

For example, consider a nearly incompressible material, such as rubber, for which the model problem solution may be optimized by scaling the bulk and shear behaviors such that the effective tangent response has a bulk modulus that is equal to twice its shear modulus. Adjusting the ratio between the bulk and shear moduli in this manner may minimize the core solver iterations for a given model problem. However, because of the inherently large difference between these two moduli in this material, significant bulk and/or shear scaling would be required to achieve these optimal scaled moduli. Such severe scaling could result in a large number of model problems to achieve the true material response. Choosing less severe scalings (scalings closer to 1), on the other hand,

could result in more core solver iterations in each model problem, but fewer model problems to arrive at the true material response.

Experience has shown that once nearly optimal scaling values have been determined for a class of problems, these can be applied successfully to families of problems with similar characteristics.

A partial explanation of scaling the material response will be given for a simple scalar relation between stress and strain. For example, the true pressure-volume relation for a nearly incompressible material is a scalar relation between pressure and volumetric strain. Appropriate generalizations of the softening and stiffening algorithms to tensor relations are easily achieved by applying the same algorithms for all of the stress quantities individually. Only the calculation of the appropriate error measures is modified to deal with the tensorial nature of the stress-strain response that is being scaled.

There are two strain and three stress quantities of interest in the control stiffness algorithm. The stress quantities are the unscaled stress calculated by the unmodified material model, the scaled stress used for equilibrium evaluation, and a reference stress used to build up stresses to achieve convergence such that the scaled model problem stress is nearly equal to the unscaled stress computed from the material model. The strain quantities of interest are the true strain computed from the kinematics and the strain necessary to achieve the scaled stress using the unmodified material model.

In the equations that follow, the subscripts refer to the particular model problem being solved. Let us begin with the true material response in model problem I given by

$$\sigma_I = f(e_I), \tag{4.11}$$

where σ_I is the true stress corresponding to the kinematic strain e_I determined from the nodal displacements, and $f(\cdot)$ is the function representing the constitutive response. Although the true material response has been written for the hyperelastic case employing total stress and strain quantities, the control stiffness algorithms that are being presented can be equally applied to hypoelastic models where the stress rate is written in terms of the strain rate. The important point here is that the unscaled constitutive equation is used to calculate the unscaled/true stress response. It is not necessary for that relation to be linear or for a particular formulation to be used in terms of total strains or incremental strain rates. However, it should be noted that control stiffness is set up to work only with certain material models.

The scaled stress response to use for equilibrium evaluations in a model problem is determined as follows:

$$s_I = r_I + \lambda \left(\sigma_I - r_I\right), \tag{4.12}$$

where s_I is the scaled stress, r_I is the reference stress, and λ is the scaling factor. For softening behavior, λ is less than 1, while for stiffening behavior, it is greater than 1. As noted previously, this scaled stress is what the core solver uses for equilibrium evaluation in a given model problem. It should be noted that the core solver typically requires a number of iterations to find the scaled stress that results in equilibrium. The softening and stiffening control stiffness algorithms differ only in terms of what is used for the reference stress. The reference stress in model problem I is

given by the following:

$$r_I = \begin{cases} s_{I-1} & \lambda < 1 \quad \text{(softening)} \\ \sigma_{I-1} & \lambda > 1 \quad \text{(stiffening)} \end{cases} \tag{4.13}$$

Figures 4.5 and 4.6 provide a graphical presentation of softening the material response, and Figures 4.7 and 4.8 offer a similar presentation of stiffening the material response.

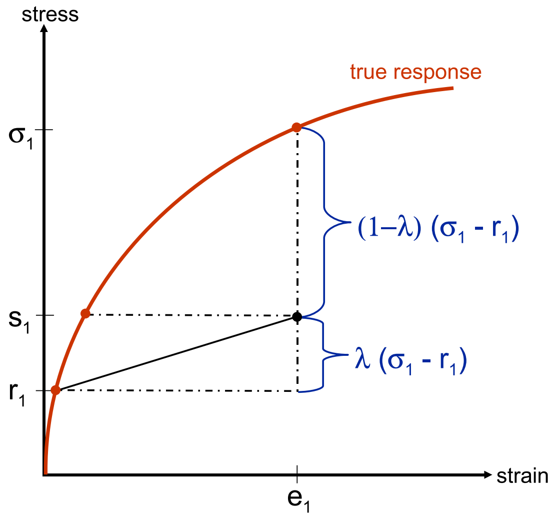


Figure 4.5: Control stiffness softening behavior in the first model problem of a time step.

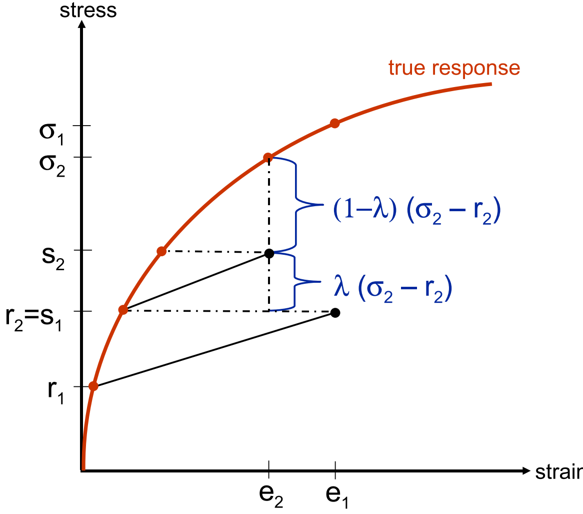


Figure 4.6: Control stiffness softening behavior in the second model problem of a time step.

When equilibrium is achieved in a model problem, it is necessary to do two things. First, an error measure related to the difference between the unscaled and scaled stress must be determined. Second, the reference stress used in the scaled stress calculations must be updated. The reference

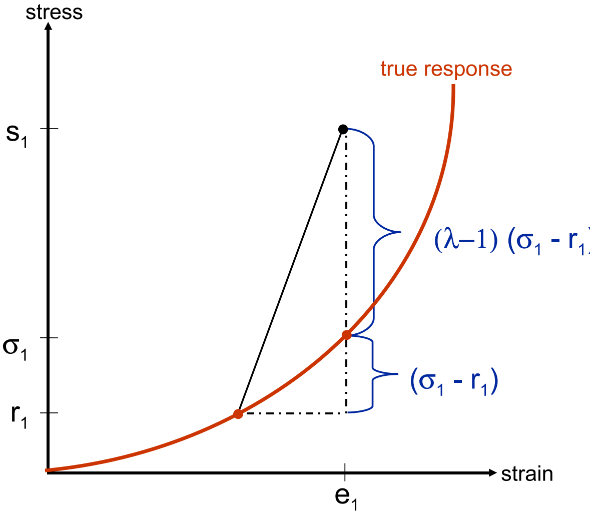


Figure 4.7: Control stiffness stiffening behavior in the first model problem of a time step.

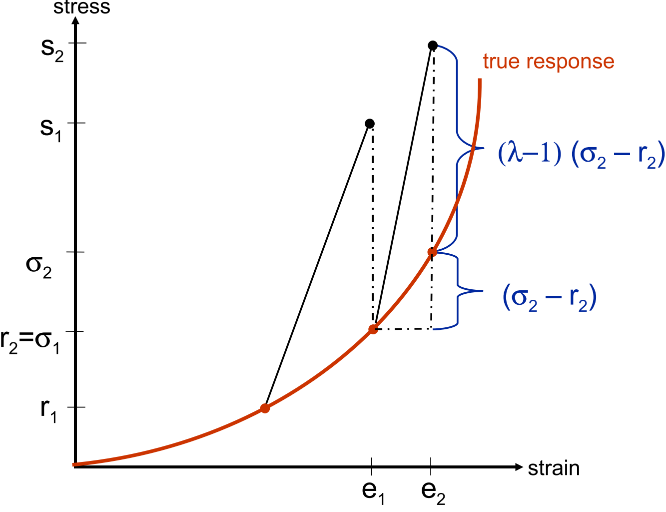


Figure 4.8: Control stiffness stiffening behavior in the second model problem of a time step.

stresses are updated according to Equation (4.13). That is, for the case of softening the material response, the reference stress is updated for the next model problem to be the scaled stress that achieved equilibrium in the current model problem. On the other hand, for the case of stiffening the material response, the reference stress is updated for the next model problem to be the true stress at the end of the current model problem.

The control stiffness updating process is converged when the scaled and unscaled stresses differ by an acceptably small amount. Several different error measures can be used to determine the degree to which the scaled and unscaled stresses differ. These can be categorized into two types: those that consider stress errors, and those that consider strain errors.

The stress convergence measures consider consider the difference between the scaled stress and the unscaled stress. The first of these stress error measures involves taking the L^2 norm of the stress difference:

$$error_I = \left\| s_I - \sigma_I \right\|_2 . \tag{4.14}$$

A second stress error measure involves normalizing that quantity:

$$error_I = \frac{\left\| s_I - \sigma_I \right\|_2}{\left\| s_I \right\|_2}. \tag{4.15}$$

The strain error convergence measure is based on a strain quantity E_I that is computed as the difference between the scaled stress and the unscaled stress and dividing that value by a modulus:

$$E_I = \frac{\left| s_I - \sigma_I \right|}{M}, \tag{4.16}$$

where M is an appropriate material modulus that is used to represent the unscaled constitutive response. For the case where the true material response is linear and the scaled stress is computed by stiffening the material response, E_I as defined by Equation (4.16) corresponds exactly to the difference between the kinematic strain e_I and the strain ϵ_I needed in the unscaled constitutive equation to give the scaled stress:

$$\epsilon_I = f^{-1}(s_I) = s_I / M, \tag{4.17}$$

where M would be the real material modulus. Nevertheless, Equation (4.16) is not dependent upon linear behavior, and it can be used in both softening and stiffening types of control stiffness scaling. The strain error quantity used to check convergence is computed by taking a global maximum (an L^∞ norm) as follows:

$$error_I = \left\| \frac{E_I}{E_{ref}} \right\|_\infty , \tag{4.18}$$

where E_{ref} is a reference strain specified in the input parameters for the material whose response is being scaled.

In summary, the user may specify error tolerances to determine convergence of the sequence of model problems solved in the control stiffness algorithm by using one of the error measures defined by Equations (4.14), (4.15), or (4.18). If the user so chooses, error tolerances corresponding to both Equations (4.14) and (4.15) can be specified, and convergence is considered whenever either tolerance is met. Because L^2 norms are computed by summing over all the elements in a mesh, users are cautioned that the tolerance used with Equation (4.14) to achieve a given level of control stiffness convergence is mesh dependent. That is, as the number of elements greatly increases, the L^2 norm of Equation (4.14) naturally increases. In most cases, the relative strain error given in Equation (4.18) is the preferred error measure. In addition to the error measures defined by Equations (4.14), (4.15), or (4.18), it is necessary to determine whether equilibrium is still achieved once the reference stresses have been updated. That is, equilibrium is re-evaluated with σ_{I+1} and s_{I+1} calculated without any additional changes to the fundamental nodal degrees of freedom such that

$$\sigma_{I+1} = \sigma_I \tag{4.19}$$

and

$$s_{I+1} = r_{I+1} + \lambda \left(\sigma_{I+1} - r_{I+1} \right) . \tag{4.20}$$

If convergence is achieved both for the difference between the scaled stress and the unscaled stress (either as a direct measure or as a strain error) and for the updated equilibrium evaluation, the time step is considered complete. Note that if control contact is also active, it is necessary that the appropriate contact convergence checks be satisfied as well.

Figure 4.9 shows an example of a linear material that has been softened through control stiffness. In this example, the time step is considered converged immediately after the second model problem update. The third model problem involved no iterations and is thus simply a re-evaluation of equilibrium and control stiffness convergence after the reference stress is updated following the second model problem.

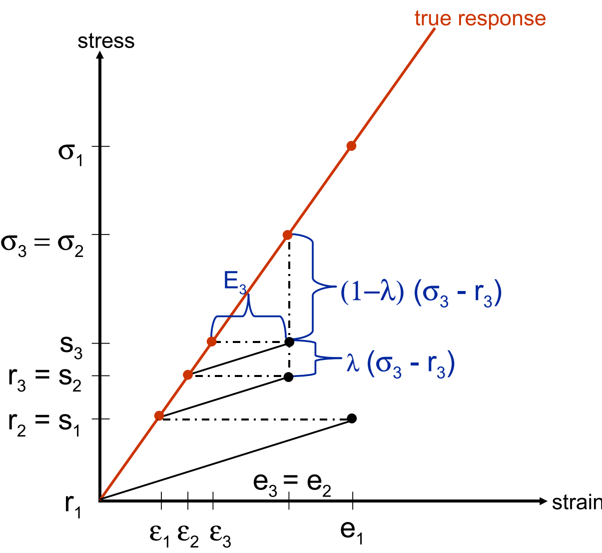


Figure 4.9: Control stiffness softening behavior convergence is achieved after solving two model problems.

The command block for control stiffness is as follows:

```
BEGIN CONTROL STIFFNESS [<string>stiffness_name]
  #
  # convergence commands
  TARGET <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
    = <real>target [DURING <string list>period_names]
  TARGET RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT|
    STRAIN INCREMENT
    = <real>target_rel [DURING <string list>period_names]
  ACCEPTABLE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
```

```
    = <real>accept [DURING <string list>period_names]
  ACCEPTABLE RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT|
    STRAIN INCREMENT
    = <real>accept_rel [DURING <string list>period_names]
  REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
    [DURING <string list>period_names]
  MINIMUM ITERATIONS  = <integer>min_iter(0)
    [DURING <string list>period_names]
  MAXIMUM ITERATIONS  = <integer>max_iter
    [DURING <string list>period_names]
  #
  # level selection command
  LEVEL = <integer>stiffness_level
  #
  # diagnostic output commands, off by default.
  ITERATION PLOT = <integer>iter_plot
    [DURING <string list>period_names]
  ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
END [CONTROL STIFFNESS <string>stiffness_name]
```

To enable control stiffness, a CONTROL STIFFNESS command block must exist in the SOLVER command block. The command lines within the CONTROL STIFFNESS command block are used to control convergence during the control stiffness updates, select the level for control stiffness within the multilevel solver, and output diagnostic information. These command lines are described in detail in Sections 4.6.1 through 4.6.3.

4.6.1 Convergence Commands

The command lines listed in this section are placed in the CONTROL STIFFNESS command block to control convergence criteria for stiffness control within the multilevel solver.

```
TARGET <string>RESIDUAL|AXIAL FORCE INCREMENT|
  PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
  = <real>target [DURING <string list>period_names]
TARGET RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
  PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT|
  STRAIN INCREMENT
  = <real>target_rel [DURING <string list>period_names]
ACCEPTABLE <string>RESIDUAL|AXIAL FORCE INCREMENT|
  PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
  = <real>accept [DURING <string list>period_names]
ACCEPTABLE RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
  PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT|
  STRAIN INCREMENT
  = <real>accept_rel [DURING <string list>period_names]
```

```
REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
  [DURING <string list>period_names]
MINIMUM ITERATIONS = <integer>min_iter(0)
  [DURING <string list>period_names]
MAXIMUM ITERATIONS = <integer>max_iter
  [DURING <string list>period_names]
```

Convergence for problems using control stiffness requires that two criteria be met. The first criterion is that equilibrium must be achieved. The second criterion is that the scaled stress used for equilibrium must be close enough to the unscaled stress that is determined with the unmodified material model as calculated from either Equation (4.14), (4.15), or (4.18).

The command lines listed above for the equilibrium evaluation are similar to those used for the core CG solver, and have similar meaning. Here, however, the command lines are applied as part of determining convergence for the control stiffness series of model problems. Note that some of the command lines have multiple options, e.g., the command line beginning with `TARGET` has five options. This practice has been used for the command lines beginning with `TARGET`, `TARGET RELATIVE`. `ACCEPTABLE`, and `ACCEPTABLE RELATIVE`. Note also that all of these command lines can be appended with the `DURING` specification, as discussed in Section 4.2.1.

Convergence of equilibrium using scaled stresses is measured by computing the L^2 norm of the residual, and comparing that residual norm with target convergence criteria specified by the user. Convergence can be monitored either directly in terms of the residual norm, or in terms of a relative residual, which is the residual norm divided by a reference quantity that is indicative of the current loading conditions on a model. Either the residual, the relative residual, or both of these can be used for checking convergence at the same time. The `TARGET RESIDUAL` command line specifies the target convergence criterion in terms of the actual residual norm. The `TARGET RELATIVE RESIDUAL` command line specifies a convergence criterion in terms of the relative residual. If both absolute and relative criteria are specified for the residual calculation, the equilibrium is accepted if either criterion is met.

The multilevel solver also allows acceptable convergence criteria to be input for residual convergence. If the solution has not converged to the specified targets before the maximum number of iterations is reached, the residual is checked against the acceptable convergence criteria. These criteria are specified via the `ACCEPTABLE RESIDUAL` and `ACCEPTABLE RELATIVE RESIDUAL` command lines. The concepts of absolute and relative values are the same here as discussed for the target limits.

If relative residuals are given in the convergence criteria, the `REFERENCE` command line can be used to select the method for computing the reference load. This command line has several options: `EXTERNAL`, `INTERNAL`, `BELYTSCHKO`, `RESIDUAL` and `ENERGY`. When the `EXTERNAL` option, which is the default, is selected, the reference load is computed by taking the L^2 norm of the current external load. The `INTERNAL` option uses the norm of the internal forces as the reference load. The `BELYTSCHKO` option uses the maximum of the norm of the external load, the norm of the reactions, and the norm of the internal forces. The `RESIDUAL` option denotes that the residual from the initial residual should be used as the reference quantity. The `ENERGY` residual is the energy conjugate norm of the residual and reference vectors. These criteria are further explained in Section 4.2.1.

For control stiffness, unlike other controls, the convergence of the series of model problems to obtain the final solution for a time step is also based on other criteria that measure how far apart the scaled and unscaled stress responses are from each other. Either a direct error measure of the difference of the scaled and unscaled stress is used or a relative strain error computed using the difference between the scaled and unscaled stress and a representative modulus is employed.

There are a number of variants of the TARGET command that can be used for the direct stress differences. The TARGET AXIAL FORCE INCREMENT command is used to base convergence on the increment of axial force in fiber membrane elements in an update. The TARGET PRESSURE INCREMENT and TARGET SDEV INCREMENT commands are used to specify convergence for nearly incompressible materials based on the pressure and deviatoric stress increments, respectively. Also, TARGET STRESS INCREMENT is used to base convergence on the stress increment for materials which have their entire behavior softened as part of a control stiffness approach. If desired, the RELATIVE form of these convergence criteria can be used. If the TARGET RELATIVE STRAIN INCREMENT command is specified, the strain error measure between the scaled and unscaled stress will be computed and used.

Any combination of these criteria can be specified. If more than one is specified, all of the criteria must be satisfied. However, each material block contributes either to the direct stress error measures or to the relative strain error measure. If a material block specifies a positive non-zero REFERENCE STRAIN in its definition, it will contribute to the relative strain error measure. Otherwise, it will contribute to the direct stress error measures. Finally, acceptable convergence criteria can be input for converging the difference between the scaled and unscaled stress. If the solution has not met the target criteria for equilibrium and the difference between the scaled and unscaled stresses, but meets the acceptable criteria, it is allowed to proceed to the next time step.

The MAXIMUM ITERATIONS and MINIMUM ITERATIONS command lines specify the maximum and minimum number of control stiffness updates, respectively. The default minimum number of iterations, min_iter, is 0. If a number greater than 0 is specified for min_iter, the multilevel solver will update stiffness at least that many times, regardless of whether the convergence criteria have been met.

4.6.2 Level Selection Command

```
LEVEL = <integer>stiffness_level
```

The LEVEL command line in the CONTROL STIFFNESS command block is used to specify the level in the multilevel solver at which stiffness is controlled. The stiffness_level parameter can have a value of either 1 or 2, with 1 being the default.

This command is used when multiple controls (i.e. control contact and control stiffness) are active in the multilevel solver. It is permissible for multiple controls to exist at a given level. The default behavior is for all controls to be at level 1. To have a control at level 2, another control must be active at level 1 because a level in the multilevel solver cannot be skipped. The level of other controls is specified using the LEVEL command line in the command blocks associated with those controls.

4.6.3 Diagnostic Output Commands

```
ITERATION PLOT = <integer>iter_plot
  [DURING <string list>period_names]
ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
```

The command lines listed above can be used to control the output of diagnostic information from the stiffness control within the multilevel solver. The ITERATION PLOT command line allows plots of the current state of the model to be written to the output database during the stiffness control updates. The value supplied in iter_plot specifies the frequency of such plots. The default behavior is not to produce such plots. The plots are generated with a fictitious time step. Every time such a plot is generated, a message is sent to the log file. This feature is useful for debugging, but it should be used with care. Note that the ITERATION PLOT command line can be appended with the DURING specification, as discussed in Section 4.2.1.

The output produced by iteration plots is sent by default to every active results output file, as specified by a RESULTS OUTPUT command block. The ITERATION PLOT OUTPUT BLOCKS command line can optionally be used to supply a list of output block names to which iteration plots should be written. Each of the output block names listed in plot_blocks, must match the name of a RESULTS OUTPUT command block (see Section 9.2.1). The ITERATION PLOT OUTPUT BLOCKS command line can be useful for directing iteration plots to separate output files so that these plots are separated from the converged solution output. It can also be used to send output plots for different levels of the multilevel solver to different files.

4.7 Control Failure

Control failure can be used to improve convergence in problems that involve material failure computed by the Multilinear Elastic-Plastic Failure material model or by user-defined global variables. To use element death, the user must define both an element death block (as defined in Section 6.5) and a multilevel solver with control failure.

The command block for control failure is as follows:

```
BEGIN CONTROL FAILURE [<string>failure_name]
  #
  #  convergence control command
  MAXIMUM ITERATIONS = <integer>max_iter
    [DURING <string list>period_names]
  #
  # level selection command
  LEVEL = <integer>failure_level
  #
  # diagnostic output commands, off by default.
  ITERATION PLOT = <integer>iter_plot
    [DURING <string list>period_names]
  ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
END [CONTROL FAILURE <string>failure_name]
```

To enable control failure, a CONTROL FAILURE command block must exist in the SOLVER command block. The command lines within the CONTROL FAILURE command block are used to establish convergence criteria, select the level for control failure within the multilevel solver, and output diagnostic information. These command lines are described in detail in Sections 4.7.1 through 4.7.3.

Convergence of control failure is defined by two criteria. The first criterion is that no new elements have been marked as ready to begin to fail in the previous model problem iteration. Currently this control only works with the Multilinear Elastic-Plastic Failure material model. An optional second criteria is that this first criteria is reached before the maximum number of iterations is reached if this command is specified by the user.

4.7.1 Convergence Command

The only user input convergence criteria that is allowed for control failure is specification of maximum iterations.

```
MAXIMUM ITERATIONS = <integer>max_iter
   [DURING <string list>period_names]
```

Iterations for control failure will continue until no new elements are failing unless the user specified number of maximum iterations is reached. If a maximum number of iterations is specified in the

input, control failure iterations will only continue until this number is reached. If the maximum number of iterations is reached the analysis will terminate in an unconverged state and with an explanatory message in the log file. Note that this command can be appended with the DURING specification, as discussed in Section 4.2.1.

4.7.2 Level Selection Command

```
LEVEL = <integer>failure_level
```

The LEVEL command line in the CONTROL FAILURE command block is used to specify the level in the multilevel solver at which failure is controlled. The failure_level parameter can have a value of either 1 or 2, with 1 being the default.

This command is used when multiple controls (i.e. control contact and control failure) are active in the multilevel solver. The control failure should be the only control at the outermost (highest numerical value) level. To have a control at level 2, another control must be active at level 1 because a level in the multilevel solver cannot be skipped. The level of other controls is specified using the LEVEL command line in the command blocks associated with those controls.

4.7.3 Diagnostic Output Commands

```
ITERATION PLOT = <integer>iter_plot
  [DURING <string list>period_names]
ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
```

The command lines listed above can be used to control the output of diagnostic information from the failure control within the multilevel solver. The ITERATION PLOT command line allows plots of the current state of the model to be written to the output database during the failure control updates. The value supplied in iter_plot specifies the frequency of such plots. The default behavior is not to produce such plots. The plots are generated with a fictitious time step. Every time such a plot is generated, a message is sent to the log file. This feature is useful for debugging, but it should be used with care. Note that the ITERATION PLOT command line can be appended with the DURING specification, as discussed in Section 4.2.1.

The output produced by iteration plots is sent by default to every active results output file, as specified by a RESULTS OUTPUT command block. The ITERATION PLOT OUTPUT BLOCKS command line can optionally be used to supply a list of output block names to which iteration plots should be written. Each of the output block names listed in plot_blocks, must match the name of a RESULTS OUTPUT command block (see Section 9.2.1). The ITERATION PLOT OUTPUT BLOCKS command line can be useful for directing iteration plots to separate output files so that these plots are separated from the converged solution output. It can also be used to send output plots for different levels of the multilevel solver to different files.

4.8 Control Modes

The core conjugate gradient solution algorithm typically works very well on "blocky" problems if the nodal preconditioners are used. As the aspect ratios of the structures modeled increase, however, the performance of the CG solver begins to degrade. Because of their lack of coupling terms, the nodal preconditioners only allow for the effects of the residual to be propagated across a single element during an iteration. Consequently, these preconditioners are effective at minimizing high frequency error, but may require many iterations to solve for the low frequency, structural bending modes in slender structures.

The full tangent preconditioners provided by Adagio greatly improve the effectiveness of the CG algorithm for solving models with slender members. However, the memory usage requirements and the computational effort needed to factorize matrices with the full tangent preconditioners can sometimes be significant.

In addition to its other solution strategies, Adagio provides a multigrid solution method within the CG algorithm. This method, known as control modes, greatly increases the effectiveness of the CG algorithm for solving slender, bending-dominated problems. In addition to the actual mesh of the model to be solved, referred to as the "reference mesh," control modes uses a coarse mesh of the model, known as a "constraint mesh," to solve for the low-frequency response. To use control modes, the user must supply both the reference mesh and the constraint mesh. While it does require some additional effort to generate a suitable coarse mesh, control modes provides very efficient solutions with only slightly higher memory usage than would be required by the standard CG solver with a nodal preconditioner.

Control modes functions somewhat like another level in the multilevel solver, although it does not appear as a control type in the `SOLVER` command block. The CG algorithm operates alternately on the residual on the coarse mesh and the fine mesh. The coarse residual is computed by performing a restriction operation from the fine mesh to the coarse mesh using the shape functions of the coarse elements. The CG solver is used to minimize the coarse residual until convergence is reached, at which point it switches to operate on the fine mesh to minimize that residual. The fine mesh iterations continue until either the fine mesh has converged or the fine residual is 50% of the coarse residual. The solver alternates between the fine and coarse meshes until convergence is obtained on both meshes.

To use control modes, the user should set up the mesh file and the input file as usual, except that the following additional items must be provided:

- A constraint mesh must be generated. The constraint mesh must be in a separate file from the reference mesh, which is the real model. The constraint mesh should be a coarse representation of the reference mesh. If there are node sets or side sets in the reference mesh that are used to prescribe kinematic boundary conditions, similar mesh entities should be provided in the coarse mesh to prescribe similar boundary conditions.

- A second `FINITE ELEMENT MODEL` command block must be provided in addition to the standard definition for the reference finite element model in the input file. This command block is set up exactly as it normally would be (see Section 6.1), except that the mesh file

referenced is the constraint mesh instead of the reference mesh. Although the constraint mesh is used purely as a solution tool, and does not use any finite elements or material models, each block in the constraint mesh must still be assigned a material model.

- A `CONTROL MODES REGION` command block must appear alongside the standard `ADAGIO REGION` command block within the `ADAGIO PROCEDURE` command block. The presence of the `CONTROL MODES REGION` command block instructs the CG solver to use the control modes logic. There are no commands within the CG solver for the region associated with the reference mesh related to control modes. The `CONTROL MODES REGION` command block is documented in Section 4.8.1. It contains the same commands used within the standard `ADAGIO REGION` command block, except that the commands in the `CONTROL MODES REGION` command block are used to control the control modes algorithm and the boundary conditions on the coarse mesh.

4.8.1 Control Modes Region

```
BEGIN CONTROL MODES REGION
  #
  # model setup
  USE FINITE ELEMENT MODEL <string>model_name
  CONTROL BLOCKS = <string list>control_blocks
  #
  # solver commands
  BEGIN SOLVER
    BEGIN LOADSTEP PREDICTOR
      #
      # Parameters for loadstep predictor
      #
    END [LOADSTEP PREDICTOR]
    BEGIN CG
      #
      # Parameters for CG
      #
    END [CG]
  END [SOLVER]
  JAS MODE [SOLVER|CONTACT|OUTPUT]
  #
  # kinematic boundary condition commands
  BEGIN FIXED DISPLACEMENT
    #
    # Parameters for fixed displacement
    #
  END [FIXED DISPLACEMENT]
  BEGIN PERIODIC
    #
    # Parameters for periodic
```

```
    #
  END [PERIODIC]
END [CONTROL MODES REGION]
```

The CONTROL MODES REGION command block controls the behavior of the control modes algorithm, and is placed alongside a standard ADAGIO REGION command block within the ADAGIO PROCEDURE scope. With the exception of the CONTROL BLOCKS command line, all the commands that can be used in this block are standard commands that appear in the Adagio region. These commands have the same meaning in either context; they simply apply to the constraint mesh or to the reference mesh, depending on the region block in which they appear. Sections 4.8.1.1 through 4.8.1.3 describe the components of the CONTROL MODES REGION command block.

4.8.1.1 Model Setup Commands

```
USE FINITE ELEMENT MODEL <string>model_name
CONTROL BLOCKS = <string list>control_blocks
```

The command lines listed above must appear in the CONTROL MODES REGION command block if control modes is used. The USE FINITE ELEMENT MODEL command line should reference the finite element model for the constraint mesh. This command line is used in the same way that the command line is used for the reference mesh (see Section 2.3).

The CONTROL BLOCKS command line provides a list of blocks in the reference mesh that will be controlled by the constraint mesh in the solver. The block names are listed using the standard method of referencing mesh entities (see Section 1.5). For example, the block with an ID of 1 would be listed as block_1 in this command. Multiple CONTROL BLOCKS command lines may appear to specify a long list of blocks over several lines.

4.8.1.2 Solver Commands

```
BEGIN SOLVER
  BEGIN LOADSTEP PREDICTOR
    #
    # Parameters for loadstep predictor
    #
  END [LOADSTEP PREDICTOR]
  BEGIN CG
    #
    # Parameters for CG
    #
  END [CG]
END SOLVER
JAS MODE [OUTPUT|CONTACT|SOLVER]
```

The constraint mesh must have solver parameters defined using the LOADSTEP PREDICTOR and CG command blocks, which must be nested within a SOLVER block in the CONTROL MODES

REGION. The constraint mesh does not have material properties or contact, so the CG command block is used to define the solver on the constraint mesh. The multilevel solver is not used on the constraint mesh, even though the multilevel solver may be used on the reference mesh in an analysis that uses a constraint mesh. All the command lines that appear in these command blocks in a standard analysis can be used for the parameters of the solver on the constraint mesh. Refer to Section 4.9.1 for a description of the command lines for the load step predictor and to Section 4.2 for a description of the CG solver commands.

The full set of preconditioners that are available on the reference mesh are also available for the constraint mesh. The full tangent preconditioner can be used on the constraint mesh, and provides a very powerful solution strategy. Because the constraint mesh is typically very small compared to the reference mesh, forming and factorizing matrices for the constraint mesh typically requires small computational resources.

The JAS MODE command line enables complete compatibility with the JAS3D legacy code. If this mode of operation is desired, the JAS MODE command line must be present in both the CONTROL MODES REGION command block and the ADAGIO REGION command block. See Section 4.10 for more information on this command line.

4.8.1.3 Kinematic Boundary Condition Commands

```
BEGIN FIXED DISPLACEMENT
  #
  # Parameters for fixed displacement
  #
END [FIXED DISPLACEMENT]
BEGIN PERIODIC
  #
  # Parameters for periodic
  #
END [PERIODIC]
```

For the solver to converge well, it is often important to create a node set or a side set on the coarse mesh that coincides geometrically with a node set or a side set on the fine mesh to which a fixed boundary condition is applied. This should be done for fixed boundary conditions, but not for boundary conditions with prescribed nonzero displacements. Any of the kinematic boundary condition command blocks that can appear in a standard Adagio region can also appear in the CONTROL MODES REGION command block. However, it is recommended that only FIXED DISPLACEMENT and PERIODIC boundary conditions be used on the constraint mesh.

The boundary conditions are applied to the specified mesh entities on the constraint mesh. The constraint mesh and the reference mesh can have a node set or a side set with the same identifier. The determination of which entity should be affected by the boundary condition is based on the context of the boundary condition specified. See Section 7.4.1 for details on the FIXED DISPLACEMENT command block.

4.9 Predictors

Predictors are an important component of Adagio's nonlinear solution strategy. A predictor is used to generate an initial trial solution for a load step or for a multilevel solver model problem. The goal of the predictor is to generate a trial solution that is as close as possible to the actual converged solution. A good prediction that gives a trial solution close to the actual solution can dramatically reduce the number of iterations required for convergence.

There are two types of predictors used in Adagio: the load step predictor and the level 1 predictor. The load step predictor, documented in Section 4.9.1, estimates a solution at the beginning of a new load step, and is applicable to all types of models. The level 1 predictor estimates the solution at the beginning of a new level 1 model problem in the multilevel solver. This predictor type, documented in Section 4.9.2, is only applicable for the multilevel solver.

4.9.1 Loadstep Predictor

```
BEGIN LOADSTEP PREDICTOR
  TYPE = <string>SCALE_FACTOR|SECANT|EXTERNAL|EXTERNAL_FIRST
    (SECANT)
  SCALE FACTOR = <real>scale_factor(1.0)
    [<real>first_scale_factor]
    [DURING <string list>period_names]
  SLIP SCALE FACTOR = <real>slip_factor(1.0)
    [DURING <string list>period_names]
END [LOADSTEP PREDICTOR]
```

The LOADSTEP PREDICTOR command block controls the behavior of the predictor that is used to predict the solution at the beginning of a new load step. This command block is placed in the SOLVER scope.

4.9.1.1 Predictor Type

```
TYPE = <string>SCALE_FACTOR|SECANT|EXTERNAL|EXTERNAL_FIRST
  (SECANT)
```

There are several types of load step predictors available in Adagio that are suitable for different types of analysis. The predictor type is selected using the TYPE command line in the LOADSTEP PREDICTOR command block.

- The scale factor predictor is selected with the SCALE_FACTOR option. This type of predictor extrapolates from the solution of the previous time step using the velocity field from that step, multiplied by a scale factor, to obtain a trial solution. The actual scale factor that is applied can be controlled with the SCALE FACTOR command line, which also appears in the LOADSTEP PREDICTOR command block. This type of predictor works well with models that have smoothly varying loads.

- The secant predictor is selected with the SECANT option, which is useful for scenarios in which the loading may not be monotonic. This type of predictor uses a line search to compute an optimal scaling factor for use in extrapolating from the previous solution. This type of predictor is the default, and provides good performance without requiring user intervention to select appropriate scale factors.

  A scale factor can optionally be specified for the SECANT predictor using the SCALE FACTOR command line. In this case, the user-specified scale factor is used instead of that computed by the line search for the time periods where a scale factor is defined. This permits switching between line search and scale factor predictors over the course of an analysis.

- The external predictor is selected with the EXTERNAL option. This type of predictor uses the solution from a file to predict the solution at new load steps. Re-using the work done in previous solutions of similar problems can be very helpful. Using this type of predictor requires that the input mesh file be a results file that contains displacement data.

- The EXTERNAL_FIRST option is used to select a special type of predictor that (1) uses the external predictor for the first load step of the solution period and (2) thereafter uses the scale factor predictor for every other load step in the solution period. If this option is chosen, the SCALE FACTOR command line must be included in the LOADSTEP PREDICTOR command block.

4.9.1.2 Scale Factor

```
SCALE FACTOR = <real>scale_factor(1.0)
  [<real>first_scale_factor]
  [DURING <string list>period_names]
```

The SCALE FACTOR command controls the behavior of the SCALE_FACTOR and SECANT predictor types (and the EXTERNAL_FIRST predictor type in all load steps except the first in a time period). The velocity field of the previous step, multiplied by the value of scale_factor, is used to extrapolate from the previous solution to obtain a trial solution. The default value of 1.0 provides good performance on models that experience smooth loading. Setting the value of scale_factor to 0.0 may improve the convergence of models with discontinuous loading.

The optional first_scale_factor parameter is used as the scale factor for the first step of a time period. If first_scale_factor is omitted, the default behavior is to use scale_factor for all time steps. Models are often subjected to loading that is mostly monotonic, but with discontinuities at the beginning of a new period. Using 1.0 and 0.0 for the values of scale_factor and first_scale_factor, respectively, often gives good performance in such cases.

The SCALE FACTOR command line can be appended with the DURING specification, as discussed in Section 4.2.1. This allows for the scale factor to be specified per time period.

4.9.1.3 Slip Scale Factor

```
SLIP SCALE FACTOR = <real>slip_factor(1.0)
```

```
[DURING <string list>period_names]
```

The prediction of slip on slave nodes that are currently active in sliding contact can be controlled separately from other nodes in the model by using the SLIP SCALE FACTOR command line. The parameter slip_factor controls the scaling of the predicted slip on these nodes. If slip_factor is left at its default value of 1.0, contact slip will be predicted using the same scale factor used for the rest of the model. If the value of slip_factor is set to 0.0, the predicted solution will not allow any sliding contact nodes to slip. This command line applies to both the SCALE_FACTOR and the SECANT types of predictors.

The SLIP SCALE FACTOR command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.9.2 Level Predictor

```
LEVEL 1 PREDICTOR = <string>NONE|DEFAULT(DEFAULT)
```

The LEVEL 1 PREDICTOR command line controls the level 1 predictor. This command line appears in the SOLVER command block, and is only applicable if the multilevel solver is used. The level 1 predictor estimates the solution for the next model problem based on the solution of the previous model problem. The default behavior is to enable this predictor, indicated by DEFAULT. The predictor can be turned off by setting it to NONE.

4.10 JAS3D Compatibility Mode

Adagio's multilevel solver and CG solver are based on the solver used in the legacy JAS3D code [1]. While the solvers in the two codes are very similar, there are some minor implementation differences between the two codes.

```
JAS MODE [SOLVER|CONTACT|OUTPUT]
```

Adagio provides an option, selectable by inserting the JAS MODE command line in the ADAGIO REGION command block, to make it use exactly the same algorithms as the JAS3D solver. If control modes is being used, the JAS MODE command line must appear both in the ADAGIO REGION and in the CONTROL MODES REGION command blocks (see Section 4.8.1.2). This option is primarily useful for migrating analyses previously done in JAS3D to Adagio.

The JAS MODE command used without any options enables all JAS3D compatibility features available. Optionally, this command can be followed by one of the three keywords SOLVER, CONTACT, or OUTPUT to enable a subset of these features. The command can be repeated on separate lines with different options to enable more than one subset of features.

The subsets of the features enabled by the JAS MODE command line are summarized below:

- SOLVER: Adagio's CG solver and multilevel solver use exactly the same algorithms. There are a number of algorithmic decisions in the solver that were made one way in JAS3D and another way in Adagio. While both methods are valid, the JAS MODE command line forces the code to always use the exact JAS3D algorithms.

- CONTACT: Adagio uses the Legacy Contact library instead of the internal Adagio and ACME code for contact search and enforcement. The Legacy Contact library is the same contact code that is used in JAS3D, so contact in Adagio models behaves the same as it would if it were run in JAS3D if this option is chosen.

- OUTPUT: The log file output produced by Adagio is formatted in the same manner as JAS3D formats log files, thus facilitating comparisons between the output from JAS3D and Adagio models.

Note that the JAS MODE command line only ensures JAS3D compatibility for features that are available in JAS3D. Advanced features such as the full tangent preconditioner are not available in JAS3D. It should also be noted that for those interested in converting JAS3D input files to Adagio input files, there is a program available for this purpose. For more information on this converter program, refer to the Adagio web page or contact an Adagio developer.

4.11 Time Step Control

Time stepping in Adagio is controlled by using a `TIME CONTROL` command block in the procedure scope. In this command block, the user controls the start time, the termination time, and the method by which time is advanced during the analysis. The analysis time can optionally be subdivided into a number of time periods. If the analysis is from time 0 to time T and it is split into three periods, the first period is defined from time 0 to time t_1, the second period is defined from time t_1 to time t_2, and the third period is defined from time t_2 to time T. (The times t_1 and t_2 are set by the user.) The sum of the times for the three periods is T.

There are many reasons for splitting an analysis into multiple time periods. It is usually desirable to start time periods when a change in the model or in the loading conditions occurs that is significant enough to warrant switching to a different set of solver parameters or boundary conditions. Many of Sierra/SM's features can be toggled on or off or changed during different time periods. For example, boundary conditions and contact can be activated during certain time periods. Similarly, many solution parameters can change based on the time period.

Time stepping can be uniform during a solution period, or it can vary during a solution period by using a function to control the time step as a function of analysis time. In Adagio, an automatic time stepping scheme can also be used to automatically take larger time steps at points in the analysis when the solution is relatively easy, and take smaller time steps when it becomes more difficult to solve a time step.

A description of the `TIME CONTROL` command block follows in Section 4.11.1. Section 4.11.2 describes the `ADAPTIVE TIME STEPPING` command block, which allows the user to adapt the size of the time step based on solution difficulty. A simple example of a `TIME CONTROL` command block is presented in Section 4.11.3.

4.11.1 Command Blocks for Time Control and Time Stepping

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK <string>time_block_name
    START TIME = <real>start_time_value
    BEGIN PARAMETERS FOR ADAGIO REGION <string>region_name
      #
      # Time control parameters specific to ADAGIO
      # are set in this command block.
      #
    END [PARAMETERS FOR ADAGIO REGION <string>region_name]
  END [TIME STEPPING BLOCK <string>time_block_name]
  TERMINATION TIME = <real>termination_time
END [TIME CONTROL]
```

Sierra/SM time control resides in a `TIME CONTROL` command block. The command block begins with the input line

```
BEGIN TIME CONTROL
```

and terminates an input line of the following form:

```
END [TIME CONTROL]
```

An arbitrary number of TIME STEPPING BLOCK command blocks can be present to define individual time stepping periods within the TIME CONTROL command block. Each TIME STEPPING BLOCK command block contains the time at which the time stepping starts and a number of parameters that set time-related values for the analysis. Each time period terminates at the start time of the following time period. The start times for the TIME STEPPING BLOCK command blocks must appear in increasing order or an error message will result. The example in Section 4.11.3 shows the overall structure of the TIME CONTROL command block.

In the above input lines, the parameters are as follows:

- The string time_block_name is a name for the time period. Every time period must have a unique name. These names are referenced to control solution parameters or to activate/deactivate functionality.
- The real value start_time_value is the start time for this TIME STEPPING BLOCK command block. A time period goes from its start time until the start time of the next period or the termination time. The start time may be negative.
- The string region_name is the name of the region affected by the parameters (see Section 2.2).

The final termination time for the analysis is given by the following command line:

```
TERMINATION TIME = <real>termination_time
```

Here, termination_time is the time at which the analysis will stop. The TERMINATION TIME command line appears inside the TIME CONTROL command block but outside of any TIME STEPPING BLOCK command block.

The TERMINATION TIME command line can appear before the first TIME STEPPING BLOCK command block or after the last TIME STEPPING BLOCK command block. Note that it is permissible to have TIME STEPPING BLOCK command blocks with start times greater than the termination time; in this case, those command blocks that have start times after the termination time are not executed. Only one TERMINATION TIME command line can appear in the TIME CONTROL command block. If more than one of these command lines appears, an error is generated.

Nested inside the TIME STEPPING BLOCK command block is a PARAMETERS FOR ADAGIO REGION command block containing parameters that control the time stepping.

```
BEGIN PARAMETERS FOR ADAGIO REGION <string>region_name
  TIME INCREMENT = <real>time_increment_value
  NUMBER OF TIME STEPS = <integer>nsteps
  TIME INCREMENT FUNCTION = <string>time_function
END [PARAMETERS FOR ADAGIO REGION <string>region_name]
```

These parameters are specific to an Adagio analysis.

The command block begins with an input line of the form

```
BEGIN PARAMETERS FOR ADAGIO REGION <string>region_name
```

and is terminated with an input line of the following form:

```
END [PARAMETERS FOR ADAGIO REGION <string>region_name]
```

As noted previously, the string `region_name` is the name of the region to which the parameters apply. The command lines nested inside the `PARAMETERS FOR ADAGIO REGION` command block are used to control the way time stepping occurs, and are described in Sections 4.11.1.1 through 4.11.1.3. Only one of these command lines (`TIME INCREMENT`, `NUMBER OF TIME STEPS`, or `TIME INCREMENT FUNCTION`) may be used in a given time period.

4.11.1.1 Time Increment

```
TIME INCREMENT = <real>time_increment_value
```

The `TIME INCREMENT` command line directly specifies the size of a uniform time step that will be used during a time period. The size of the time step is specified with the parameter `time_increment_value`.

4.11.1.2 Number of Time Steps

```
NUMBER OF TIME STEPS = <integer>nsteps
```

The `NUMBER OF TIME STEPS` command line specifies the number of uniform time steps that will be used during a time period. The number of time steps is specified with the parameter `nsteps`. If this command line is used, the time step size is computed by dividing the length of the time period by the specified number of time steps.

4.11.1.3 Time Increment Function

```
TIME INCREMENT FUNCTION = <string>time_function
```

The `TIME INCREMENT FUNCTION` command line allows the time step to be specified as a function of analysis time. The function used to control the time step is referenced by the parameter `time_function`. The actual function is defined within a `DEFINITION FOR FUNCTION` command block within the SIERRA scope. It is often convenient to use a function of the `PIECEWISE CONSTANT` type (see Section 2.1.5) with this command line. Using a function to control time incrementation allows for the time increment to be conveniently switched many times during an analysis without creating a new time block every time the time increment needs to be changed.

4.11.2 Adaptive Time Stepping

```
BEGIN ADAPTIVE TIME STEPPING
  METHOD = <string>SOLVER|MATERIAL(SOLVER)
    [DURING <string list>period_names]
  TARGET ITERATIONS = <integer>target_iter(0)
    [DURING <string list>period_names]
  ITERATION WINDOW = <integer>iter_window
    [DURING <string list>period_names]
  CUTBACK FACTOR = <real>cutback_factor(0.5)
    [DURING <string list>period_names]
  GROWTH FACTOR = <real>growth_factor(1.5)
    [DURING <string list>period_names]
  MAXIMUM FAILURE CUTBACKS = <integer>max_cutbacks(5)
    [DURING <string list>period_names]
  MAXIMUM MULTIPLIER = <real>max_multiplier(infinity)
    [DURING <string list>period_names]
  MINIMUM MULTIPLIER = <real>min_multiplier(1.0e-8)
    [DURING <string list>period_names]
  RESET AT NEW PERIOD = TRUE|FALSE(TRUE)
    [DURING <string list>period_names]
  ACTIVE PERIODS = <string list>period_names
END [ADAPTIVE TIME STEPPING]
```

Adagio has the capability to adapt the time step size during an analysis based on either solution difficulty or on feedback from material models. This capability is enabled by inserting an `ADAPTIVE TIME STEPPING` command block in the region scope. If this command block is not present in the Adagio region, the time stepping specified in the `TIME CONTROL` command block will be used. In addition to adjusting the size of time steps as the analysis proceeds, the adaptive time stepping algorithm can attempt to solve a load step by using a smaller time step if a solution fails to converge.

The adaptive time stepping algorithm works by computing a multiplier that is applied to the time step specified in the `TIME CONTROL` command block. Thus, even if adaptive time stepping is used, the time step must be specified for all loading periods in the `TIME CONTROL` command block. For the initial load step in each solution period, Adagio always uses the prescribed time step. In subsequent steps, the adaptive time stepping algorithm computes a factor that is multiplied by the prescribed time step to adjust the time step based on solution difficulty.

The command lines nested within the `ADAPTIVE TIME STEPPING` command are used to control the behavior of this feature and are described in Sections 4.11.2.1 through 4.11.2.10.

Many of these command lines can optionally be set for a specific time period. To set a parameter for a specific time period, the user can append the `DURING` keyword followed by a list of applicable period names to a command line (see Section 4.2.1. Multiple instances of these command lines can exist to set the parameter differently for different periods. If a command line is not appended with the `DURING` specification, it is used to set the default behavior for all time periods. Setting a default in this way does not preclude setting a period-specific value for that parameter with another

instance of the same command line.

4.11.2.1 Method

```
METHOD = <string>SOLVER|MATERIAL(SOLVER)
  [DURING <string list>period_names]
```

The METHOD command line is used to select the type of adaptive time stepping to be used. The default SOLVER method adapts the time step based on the difficulty of the solution. Solution difficulty is determined based on the number of iterations required to achieve convergence. If the SOLVER method is used, the TARGET ITERATIONS command line can be used to specify the target iterations used by this method to adapt the time step. The default target iterations is zero allowing the code to cutback the time step as necessary when an element inverts, contact requires a smaller time step, or a material model suggests a smaller time step.

The MATERIAL method adapts the time step based on feedback from the material model. Some material models are capable of computing a recommended time step. When material-based adaptive time stepping is used, the minimum time step recommended by all material integration points in the model is used.

4.11.2.2 Target Iterations

```
TARGET ITERATIONS = <integer>target_iter(0)
  [DURING <string list>period_names]
```

The solver method for adaptive time stepping adjusts the time step to achieve a target level of difficulty. The level of difficulty is determined by the total number of core solver iterations required to solve for a time step. The TARGET ITERATIONS command is used to specify a target number of iterations per time step. This command line is optional.

The specified value of target_iter is used in conjunction with the value of iter_window (specified with the ITERATION WINDOW command documented in Section 4.11.2.3) to control the next step size. If the number of iterations for the previous step is greater than (target_iter-iter_window), and less than (target_iter+iter_window), the step size is kept the same. If the number of iterations is greater than (target_iter+iter_window), the step size is decreased, and if it is less than (target_iter-iter_window), the step size is increased.

4.11.2.3 Iteration Window

```
ITERATION WINDOW = <integer>iter_window
  [DURING <string list>period_names]
```

The ITERATION WINDOW command is used to specify the size of the window for the iteration count used in the solver type of adaptive time stepping. See Section 4.11.2.2 for an explanation

of how this is used in conjunction with the specified target iterations to adaptively choose the time step size. The default value of iter_window is the value specified in the TARGET ITERATIONS command line divided by 5.

4.11.2.4 Cutback Factor

```
CUTBACK FACTOR = <real>cutback_factor(0.5)
  [DURING <string list>period_names]
```

The CUTBACK FACTOR command line controls the amount by which the next time step is reduced if the solution is difficult. The parameter cutback_factor, which has a default value of 0.5, specifies the multiplier that is applied to the current time step with each cutback. The parameter also controls the amount that the current time step is cut back if a solution fails. Note that the CUTBACK FACTOR command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.11.2.5 Growth Factor

```
GROWTH FACTOR = <real>growth_factor(1.5)
  [DURING <string list>period_names]
```

The GROWTH FACTOR command line controls the amount by which the next time step is increased if the solution is easy. The parameter growth_factor, which has a default value of 1.5, specifies the multiplier (a 50 percent increase) that is applied to the current time step each time the time step is increased. Note that the GROWTH FACTOR command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.11.2.6 Maximum Failure Cutbacks

```
MAXIMUM FAILURE CUTBACKS = <integer>max_cutbacks(5)
  [DURING <string list>period_names]
```

The MAXIMUM FAILURE CUTBACKS command line controls how many cutbacks to a time step can be taken if a solution fails to converge. The The parameter max_cutbacks has a default value of 5 if adaptive time stepping is enabled. If an ADAPTIVE TIME STEPPING command block is not present, the code will exit rather than attempting to cut back in the event of an unsuccessful solution. Note that the MAXIMUM FAILURE CUTBACKS command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.11.2.7 Maximum Multiplier

```
MAXIMUM MULTIPLIER = <real>max_multiplier(infinity)
  [DURING <string list>period_names]
```

Time steps are adaptively adjusted by updating a multiplier that is applied to the base time step specified in the TIME CONTROL command block. By default, there are no limits on the size of this multiplier. The MAXIMUM MULTIPLIER command line can be used to limit the growth of the time step. The parameter max_multiplier specifies the upper bound for the multiplier. The largest possible time step in an analysis is the product of max_multiplier and the baseline time step size. This baseline size is specified in the TIME CONTROL command block. Note that the MAXIMUM MULTIPLIER command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.11.2.8 Minimum Multiplier

```
MINIMUM MULTIPLIER = <real>min_multiplier(1.0e-8)
  [DURING <string list>period_names]
```

Time steps are adaptively adjusted by updating a multiplier that is applied to the base time step specified in the TIME CONTROL command block. By default, the MINIMUM MULTIPLIER is set to the smaller of $1.0e-8$ and $(cutback_factor)^{(max_cutbacks)}$. The MINIMUM MULTIPLIER command line can be used to limit the shrinkage of the time step. The parameter min_multiplier specifies the lower bound for the multiplier. The smallest possible time step in an analysis is the product of min_multiplier and the baseline time step size. Note that the MINIMUM MULTIPLIER command line can be appended with the DURING specification, as discussed in Section 4.2.1.

4.11.2.9 Reset at New Period

```
RESET AT NEW PERIOD = TRUE|FALSE(TRUE)
  [DURING <string list>period_names]
```

This command controls whether the time step is reset to the user-specified base time step at the beginning of a new time period. If this is set to TRUE, which is the default value, the time step is reset. If this parameter is set to FALSE, the time step at the end of the previous period is maintained. If the base time step is changed in the new period, the multiplier is adjusted to maintain the same time step. If the resulting multiplier is outside the specified range, it will be adjusted to stay within that range and the time step will change. The value of this parameter can optionally be specified per time period with the DURING option (see Section 4.2.1). If adaptive time stepping is inactive for a given period, the time step will always be reset even if it is requested that the adaptive time step not be reset.

4.11.2.10 Active Periods

```
ACTIVE PERIODS = <string list>period_names
```

If the ADAPTIVE TIME STEPPING command block is present in the Adagio region, it is active for all periods by default. Adaptive time stepping can be activated for specific time periods with

the ACTIVE PERIODS command line. See Section 2.5 for more information about this optional command line.

4.11.3 Time Control Example

The following is a simple example of a TIME CONTROL command block:

```
BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK p1
    START TIME = 0.0
    BEGIN PARAMETERS FOR ADAGIO REGION adagio_region
      NUMBER OF TIME STEPS = 10
    END
  END
  BEGIN TIME STEPPING BLOCK p2
    START TIME = 1.0
    BEGIN PARAMETERS FOR ADAGIO REGION adagio_region
      TIME INCREMENT = 0.2
    END
  END
  TERMINATION TIME = 2.0
END
```

The first TIME STEPPING BLOCK, p1, begins at time 0.0, the initial start time, and terminates at time 1.0. The second TIME STEPPING BLOCK, p2, begins at time 1.0 and terminates at time 2.0, the time listed on the TERMINATION TIME command line. The TIME STEPPING BLOCK names p1 and p2 can be referenced elsewhere to activate boundary conditions or control the solver in different ways during the two periods.

4.12 Implicit Dynamic Time Integration

Adagio has the ability to perform implicit solutions on both quasistatic and dynamic problems. In quasistatic calculations, the solution for static equilibrium is obtained at each step, ignoring the inertial terms of the equations of motion. In dynamic problems, the inertial forces are included, and the HHT time integrator [5] is used to integrate the equations of motion in time. The behavior of the HHT integrator is controlled with three parameters: α, β, and γ. With proper selection of these parameters, the HHT integrator is unconditionally stable. If $\alpha = 0$, this reduces to the Newmark method [7]. The trapezoidal rule is recovered if $\alpha = 0$, $\beta = 0.25$, and $\gamma = 0.5$. For a detailed discussion of the theory of implicit time integrators, see Reference 6.

4.12.1 Implicit Dynamics

```
BEGIN IMPLICIT DYNAMICS
  ACTIVE PERIODS = <string list>period_names
  USE HHT INTEGRATION
  ALPHA = <real>alpha(-0.1) [DURING <string list>period_names]
  BETA = <real>beta(0.6) [DURING <string list>period_names]
  GAMMA = <real>beta(0.3025) [DURING <string list>period_names]
  TIME INTEGRATION CONTROL = <string>ADAPTIVE|COMPUTERESIDUAL|
    IGNORE(IGNORE) [DURING <string list>period_names]
  INCREASE ERROR THRESHOLD = <real>increase_threshold(0.02)
    [DURING <string list>period_names]
  HOLD ERROR THRESHOLD = <real>hold_threshold(0.10)
    [DURING <string list>period_names]
  DECREASE ERROR THRESHOLD = <real>decrease_threshold(0.25)
    [DURING <string list>period_names]
END [IMPLICIT DYNAMICS]
```

The `IMPLICIT DYNAMICS` command block enables implicit dynamics and contains commands to control the behavior of implicit time integration. This command block is placed in the `ADAGIO REGION` command block. Note that prior versions of Adagio required an Andante region to use implicit dynamics. The Andante region has been eliminated, and this capability has all been moved to the Adagio region, which gives the analyst much more flexibility. Formerly, it was necessary to create a new procedure for switching between implicit dynamics and quasistatics in an analysis. Now, these different analysis types can all be done within a single region, and the command lines in the `IMPLICIT DYNAMICS` command block can be used to enable or disable implicit dynamics for specific time periods within a single region. Sections 4.12.1.1 through 4.12.1.4 describe the command lines in this command block.

4.12.1.1 Active Periods

```
ACTIVE PERIODS = <string list>period_names
```

If the `IMPLICIT DYNAMICS` command block is present in the Adagio region, it is active for all periods by default. Implicit dynamics can be activated for specific time periods with the `ACTIVE PERIODS` command line. See Section 2.5 for more information about this optional command line.

4.12.1.2 Use HHT Integration

```
USE HHT INTEGRATION
```

Adagio currently only supports the HHT algorithm for implicit time integration. The `USE HHT INTEGRATION` command line is provided as a placeholder to allow for other time integrators in the future, but currently this command line has no effect.

4.12.1.3 HHT Parameters

```
ALPHA = <real>alpha(-0.1) [DURING <string list>period_names]
BETA = <real>beta(0.6) [DURING <string list>period_names]
GAMMA = <real>gamma(0.3025) [DURING <string list>period_names]
```

The `ALPHA`, `BETA`, and `GAMMA` command lines specify the HHT integration parameters. These parameters can all be specified for the entire analysis, or they can vary by solution period by appending them with the optional `DURING` specification (see Section 4.2.1).

- The `ALPHA` command line specifies α, which is the dissipation factor applied to the internal force vector. The value of `alpha` controls numerical damping and must always be less than or equal to zero. Its default value of -0.1 results in slight numerical damping of modes with frequencies higher than can be resolved with a given time step. To maintain second-order accuracy, the value of `alpha` should not be less than $-\frac{1}{3}$.
- The `BETA` command line specifies the stability parameter β for time integrators in the Newmark family. The default value of `beta` is 0.6. It should have a value of $0.5 - \alpha$ to maintain second-order accuracy.
- The `GAMMA` command line specifies the dissipation factor γ for Newmark time integrators. The default value of `gamma` is 0.3025. It should have a value of $0.25(1 - \alpha)^2$ to maintain second-order accuracy.

4.12.1.4 Implicit Dynamic Adaptive Time Stepping

```
TIME INTEGRATION CONTROL = <string>ADAPTIVE|COMPUTERESIDUAL|
  IGNORE(IGNORE) [DURING <string list>period_names]
INCREASE ERROR THRESHOLD = <real>increase_threshold(0.02)
  [DURING <string list>period_names]
HOLD ERROR THRESHOLD = <real>hold_threshold(0.10)
  [DURING <string list>period_names]
```

```
DECREASE ERROR THRESHOLD = <real>decrease_threshold(0.25)
  [DURING <string list>period_names]
```

Adagio provides an adaptive time stepping capability that adjusts the time step based on the error due to time discretization with implicit time integration. If this capability is activated, after a solution is obtained for each time step, an interpolated solution for the half step is computed based on the assumption that acceleration is constant over the time step. The residual is computed using this interpolated solution. The residual is indicative of the error introduced due to the time step, and is used to adjust the time step. The residual is always evaluated relative to the reference load. The size of the next time step is increased if the residual is low, or decreased if the residual is high.

The command lines for adaptive time stepping can all be optionally used in the `IMPLICIT DYNAMICS` command block. These command lines control the behavior of the implicit time integrator's algorithm for adaptive time stepping. All the parameters specified by these command lines can vary by period by appending the command lines with the optional `DURING` specification (see Section 4.2.1).

- The `TIME INTEGRATION CONTROL` command line controls whether the algorithm for adaptive time stepping should be used. Three options are available: `ADAPTIVE`, `COMPUTERESIDUAL`, and `IGNORE`. If the option is set to `ADAPTIVE`, the midstep residuals are computed and the time step is adjusted based on the residuals. If the option is set to `COMPUTERESIDUAL`, the midstep residual is computed and reported in the log file, but the time step is not adjusted. If the option is set to `IGNORE`, the default, no midstep residuals are computed, and the time step is not adjusted.

- The `INCREASE ERROR THRESHOLD` command line controls when the time step is increased. If the relative midstep residual is below the specified value, the next time step will be increased. The default value of `increase_threshold` is 0.02.

- The `HOLD ERROR THRESHOLD` command line controls when the time step is held constant. If the relative midstep residual is below the hold threshold and greater than the increase threshold, the next time step will be held constant. The default value of `hold_threshold` is 0.10, which must be greater than the value of `increase_threshold`.

- The `DECREASE ERROR THRESHOLD` command line controls when the time step is decreased. If the relative midstep residual is below the decrease threshold and greater than the hold threshold, the current time step is accepted but the next time step will be decreased. If the relative midstep residual is greater than the decrease threshold, the current solution is rejected and recomputed with a smaller time step. The default value of `decrease_threshold` is 0.25, which must be greater than `hold_threshold`.

4.13 References

1. Blanford, M. L., M. W. Heinstein, and S. W. Key. *JAS3D – A Multi-Strategy Iterative Code for Solid Mechanics Analysis Users' Instructions, Release 2.0*, Draft SAND report. Albuquerque, NM: Sandia National Laboratories, September 2001.

2. Fletcher, R., and C. M. Reeves. "Function Minimization by Conjugate Gradients." *The Computer Journal* 7 (1964): 149–154.

3. Farhat, C., M. Lesoinne, and K. Pierson. "A Scalable Dual-Primal Domain Decomposition Method." *Numerical Linear Algebra with Applications* 7 (2000): 687–714.

4. Farhat, C., M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen. "FETI-DP: A Dual-Primal Unified FETI Method – Part I: A Faster Alternative to the Two-Level FETI Method." *International Journal for Numerical Methods in Engineering* 50 (2001): 1523–1544.

5. 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* 5 (1977): 283–292.

6. Hughes, T. J. R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

7. Newmark, N. M. "A Method of Computation for Structural Dynamics." *Journal of the Engineering Mechanics Division*, ASCE 85, no. EM3 (1959): 67–94.

Chapter 5

Materials

This chapter describes material models that can be used in conjunction with the elements in Presto and Adagio. Most of the material models have an interface that allows them to be used by the elements in both codes. Even though a material model can be used by both codes, usage of the model may be better suited for the type of problems solved by one code rather than the type of problems solved by the other code. For example, a material model that was built to characterize behavior over a long time would be better suited for use in Adagio than in Presto. If a particular material model is better suited for one code rather than the other, this usage information is provided in the description of that model.

The material models described in this chapter are, in general, applicable to solid elements. The structural elements, such as shells and beams, have a much more limited set of material models. Chapter 6 describes the element library, including which material models are available for the various elements. The introduction to Chapter 6 summarizes all the element types in Presto and Adagio. For each element type, a list of available material models is provided.

When using the nonlinear material models, you may want to output state variables that are associated with these models. See Section 9.9.2 to learn how to output the state variables for the various nonlinear material models.

Most material models for solid elements are available in two libraries. The newer library is the LAME library [17], but it is not the default. The line command to activate the LAME material library for a particular section is described in Section 6.2.1.

General Model Form. PROPERTY SPECIFICATION FOR MATERIAL command blocks appear in the SIERRA scope in the general form shown below.

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  # Command lines and command blocks for material
  # models appear in this scope.
  #
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

PROPERTY SPECIFICATION FOR MATERIAL command blocks are physics independent in the sense that the information in them can be shared by more than one application. For example, some of the PROPERTY SPECIFICATION FOR MATERIAL command blocks contain density information that can be shared among several applications.

The command block begins with the line:

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
```

and terminates with the line:

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

Here, the string mat_name is a user-specified name for the command block. This name is typically descriptive of the material being modeled, e.g., aluminum_t6061.

Within a PROPERTY SPECIFICATION FOR MATERIAL command block, there will be other command blocks and possibly other general material command lines that are used to describe particular material models. The general material command lines, if present, are listed first, followed by one or more material-model command blocks. The general material command lines may be used to specify the density of the material, the Biot's coefficient, and the application of temperatures and thermal strains to one-, two- or three-dimensional elements. Each material-model command block follows the naming convention of PARAMETERS FOR MODEL model_name, where model_name identifies a particular material model, such as elastic, elastic-plastic, or orthotropic crush. Each such command block contains all the parameters needed to describe a particular material model.

As noted above, more than one material-model command block can appear within a PROPERTY SPECIFICATION FOR MATERIAL command block. Suppose we have a PROPERTY SPECIFICATION FOR MATERIAL command block called steel. It would be possible to have two material-model command blocks within this command block. One of the material-model command blocks would provide an elastic model for steel; the other material-model command block would provide an elastic-plastic model for steel. The general form of a PROPERTY SPECIFICATION FOR MATERIAL command block would be as follows:

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  # General material command lines
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_coefficient_value
  #
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL <string>model_name1
    #
    # Parameters for material model model_name1
  END PARAMETERS FOR MODEL <string>model_name1
  #
  BEGIN PARAMETERS FOR MODEL <string> model_name2
    #
    # Parameters for material model model_name2
  END PARAMETERS FOR MODEL <string> model_name2
  #
  # Additional model command blocks if required
  #
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

In the above general form for a PROPERTY SPECIFICATION FOR MATERIAL command block, the string model_name1 could be ELASTIC and the string model_name2 could be ORTHOTROPIC CRUSH. Typically, however, only one material model would be desired for a given block, and the PROPERTY SPECIFICATION FOR MATERIAL command block would have only one PARAMETERS FOR MODEL command block. A particular material model may only appear once within a given PROPERTY SPECIFICATION FOR MATERIAL command block.

Although multiple material models can be defined for one material within a PROPERTY SPECIFICATION FOR MATERIAL command block, only one material model is actually used for a given element block during an analysis. The ability to define multiple constitutive models for one material is provided as a convenience to enable the user to easily switch between models. The material name and the model name are both referenced when material models are assigned to element blocks within the FINITE ELEMENT MODEL command block, which is described in Section 6.1.

This chapter is organized to correspond to the general form presented for the PROPERTY SPECIFICATION FOR MATERIAL command block. Section 5.1 discusses the DENSITY command line, the BIOTS COEFFICIENT command line, and the command lines used for thermal strains, and also explains how temperatures and thermal strains are applied. Section 5.2 describes each of the material models that are shared by Presto and Adagio. References applicable for both

Presto and Adagio are listed in Section 5.4.

As indicated in the introductory material, not all the material models available are applicable to all element types. As one example, there are the material models available for use with cohesive zone elements or with Dash contact, as detailed in Section 5.3. As another example, there is a one-dimensional elastic material model that is used for a truss element but is not applicable to solid elements such as hexahedra or tetrahedra. For this particular example, the specific material-model usage is hidden from the user. If the user specifies a linear elastic material model for a truss, the one-dimensional elastic material model is used. If the user specifies a linear elastic material model for a hexahedron, a full three-dimensional elastic material model is used. As another example, the energy-dependent material models available in Presto cannot be used for a one-dimensional element such as a truss. The energy-dependent material models can only be used for solid elements such as hexahedra and tetrahedra. (Chapter 6 indicates what material models are available for which element models.)

For each material model, the parameters needed to describe that model are listed in the section pertinent to that particular model. Solid models with elastic constants require only two elastic constants. These two constants are then used to generate all the elastic constants for the model. For example, if the user specifies Young's modulus and Poisson's ratio, then the shear modulus, bulk modulus, and lambda are calculated. If the shear modulus and lambda are specified, then Young's modulus, Poisson's ratio, and the bulk modulus are calculated.

The various nonlinear material models have state variables. See Section 9.9.2 to learn how to output the state variables for the nonlinear material models.

Note that only brief descriptions of the material models are presented in this chapter. For a detailed description of the various material models, you will need to consult a variety of references. Specific references are identified for most of the material models shared by Presto and Adagio.

5.1 General Material Commands

A PROPERTY SPECIFICATION FOR MATERIAL command block for a particular material may include additional command lines that are applicable to all the material models specified within the command block. These command lines related to density, to Biot's coefficient, and to thermal strain behavior are discussed, respectively, in Section 5.1.1, Section 5.1.2, and Section 5.1.3.

5.1.1 Density Command

```
DENSITY = <real>density_value
```

This command line specifies the density of the material described in a PROPERTY SPECIFICATION FOR MATERIAL command block. The units of the input parameter density_value are specified as mass per unit volume.

As previously explained, a PROPERTY SPECIFICATION FOR MATERIAL command block can contain one or more PARAMETERS FOR MODEL command blocks. The specified density_value for the material will be used with all of the models described in these PARAMETERS FOR MODEL command blocks.

5.1.2 Biot's Coefficient Command

```
BIOTS COEFFICIENT = <real>biots_value
```

This command line specifies the Biot's coefficient of the material. Biot's coefficient is used with the pore pressure capability. See Section 7.10 for more information on pore pressure. If not given, the value defaults to 1.0. This parameter is unitless.

As previously explained, a PROPERTY SPECIFICATION FOR MATERIAL command block can contain one or more PARAMETERS FOR MODEL command blocks. The specified biots_value for the material will be used with all of the models described in these PARAMETERS FOR MODEL command blocks.

5.1.3 Thermal Strain Behavior

Isotropic and orthotropic thermal strains may be defined for use by material models listed in a PROPERTY SPECIFICATION FOR MATERIAL command block. Section 5.1.3.1 describes the command lines that are used to define the thermal strain behavior. These command lines must be used in conjunction with other command blocks outside of a PROPERTY SPECIFICATION FOR MATERIAL command block for the calculations of thermal strains to be activated. Section 5.1.3.2 explains the process for activating thermal strains.

5.1.3.1 Defining Thermal Strains

```
THERMAL STRAIN FUNCTION = <string>thermal_strain_function
# or all three of the following
THERMAL STRAIN X FUNCTION =
  <string>thermal_strain_x_function
THERMAL STRAIN Y FUNCTION =
  <string>thermal_strain_y_function
THERMAL STRAIN Z FUNCTION =
  <string>thermal_strain_z_function
```

A `PROPERTY SPECIFICATION FOR MATERIAL` command block may include command lines that define thermal strain behavior. It is possible to specify either an isotropic thermal-strain field using the command line `THERMAL STRAIN FUNCTION` or an orthotropic thermal-strain field using the command lines `THERMAL STRAIN X FUNCTION`, `THERMAL STRAIN Y FUNCTION`, and `THERMAL STRAIN Z FUNCTION`. For any of these command lines, the user supplies a thermal strain function (via a `DEFINITION FOR FUNCTION` command block), which defines the thermal strain as a function of temperature. The computed thermal strain is then subtracted from the strain passed to the material model.

A thermal strain can be applied to any one-dimensional, two-dimensional or three-dimensional element, that utilizes a material model, regardless of material type. For a three-dimensional element such as a hexahedron or tetrahedron, the thermal strains are applied to the strain in the global *XYZ* coordinate system. For the isotropic case, the thermal strains are the same in the *X*-direction, the *Y*-direction, and the *Z*-direction. For the anisotropic case, the thermal strains can be different in each of the three global directions—*X*, *Y*, and *Z*. For a two-dimensional element, shell or membrane, the thermal strain corresponding to the `THERMAL STRAIN X FUNCTION` command line is applied to the strain in the shell (or membrane) *r*-direction. (Reference Section 6.2.4 for a discussion of the shell *rst* coordinate system.) The thermal strain corresponding to the `THERMAL STRAIN Y FUNCTION` command line is applied to the strain in the shell (or membrane) *s*-direction. For two-dimensional elements, the current implementation of orthotropic thermal strains is limited, for practical purposes, to special cases—flat sheets of uniform shell elements lying in one of the global planes, e.g., *XY*, *YZ*, or *ZX*. The current orthotropic thermal-strain capability has limited use for shells and membranes in the current release of the code. Tying the orthotropic thermal-strain functionality to the shell orientation functionality (Section 6.2.4) in the future will provide much more useful orthotropic thermal-strain functionality for two-dimensional elements. For a one-dimensional element, beam or truss, only the isotropic thermal-strain function, `THERMAL STRAIN FUNCTION` is applicable.

If an isotropic thermal-strain field is to be applied, the `THERMAL STRAIN FUNCTION` command line is placed in the `PROPERTY SPECIFICATION FOR MATERIAL` command block, outside of the specifications of any material models in the block. Such placement is necessary because the isotropic thermal strain is a general material property, not a property that is specific to any particular constitutive model, such as `ELASTIC` or `ELASTIC-PLASTIC`. The input value of `thermal_strain_function` is the name of the function that defines thermal strain as a function of temperature for the material model described in this particular `PROPERTY SPECIFICATION FOR MATERIAL` command block. The function is defined within the SIERRA scope using a

DEFINITION FOR FUNCTION command block. For more information on how to set the input to compute thermal strains and how to apply temperatures, see Section 5.1.3.2.

The specification of an orthotropic thermal-strain field requires that all three of the THERMAL STRAIN X FUNCTION, THERMAL STRAIN Y FUNCTION, and THERMAL STRAIN Z FUNCTION command lines be placed in the PROPERTY SPECIFICATION FOR MATERIAL command block. All three command lines must be provided, even when there is no thermal strain in one or more directions. The values of thermal_strain_x_function, thermal_strain_y_function, and thermal_strain_z_function are the names of the functions for thermal strains in the *X*-direction, the *Y*-direction, and the *Z*-direction, respectively. These functions are defined within the SIERRA scope using DEFINITION FOR FUNCTION command blocks. To specify that there should be no thermal strain in a given direction, use a function that always evaluates to zero for that direction.

The THERMAL STRAIN FUNCTION command line and the THERMAL STRAIN X FUNCTION, THERMAL STRAIN Y FUNCTION, and THERMAL STRAIN Z FUNCTION command lines are not used for several of the material models, as discussed in Section 5.1.3.2. Note that specification of a thermal strain is identified in the descriptions of the material models in Section 5.2 by the notation "thermal strain option".

5.1.3.2 Activating Thermal Strains

Sierra/SM has the capability to compute thermal strains on three-dimensional continuum, two-dimensional (shell, membrane), and one-dimensional (beam, truss) elements. Three things are required to activate thermal strains:

- First, one or more thermal strain functions (strain as a function of temperature) must be defined. Each thermal strain function is defined with a `DEFINITION FOR FUNCTION` command block. (This function is the standard function definition that appears in the SIERRA scope.) The thermal strain function gives the total thermal strain associated with a given temperature. It is the change in thermal strain with the change in temperature that gives rise to thermal stresses in a body.
- Second, the material models used by blocks that experience thermal strain must have their thermal strain behavior defined. The command lines for defining isotropic and orthotropic thermal strain are described in Section 5.1.3.1. Materials with isotropic thermal strain use the `THERMAL STRAIN FUNCTION` command line, while those with orthotropic thermal strain must define thermal strain in all three directions using the `THERMAL STRAIN X FUNCTION`, `THERMAL STRAIN Y FUNCTION`, and `THERMAL STRAIN Z FUNCTION` command lines. These inputs can be used with all material models with the exception of the following: elastic three-dimensional orthotropic, elastic laminate, Mooney-Rivlin, NLVE three-dimensional orthotropic, Swanson, and viscoelastic Swanson. These models require their own model-specific inputs to define thermal strain and must not use these standard commands. Information for defining thermal strains is provided in the individual descriptions of these models in Section 5.2.
- Third, a temperature field must be applied to the affected blocks. The command block to specify the application of temperatures is `PRESCRIBED TEMPERATURE`, which is implemented as a standard boundary condition. A description of the `PRESCRIBED TEMPERATURE` command block is given in Section 7.9.

Whenever a temperature field is applied, the temperature is prescribed at the nodes, but thermal strain is computed based on element temperature. Element temperatures are obtained by averaging the temperatures of the nodes connected to a given element. Thermal strains are applied in rate form, so the thermal strain in an element is relative to the thermal strain at the initial temperature. Thus, the initial temperature is the stress-free temperature. If desired, a different stress-free temperature can be used by prescribing the initial temperature with the `INITIAL CONDITION` command block as described in Section 7.3.

5.2 Material Models

This section contains descriptions of the material models that are shared by Presto and Adagio.

5.2.1 Elastic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
  END [PARAMETERS FOR MODEL ELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

An elastic material model is used to describe simple linear elastic behavior of materials. This model is generally valid for small deformations.

The command block for an elastic material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ELASTIC
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ELASTIC]
```

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.

- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.

There are no output variables available for the elastic model. For information about the elastic model, consult Reference 1.

5.2.2 Thermoelastic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL THERMOELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
  END [PARAMETERS FOR MODEL THERMOELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The thermoelastic material model is used to describe the temperature-dependent linear elastic behavior of materials. This model is generally valid for small deformations.

The command block for a thermoelastic material starts with the line:

```
BEGIN PARAMETERS FOR MODEL THERMOELASTIC
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL THERMOELASTIC]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.

- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.

- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.

- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The `YOUNGS MODULUS FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes a scale factor on Young's modulus as a function of temperature.
- The `POISSONS RATIO FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes a scale factor on Poisson's ratio as a function of temperature.

There are no output variables available for the thermoelastic model. For information about the thermoelastic model, consult Reference 1.

5.2.3 Neo-Hookean Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL NEO_HOOKEAN
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
  END [PARAMETERS FOR MODEL NEO_HOOKEAN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The neo-Hookean material model is used to describe linear elastic behavior of materials. Unlike the elastic model, this model is valid for both large and small deformations. The neo-Hookean model uses a strain energy density, which makes it a hyperelastic constitutive model. This feature makes it applicable to finite strains. Currently the neo-Hookean model, as implemented in LAME, does not support thermal strains.

The command block for a neo-Hookean material starts with the line:

```
BEGIN PARAMETERS FOR MODEL NEO_HOOKEAN
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL NEO_HOOKEAN]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.

- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.

There are no output variables available for the neo-Hookean model. For information about the neo-Hookean model, consult Reference 5.

5.2.4 Elastic Fracture Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC_FRACTURE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    MAX STRESS = <real>max_stress
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
  END [PARAMETERS FOR MODEL ELASTIC_FRACTURE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

An elastic fracture material is a simple failure model that is based on linear elastic behavior. The model uses a maximum-principal-stress failure criterion. The stress decays isotropically based on the component of strain parallel to the maximum principal stress. The value of the component of strain over which the stress is decayed to zero is a user-defined parameter for the model. This strain parameter can be adjusted so that failure is mesh independent.

The command block for an elastic fracture material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ELASTIC_FRACTURE
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ELASTIC_FRACTURE]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.

- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The maximum principal stress at which failure occurs is defined with the `MAX STRESS` command line.
- The component of strain over which the stress decays to zero is defined with the `CRITICAL CRACK OPENING STRAIN` command line. This component of strain is aligned with the maximum-principal-stress direction at failure.

Output variables available for this model are listed in Table 9.20.

5.2.5 Elastic-Plastic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING MODULUS = <real>hardening_modulus
    BETA = <real>beta_parameter(1.0)
  END [PARAMETERS FOR MODEL ELASTIC_PLASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The elastic-plastic linear hardening models are used to model materials, typically metals, that undergoing plastic deformation at finite strains. Linear hardening generally refers to the shape of a uniaxial stress-strain curve where the stress increases linearly with the plastic, or permanent, strain. In a three-dimensional framework, hardening is the law that governs how the yield surface grows in stress space. If the yield surface grows uniformly in stress space, the hardening is referred to as isotropic hardening. When `BETA` is 1.0, we have only isotropic hardening.

Because the linear hardening model is relatively simple to integrate, the model also has the ability to define a yield surface that not only grows, or hardens, but also moves in stress space. This ability is known as kinematic hardening. When `BETA` is 0.0, we have only kinematic hardening. The elastic-plastic linear hardening model allows for isotropic hardening, kinematic hardening, or a combination of the two.

The command block for an elastic-plastic material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ELASTIC_PLASTIC]
```

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.
- The thermal strain option is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the YOUNGS MODULUS command line.
  - Poisson's ratio is defined with the POISSONS RATIO command line.
  - The bulk modulus is defined with the BULK MODULUS command line.
  - The shear modulus is defined with the SHEAR MODULUS command line.
  - Lambda is defined with the LAMBDA command line.
- The yield stress is defined with the YIELD STRESS command line.
- The hardening modulus is defined with the HARDENING MODULUS command line.
- The beta parameter is defined with the BETA command line.

Output variables available for this model are listed in Table 9.21. For information about the elastic-plastic model, consult Reference 1.

5.2.6 Elastic-Plastic Power-Law Hardening Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN = <real>luders_strain
  END [PARAMETERS FOR MODEL EP_POWER_HARD]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

A power-law hardening model for elastic-plastic materials is used for modeling metal plasticity up to finite strains. The power-law hardening model, as opposed to the linear hardening model, has a power law fit for the uniaxial stress-strain curve that has the stress increase as the plastic strain raised to some power. The power-law hardening model also has the ability to model materials that exhibit Luder's strains after yield. Due to the more complicated yield behavior, the power-law hardening model can only be used with isotropic hardening.

The command block for an elastic-plastic power-law hardening material starts with the line:

```
BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL EP_POWER_HARD]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.

- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.

- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on specifying and applying thermal strains and temperatures.

- Any two of the following elastic constants are required:

  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.

- The yield stress is defined with the `YIELD STRESS` command line.

- The hardening constant is defined with the `HARDENING CONSTANT` command line.

- The hardening exponent is defined with the `HARDENING EXPONENT` command line.

- The Luder's strain is defined with the `LUDERS STRAIN` command line.

Output variables available for this model are listed in Table 9.22. For information about the elastic-plastic power-law hardening model, consult Reference 1.

5.2.7 Ductile Fracture Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL DUCTILE_FRACTURE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN <real>luders_strain
    CRITICAL TEARING PARAMETER = <real>crit_tearing
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
  END [PARAMETERS FOR MODEL DUCTILE_FRACTURE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This model is identical to the elastic-plastic power-law hardening model with the addition of a failure criterion and a post-failure isotropic decay of the stress to zero within the constitutive model. The point at which failure occurs is defined by a critical tearing parameter. The critical tearing parameter t_p is related to the plastic strain at failure ε_f by the evolution integral

$$t_p = \int_0^{\varepsilon_f} \langle \frac{2\sigma_{\max}}{3\left(\sigma_{\max} - \sigma_m\right)} \rangle^4 d\varepsilon_p \,. \tag{5.1}$$

In Equation (5.1), $\sigma_{\max}$ is the maximum principal stress, and σ_m is the mean stress. The quantity in the angle brackets, the expression

$$\frac{2\sigma_{\max}}{3\left(\sigma_{\max} - \sigma_m\right)} , \tag{5.2}$$

is nonzero only if it evaluates to a positive value. This quantity is set to zero if it has a negative value.

The stress decays isotropically based on the component of strain parallel to the maximum principal stress. The value of the component of strain over which the stress is decayed to zero is a user-defined parameter for the model. This strain parameter can be adjusted so that failure is mesh independent.

The command block for an elastic-plastic power-law hardening material with failure starts with the line:

```
BEGIN PARAMETERS FOR MODEL DUCTILE_FRACTURE
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL DUCTILE_FRACTURE]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains and temperatures.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The yield stress is defined with the `YIELD STRESS` command line.
- The hardening constant is defined with the `HARDENING CONSTANT` command line.
- The hardening exponent is defined with the `HARDENING EXPONENT` command line.
- The Luder's strain is defined with the `LUDERS STRAIN` command line.
- The critical tearing parameter is defined with the `CRITICAL TEARING PARAMETER` command line.
- The component of strain over which the stress decays to zero is defined with the `CRITICAL CRACK OPENING STRAIN` command line. This component of strain is aligned with the maximum-principal-stress direction at failure.

Output variables available for this model are listed in Table 9.23. For information about the ductile fracture material model, consult Reference 1.

5.2.8 Multilinear Elastic-Plastic Hardening Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL MULTILINEAR_EP
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    BETA = <real>beta_parameter(1.0)
    HARDENING FUNCTION = <string>hardening_function_name
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
    YIELD STRESS FUNCTION = <string>yield_stress_function_name
  END [PARAMETERS FOR MODEL MULTILINEAR_EP]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This model is similar to the power-law hardening model except that the hardening behavior is described with a piecewise-linear curve as opposed to a power law.

The command block for a multi-linear elastic-plastic hardening material starts with the line:

```
BEGIN PARAMETERS FOR MODEL MULTILINEAR_EP
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL MULTILINEAR_EP]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.

- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.

- The thermal strain option is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.

- Any two of the following elastic constants are required:

  - Young's modulus is defined with the YOUNGS MODULUS command line.
  - Poisson's ratio is defined with the POISSONS RATIO command line.
  - The bulk modulus is defined with the BULK MODULUS command line.
  - The shear modulus is defined with the SHEAR MODULUS command line.
  - Lambda is defined with the LAMBDA command line.

- The yield stress is defined with the YIELD STRESS command line.

- The beta parameter is defined with the BETA command line.

- The HARDENING FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the hardening behavior of the material as a stress versus equivalent plastic strain. This curve is expressed as the additional increment of stress over the yield stress versus equivalent plastic strain, thus the first point on the curve should be (0.0, 0.0).

- The YOUNGS MODULUS FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes a scale factor on Young's modulus as a function of temperature.

- The POISSONS RATIO FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes a scale factor on Poisson's ratio as a function of temperature.

- The YIELD STRESS FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes a scale factor on the yield stress as a function of temperature.

Output variables available for this model are listed in Table 9.24.

5.2.9 Multilinear Elastic-Plastic Hardening Model with Failure

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ML_EP_FAIL
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    BETA = <real>beta_parameter(1.0)
    HARDENING FUNCTION = <string>hardening_function_name
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
    YIELD STRESS FUNCTION = <string>yield_stress_function_name
    CRITICAL TEARING PARAMETER = <real>crit_tearing
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
    CRITICAL BIAXIALITY RATIO = <real>critical_ratio(0.0)
  END [PARAMETERS FOR MODEL ML_EP_FAIL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

This model is similar to the power-law hardening model except that the hardening behavior is described with a piecewise-linear curve as opposed to a power law. This model incorporates a failure criterion and a post-failure isotropic decay of the stress to zero within the constitutive model. The point at which failure occurs is defined by a critical tearing parameter. The critical tearing parameter t_p is related to the plastic strain at failure ε_f by the evolution integral:

$$t_p = \int_0^{\varepsilon_f} \left\langle \frac{2\sigma_{\max}}{3\left(\sigma_{\max} - \sigma_m\right)} \right\rangle^4 d\varepsilon_p \,. \tag{5.3}$$

In Equation (5.3), $\sigma_{\max}$ is the maximum principal stress, and σ_m is the mean stress. The quantity in the angle brackets, the expression

$$\frac{2\sigma_{\max}}{3\left(\sigma_{\max} - \sigma_m\right)}\,, \tag{5.4}$$

is nonzero only if it evaluates to a positive value. This quantity is set to zero if it has a negative value.

The stress decays isotropically based on the component of strain parallel to the maximum principal stress. The value of the component of strain over which the stress is decayed to zero is a user-defined parameter for the model. This strain parameter can be adjusted so that failure is mesh independent.

The command block for a multi-linear elastic-plastic hardening material with failure starts with the line:

```
BEGIN PARAMETERS FOR MODEL ML_EP_FAIL
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ML_EP_FAIL]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The yield stress is defined with the `YIELD STRESS` command line.
- The beta parameter is defined with the `BETA` command line.
- The `HARDENING FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the hardening behavior of the material as a stress versus equivalent plastic strain. This curve is expressed as the additional increment of stress over the yield stress versus equivalent plastic strain, thus the first point on the curve should be (0.0, 0.0).

- The `YOUNGS MODULUS FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes a scale factor on Young's modulus as a function of temperature.

- The `POISSONS RATIO FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes a scale factor on Poisson's ratio as a function of temperature.

- The `YIELD STRESS FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes a scale factor on the yield stress as a function of temperature.

- The critical tearing parameter is defined with the `CRITICAL TEARING PARAMETER` command line.

- The component of strain over which the stress decays to zero is defined with the `CRITICAL CRACK OPENING STRAIN` command line. This component of strain is aligned with the maximum-principal-strain direction at failure.

- The `CRITICAL BIAXIALITY RATIO` command line should be only be used under very specific conditions and with extreme caution. It is intended only for the special case where the stress state is very nearly biaxial, resulting in nearly identical principal strains. In this case, the eigenvector computation can give unreliable results for the direction vectors for the principal strains. If the ratio of the difference between two principal strains divided by their magnitude is less that the value specified by the `CRITICAL BIAXIALITY RATIO` command, the direction of the vector defining the crack opening strain will be given equal weight in each of the principal directions associated with those strains. The default value for the critical ratio is 0.0, which means that the principal directions will be accepted directly from the eigenvector computation. This command should only be used as a last resort if the loading is nearly biaxial and the default value has been demonstrated to lead to elements with very high strains that are not failing long after reaching the critical tearing parameter.

Output variables available for this model are listed in Table 9.25.

5.2.10 Johnson-Cook Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL JOHNSON_COOK
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    RHOCV = <real>rho_cv
    RATE CONSTANT = <real>rate_constant
    THERMAL EXPONENT = <real>thermal_exponent
    REFERENCE TEMPERATURE = <real>reference_temperature
    MELT TEMPERATURE = <real>melt_temperature
  END [PARAMETERS FOR MODEL JOHNSON_COOK]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Johnson-Cook model is used to model materials, typically metals, undergoing plastic deformation at finite strains. The hardening function is defined by:

$$\sigma = (\sigma_y + B(\bar{\epsilon}_p)^N)(1 + C\ln(e^*))(1 - (T^*)^M)\,, \tag{5.5}$$

where σ_y is the yield stress, B is the hardening constant, $\bar{\epsilon}_p$ is the equivalent plastic strain, N is the hardening exponent, C is the rate constant, e^* is the non-dimensional effective total strain rate, and T^* is the homologous temperature. T^* is defined as:

$$T^* = (T - T_{ref})/(T_{melt} - T_{ref})\,, \tag{5.6}$$

where T is the current temperature, T_{ref} is the reference temperature, and T_{melt} is the melt temperature. In the case where $M <= 0$, $(1 - (T^*)^M) = 1$.

Plastic work results in adiabatic heating. The resulting change in temperature is computed according to the function:

$$\Delta T = \frac{0.95\sigma_y \Delta\bar{\epsilon}_p}{\rho c_v}, \tag{5.7}$$

where ρ is the material density and c_v is the specific heat.

For more information about the Johnson-Cook material model, consult Reference 22.

In the command blocks that define the Johnson-Cook model:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The yield stress is defined with the `YIELD STRESS` command line.
- The hardening constant is defined with the `HARDENING CONSTANT` command line.
- The hardening exponent is defined with the `HARDENING EXPONENT` command line.
- The product ρc_v is defined with the `RHOCV` command line.
- The thermal exponent is defined with the `THERMAL EXPONENT` command line.
- The reference temperature is defined with the `REFERENCE TEMPERATURE` command line.
- The melt temperature is defined with the `MELT TEMPERATURE` command line.

5.2.11 BCJ Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL BCJ
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    C1 = <real>c1
    C2 = <real>c2
    C3 = <real>c3
    C4 = <real>c4
    C5 = <real>c5
    C6 = <real>c6
    C7 = <real>c7
    C8 = <real>c8
    C9 = <real>c9
    C10 = <real>c10
    C11 = <real>c11
    C12 = <real>c12
    C13 = <real>c13
    C14 = <real>c14
    C15 = <real>c15
    C16 = <real>c16
    C17 = <real>c17
    C18 = <real>c18
    C19 = <real>c19
    C20 = <real>c20
    DAMAGE EXPONENT = <real>damage_exponent
    INITIAL ALPHA_XX = <real>alpha_xx
    INITIAL ALPHA_YY = <real>alpha_yy
    INITIAL ALPHA_ZZ = <real>alpha_zz
    INITIAL ALPHA_XY = <real>alpha_xy
```

```
        INITIAL ALPHA_YZ = <real>alpha_yz
        INITIAL ALPHA_XZ = <real>alpha_xz
        INITIAL KAPPA = <real>initial_kappa
        INITIAL DAMAGE = <real>initial_damage
        YOUNGS MODULUS FUNCTION = <string>ym_function_name
        POISSONS RATIO FUNCTION = <string>pr_function_name
        SPECIFIC HEAT = <real>specific_heat
        THETA OPT = <integer>theta_opt
        FACTOR = <real>factor
        RHO = <real>rho
        TEMP0 = <real>temp0
      END [PARAMETERS FOR MODEL BCJ]
    END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The BCJ plasticity model is a state-variable model for describing the finite deformation behavior of metals. It uses a multiplicative decomposition of the deformation gradient into elastic, volumetric plastic, and deviatoric parts. The model considers the natural configuration defined by this decomposition and its associated thermodynamics. The model incorporates strain rate and temperature sensitivity, as well as damage, through a yield-surface approach in which state variables follow a hardening-minus-recovery format.

Because the BCJ model has such an extensive list of parameters, we will not present the usual synopsis of parameter names with command lines. As with most other material models, the `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains. In addition, only two of the five elastic constants are required. The user should consult References 2, 3, and 4 for a description of the various parameters. Note that the parameters for the `SPECIFIC HEAT`, `THETA OPT`, `FACTOR`, `RHO`, and `TEMP0` command lines are used to accommodate changes to the model for heat generation due to plastic dissipation. For coupled solid/thermal calculations, the plastic dissipation rate is stored as a state variable and passed to a thermal code as a heat source term. For uncoupled calculations, temperature is stored as a state variable, and temperature increases due to plastic dissipation are calculated within the material model.

If temperature is provided from an external source, `theta_opt` is set to 0. If the temperature is calculated by the BCJ model, `theta_opt` is set to 1.

5.2.12 Power Law Creep

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL POWER_LAW_CREEP
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CREEP CONSTANT = <real>creep_constant
    CREEP EXPONENT = <real>creep_exponent
    THERMAL CONSTANT = <real>thermal_constant
  END [PARAMETERS FOR MODEL POWER_LAW_CREEP]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The power law creep model is a secondary creep model that can be used to model the time-dependent behavior of metals, brazes or solders at high homologous temperatures. It can also be used as a simple model for the time-dependent behavior of geologic materials such as salt.

In the power law creep model, the effective creep strain rate is related to the effective stress raised to a power

$$\dot{\bar{\varepsilon}}^{c} = A \bar{\sigma}^{m} \exp\left(\frac{-Q}{RT}\right), \tag{5.8}$$

where $\dot{\bar{\varepsilon}}^{c}$ is the effective creep strain rate, $\bar{\sigma}$ is the effective stress, A is the creep constant, m is the creep exponent, Q is the activation energy, R is the universal gas constant (1.987 cal/mole K), and T is the absolute temperature.

If an analysis is run isothermally, then the `THERMAL CONSTANT` is simply Q/RT for the given temperature. If the analysis is a thermal analysis, then the `THERMAL CONSTANT` is Q/R, and T is in general a function of space and time.

The command block for a power law creep material starts with the line:

```
BEGIN PARAMETERS FOR MODEL POWER_LAW_CREEP
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL POWER_LAW_CREEP]
```

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.
- The thermal strain option is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the YOUNGS MODULUS command line.
  - Poisson's ratio is defined with the POISSONS RATIO command line.
  - The bulk modulus is defined with the BULK MODULUS command line.
  - The shear modulus is defined with the SHEAR MODULUS command line.
  - Lambda is defined with the LAMBDA command line.
- The creep constant, A, in Equation (5.8) is defined with the CREEP CONSTANT command line.
- The creep exponent, m, in Equation (5.8) is defined with the CREEP EXPONENT command line.
- The thermal constant, the particular form depending on if the analysis is isothermal or not, is defined with the THERMAL CONSTANT command line.

Output variables available for this model are listed in Table 9.37.

5.2.13 Soil and Crushable Foam Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL SOIL_FOAM
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A0 = <real>const_coeff_yieldsurf
    A1 = <real>lin_coeff_yieldsurf
    A2 = <real>quad_coeff_yieldsurf
    PRESSURE CUTOFF = <real>pressure_cutoff
    PRESSURE FUNCTION = <string>function_press_volstrain
  END [PARAMETERS FOR MODEL SOIL_FOAM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The soil and crushable foam model is a plasticity model that can be used for modeling soil or crushable foam. Given the right input, the model is a Drucker-Prager model.

For the soil and crushable foam model, the yield surface is a surface of revolution about the hydrostat in principal stress space. A planar end cap is assumed for the yield surface so that the yield surface is closed. The yield stress σ_{yd} is specified as a polynomial in pressure p. The yield stress is given as:

$$\sigma_{yd} = a_0 + a_1 p + a_2 p^2 , \tag{5.9}$$

where p is positive in compression. The determination of the yield stress from Equation (5.9) places severe restrictions on the admissible values of a_0, a_1, and a_2. There are three valid cases for the yield surface. In the first case, there is an elastic–perfectly plastic deviatoric response, and the yield surface is a cylinder oriented along the hydrostat in principal stress space. In this case, a_0 is positive, and a_1 and a_2 are zero. In the second case, the yield surface is conical. A conical yield surface is obtained by setting a_2 to zero and using appropriate values for a_0 and a_1. In the third case, the yield surface has a parabolic shape. For the parabolic yield surface, all three coefficients

in Equation (5.9) are nonzero. The coefficients are checked to determine that a valid negative tensile-failure pressure can be derived based on the specified coefficients.

For the case of the cylindrical yield surface (e.g., $a_0 > 0$ and $a_1 = a_2 = 0$), there is no tensile-failure pressure. For the other two cases, the computed tensile-failure pressure may be too low. To handle the situations where there is no tensile-failure pressure or the tensile-failure pressure is too low, a pressure cutoff can be defined. If a pressure cutoff is defined, the tensile-failure pressure is the larger of the computed tensile-failure pressure and the defined cutoff pressure.

The plasticity theories for the volumetric and deviatoric parts of the material response are completely uncoupled. The volumetric response is computed first. The mean pressure p is assumed to be positive in compression, and a yield function ϕ_p is written for the volumetric response as:

$$\phi_p = p - f_p(\varepsilon_V) \ , \tag{5.10}$$

where $f_p(\varepsilon_V)$ defines the volumetric stress-strain curve for the pressure. The yield function ϕ_p determines the motion of the end cap along the hydrostat.

The command block for a soil and crushable foam material starts with the line:

```
BEGIN PARAMETERS FOR MODEL SOIL_FOAM
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL SOIL_FOAM]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The constant in the equation for the yield surface is defined with the `A0` command line.

- The coefficient for the linear term in the equation for the yield surface is defined with the `A1` command line.

- The coefficient for the quadratic term in the equation for the yield surface is defined with the `A2` command line.

- The user-defined tensile-failure pressure is defined with the `PRESSURE CUTOFF` command line.

- The pressure as a function of volumetric strain is defined with the function named on the `PRESSURE FUNCTION` command line.

For information about the soil and crushable foam model, see the PRONTO3D document listed as Reference 6. The soil and crushable foam model is the same as the soil and crushable foam model in PRONTO3D. The PRONTO3D model is based on a material model developed by Krieg [7]. The Krieg version of the soil and crushable foam model was later modified by Swenson and Taylor [8]. The soil and crushable foam model developed by Swenson and Taylor is the model in PRONTO3D and is also the shared model for Presto and Adagio.

Output variables available for this model are listed in Table 9.39.

5.2.14 Karagozian and Case Concrete Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
   DENSITY = <real>density_value
   BIOTS COEFFICIENT = <real>biots_value
   #
   # thermal strain option
   THERMAL STRAIN FUNCTION = <string>thermal_strain_function
   # or all three of the following
   THERMAL STRAIN X FUNCTION =
     <string>thermal_strain_x_function
   THERMAL STRAIN Y FUNCTION =
     <string>thermal_strain_y_function
   THERMAL STRAIN Z FUNCTION =
     <string>thermal_strain_z_function
   BEGIN PARAMETERS FOR MODEL KC_CONCRETE
      YOUNGS MODULUS = <real>youngs_modulus
      POISSONS RATIO = <real>poissons_ratio
      BULK MODULUS = <real>bulkmodulus
      SHEAR MODULUS = <real>shearmodulus
      COMPRESSIVE STRENGTH = <real>compressive_strength
      TENSILE STRENGTH = <real>tensile_strength
      LAMBDA = <real>lambda
      LAMBDAM = <real>lambda_m
      LAMBDAZ = <real>lambda_z
      SINGLE RATE ENHANCEMENT = <enum>TRUE|FALSE
      FRACTIONAL DILATANCY = <real>omega
      MAXIMUM AGGREGATE SIZE = <real>max_aggregate_size
      ONE INCH = <real>one_inch
      RATE SENSITIVITY FUNCTION = <string>rate_function_name
      PRESSURE FUNCTION = <string>pressure_function_name
      UNLOAD BULK MODULUS FUNCTION = <string>bulk_function_name
      HARDEN-SOFTEN FUNCTION = <string>harden_soften_function_name
   END PARAMETERS FOR MODEL KC_CONCRETE
END PROPERTY SPECIFICATION FOR MATERIAL name
```

The Karagozian & Case (or K&C) concrete model is an inelasticity model appropriate for approximating the constitutive behavior of concrete. Coupled with appropriate elements for capturing the embedded deformation of reinforcing steel, the K&C concrete model can be utilized effectively for simulating the mechanical response of reinforced concrete structures. The K&C model has several useful characteristics for estimating concrete response, including strain-softening capabilities, some degree of tensile response, and a nonlinear stress-strain characterization that robustly simulates the behavior of plain concrete. This model is described in detail in Reference 9.

The command block for the K&C concrete material model starts with the line:

```
BEGIN PARAMETERS FOR MODEL KC_CONCRETE
```

and terminates with the line:

```
END PARAMETERS FOR MODEL KC_CONCRETE
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.

In addition to these material moduli, the following constitutive parameters should be defined:

- The compressive strength for a uniaxial compression test is defined with the `COMPRESSIVE STRENGTH` command line.
- The tensile strength for the uniaxial tension test is defined with the `TENSILE STRENGTH` command line.
- The abscissa of the hardening/softening curve where this curve takes on the value of one is termed Lambda-M, and it is defined with the `LAMBDAM` command line (Reference 9, pg. B-3).
- The abscissa of the hardening/softening curve where this curve takes on the value of zero after its peak value has been attained is termed Lambda-Z, and it is defined with the `LAMBDAZ` command line. This parameter should satisfy LAMBDAZ > LAMBDAM (Reference 9, pg. B-3). This input is Sierra-specific, and differs from the previous PRONTO3D definitions.
- The `SINGLE RATE ENHANCEMENT` parameter indicates whether the rate enhancement of the model should be independent of the sign of the deformation. If this parameter is set to `TRUE`, the same enhancement function is used for both compression and tension. If it is set to `FALSE`, the enhancement function must assign values for both positive and negative values of strain rate (Reference 9, pg. B-5). This parameter is also Sierra-specific, and is different from the previous PRONTO3D definitions.

- The FRACTIONAL DILATANCY is an estimate of the size of the plastic volume strain increment relative to that corresponding to straining in the hydrostatic plane. This value normally ranges from 0.3 to 0.7, and a value of one-half is commonly utilized in practice.

- The MAXIMUM AGGREGATE SIZE parameter provides an estimate of the largest length dimension for the aggregate component of the concrete mix. The American Concrete Institute code [10] includes specifications for maximum aggregate size that are based on member depth and clear spacing between adjacent reinforcement elements.

- The parameter ONE INCH provides for conversion to units other than the pounds/inch system commonly used in U.S. concrete venues. This parameter should be set to the equivalent length in the system utilized for analysis, e.g., if centimeters are used, then this parameter is set to 2.54.

The following functions describe the evolution of material coefficients in this model:

- The function characterizing the enhancement of strength with strain rate is described via the RATE SENSITIVITY FUNCTION (Reference 9, pg. B-3).

Warning: The RATE SENSITIVITY FUNCTION command should be used with caution. The implementation appears to provide unconservative estimates of concrete strength in tension, and users are cautioned to provide rate sensitivity function values that have the value of 1.0 for positive (tensile) values of strain rate. These values correspond to no additional strength in tension due to strain rate, and are both physically realistic and properly conservative.

- The function describing the relationship between pressure and volumetric strain is described via the PRESSURE FUNCTION.

- The function characterizing the relationship between bulk modulus and volumetric strain during unloading is described via the UNLOAD BULK MODULUS FUNCTION.

- The function describing the hardening and softening functions function eta as a function of the material parameters lambda (see LAMBDAM and LAMBDAZ) is defined via the HARDEN-SOFTEN FUNCTION (Reference 9, pg. B-3).

5.2.15 Foam Plasticity Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    PHI = <real>phi
    SHEAR STRENGTH  = <real>shear_strength
    SHEAR HARDENING = <real>shear_hardening
    SHEAR EXPONENT  = <real>shear_exponent
    HYDRO STRENGTH  = <real>hydro_strength
    HYDRO HARDENING = <real>hydro_hardening
    HYDRO EXPONENT  = <real>hydro_exponent
    BETA = <real>beta
  END [PARAMETERS FOR MODEL FOAM_PLASTICITY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The foam plasticity model was developed to describe the response of porous elastic-plastic materials like closed-cell polyurethane foam to large deformation. Like solid metals, these foams can exhibit significant plastic deviatoric strains (permanent shape changes). Unlike metals, these foams can also exhibit significant plastic volume strains (permanent volume changes). The foam plasticity model is characterized by an initial yield surface that is an ellipsoid about the hydrostat.

When foams are compressed, they typically exhibit an initial elastic regime followed by a plateau regime in which the stress needed to compress the foam remains nearly constant. At some point in the compression process, the densification regime is reached, and the stress needed to compress the foam further begins to rapidly increase.

The foam plasticity model can be used to describe the response of metal foams and many closed-cell polymeric foams (including polyurethane, polystyrene bead, etc.) subjected to large deformation. This model is not appropriate for flexible foams that return to their undeformed shape after loads are removed.

The command block for a foam plasticity material starts with the line:

```
BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL FOAM_PLASTICITY]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The initial volume fraction of solid material in the foam, φ, is defined with the `PHI` command line. For example, solid polyurethane weighs 75 pounds per cubic foot (pcf); uncompressed 10 pcf polyurethane foam would have a φ of 0.133 = 10/75.
- The shear (deviatoric) strength of uncompressed foam is defined with the `SHEAR STRENGTH` command line.
- The shear hardening modulus for the foam is defined with the `SHEAR HARDENING` command line.
- The shear hardening exponent is defined with the `SHEAR EXPONENT` command line. The deviatoric strength is given by `(SHEAR STRENGTH) + (SHEAR HARDENING) * PHI**(SHEAR EXPONENT)`.
- The hydrostatic (volumetric) strength of the uncompressed foam is defined with the `HYDRO STRENGTH` command line.
- The hydrodynamic hardening modulus is defined with the `HYDRO HARDENING` command line.

- The hydrodynamic hardening exponent is defined with the HYDRO EXPONENT command line. The hydrostatic strength is given by (HYDRO STRENGTH) + (HYDRO HARDENING) * PHI**(HYDRO EXPONENT).

- The prescription for nonassociated flow, β, is defined with the BETA command line. When $\beta = 0.0$, the flow direction is given by the normal to the yield surface (associated flow). When $\beta = 1.0$, the flow direction is given by the stress tensor. Values between 0.0 and 0.95 are recommended.

Output variables available for this model are listed in Table 9.26.

5.2.16 Low Density Foam Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL LOW_DENSITY_FOAM
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A = <real>A
    B = <real>B
    C = <real>C
    NAIR  = <real>NAir
    P0  = <real>P0
    PHI = <real>Phi
  END [PARAMETERS FOR MODEL LOW_DENSITY_FOAM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The LOW_DENSITY_FOAM material model is a phenomenological model for low density polyurethane foams, where A, B, and C are material constants, NAIR is the mole fraction of air, P0 is the initial air pressure, and PHI is the volume fraction of solid material. The yield function for this model has the form

$$\bar{\sigma} = \mathrm{A}\left\langle \mathrm{I}_2' \right\rangle + \mathrm{B}\left(1.0 + \mathrm{C}\mathrm{I}_1\right), \tag{5.11}$$

where $\langle\rangle$ denotes the Heaviside step function, I_1 is the first invariant of the deviatoric strain, and I_2' is the second invariant of the strain.

For more information on the low density foam material model, see [11].

5.2.17 Elastic Three-Dimensional Orthotropic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  BEGIN PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC
    # general parameters (any two are required)
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    # required parameters
    YOUNGS MODULUS AA = <real>Eaa_value
    YOUNGS MODULUS BB = <real>Ebb_value
    YOUNGS MODULUS CC = <real>Ecc_value
    POISSONS RATIO AB = <real>NUab_value
    POISSONS RATIO BC = <real>NUbc_value
    POISSONS RATIO CA = <real>NUca_value
    SHEAR MODULUS AB = <real>Gab_value
    SHEAR MODULUS BC = <real>Gbc_value
    SHEAR MODULUS CA = <real>Gca_value
    COORDINATE SYSTEM = <string>coordinate_system_name
    DIRECTION FOR ROTATION = <real>1|2|3
    ALPHA = <real>alpha_in_degrees
    SECOND DIRECTION FOR ROTATION = <real>1|2|3
    SECOND ALPHA = <real>second_alpha_in_degrees
    THERMAL STRAIN AA FUNCTION = <string>ethaa_function_name
    THERMAL STRAIN BB FUNCTION = <string>ethbb_function_name
    THERMAL STRAIN CC FUNCTION = <string>ethcc_function_name
  END [PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The elastic three-dimensional orthotropic model describes the linear elastic response of an orthotropic material where the orientation of the principal material directions can be arbitrary. These principal axes are here denoted as A, B, and C. Thermal strains are also given along the principal material axes. The specification of these material axes is accomplished by selecting a user-defined coordinate system that can then be rotated twice about one of its current axes to give the final desired directions.

The command block for an elastic three-dimensional orthotropic material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC]
```

In the above command blocks all of the following are required inputs, including two of the five general elastic material constants:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The Youngs moduli corresponding to the principal material axes A, B, and C are given by the `YOUNGS MODULUS AA`, `YOUNGS MODULUS BB`, and `YOUNGS MODULUS CC` command lines.
- The Poisson's ratio defining the BB normal strain when the material is subjected only to AA normal stress is given by the `POISSONS RATIO AB` command line.
- The Poisson's ratio defining the CC normal strain when the material is subjected only to BB normal stress is given by the `POISSONS RATIO BC` command line.
- The Poisson's ratio defining the AA normal strain when the material is subjected only to CC normal stress is given by the `POISSONS RATIO CA` command line.
- The shear moduli for shear in the AB, BC, and CA planes are given by the `SHEAR MODULUS AB`, `SHEAR MODULUS BC`, and `SHEAR MODULUS CA` command lines, respectively.
- The specification of the principal material directions begins with the selection of a user-specified coordinate system given by the `COORDINATE SYSTEM` command line. This initial coordinate system can then be rotated twice to give the final material directions.
- The rotation of the initial coordinate system is defined using the `DIRECTION FOR ROTATION` and `ALPHA` command lines. The axis for rotation of the initial coordinate system is specified by the `DIRECTION FOR ROTATION` command line, while the angle of rotation is given by the `ALPHA` command line. This gives an intermediate specification of the material directions.
- The rotation of the intermediate coordinate system is defined using the `SECOND DIRECTION FOR ROTATION` and `SECOND ALPHA` command lines. The axis for rotation of the intermediate coordinate system is specified by the `SECOND DIRECTION FOR ROTATION` command line, while the angle of rotation is given by the `SECOND ALPHA` command line. The resulting coordinate system gives the final specification of the material directions.

- The thermal strain functions for normal thermal strains along the principal material directions are given by the `THERMAL STRAIN AA FUNCTION`, `THERMAL STRAIN BB FUNCTION`, and `THERMAL STRAIN CC FUNCTION` command lines.

See Reference 13 for more information about the elastic three-dimensional orthotropic model.

5.2.18 Wire Mesh Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL WIRE_MESH
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD FUNCTION = <string>yield_function
    TENSION = <real>tensile_strength
  END [PARAMETERS FOR MODEL WIRE_MESH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The wire mesh constitutive model was developed at Sandia National Laboratories to model layers of mesh for analyses of packaging for transportation of hazardous materials.

The wire mesh model decomposes the Cauchy stress tensor into its principal components. It then checks each of the principal stress to see whether they exceed the yield criterion. If a principal stress is negative, i.e. in compression, the yield condition is

$$\psi = \sigma - f(e_v) \tag{5.12}$$

The yield function for this model can be input by the user, and defines the yield strength as a function of the volumetric engineering strain, e_v.

If the principal stress is tensile, a cutoff value of the principal tensile stress is used, and the yield condition is

$$\psi = \sigma - \tau \tag{5.13}$$

The value for τ is given by the value of `TENSION`.

The command block for a wire mesh material starts with the line:

```
BEGIN PARAMETERS FOR MODEL WIRE_MESH
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL WIRE_MESH]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The yield function in compression is defined with the `YIELD FUNCTION` command line.
- The tensile strength is give by the `TENSION` command line.

Output variables available for this model are listed in Table 9.27.

More information on the model can be found in the report by Neilsen, et. al. [12]

5.2.19 Orthotropic Crush Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
    EX  = <real>modulus_x_pre_lockup
    EY  = <real>modulus_y_pre_lockup
    EZ  = <real>modulus_z_pre_lockup
    GXY = <real>shear_modulus_xy_pre_lockup
    GYZ = <real>shear_modulus_yz_pre_lockup
    GZX = <real>shear_modulus_zx_pre_lockup
    CRUSH XX = <string>stress_volume_xx_function_name
    CRUSH YY = <string>stress_volume_yy_function_name
    CRUSH ZZ = <string>stress_volume_zz_function_name
    CRUSH XY = <string>shear_stress_volume_xy_function_name
    CRUSH YZ = <string>shear_stress_volume_yz_function_name
    CRUSH ZX = <string>shear_stress_volume_zx_function_name
    VMIN = <real>lockup_volumetric_strain
    YOUNGS MODULUS = <real>youngs_modulus_post_lockup
    POISSONS RATIO = <real>poissons_ratio_post_lockup
    SHEAR MODULUS = <real>shear_modulus_post_lockup
    BULK MODULUS = <real>bulk_modulus_post_lockup
    LAMBDA = <real>lambda_post_lockup
    YIELD STRESS = <real>yield_stress_post_lockup
  END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The orthotropic crush model is an empirically based constitutive relation that is useful for modeling materials like metallic honeycomb and wood. This particular implementation follows the formulation of the metallic honeycomb model in DYNA3D [14]. The orthotropic crush model divides material behavior into three phases:

- orthotropic elastic,
- volumetric crush (partially compacted), and

- elastic–perfectly plastic (fully compacted).

The command block for an orthotropic crush material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH]
```

In the above command blocks:

- The uncompacted density of the material is defined with the `DENSITY` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- The initial directional modulus E_{xx} is defined with the `EX` command line.
- The initial directional modulus E_{yy} is defined with the `EY` command line.
- The initial directional modulus E_{zz} is defined with the `EZ` command line.
- The initial directional shear modulus G_{xy} is defined with the `GXY` command line.
- The initial directional shear modulus G_{yz} is defined with the `GYZ` command line.
- The initial directional shear modulus G_{zx} is defined with the `GZX` command line.
- The compressive volumetric strain at lockup or full compaction is defined with the `VMIN` command line.
- The directional stress σ_{xx} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH XX` command line.
- The directional stress σ_{yy} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH YY` command line.
- The directional stress σ_{zz} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH ZZ` command line.
- The directional stress σ_{xy} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH XY` command line.
- The directional stress σ_{yz} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH YZ` command line.
- The directional stress σ_{zx} as a function of the compressive volumetric strain is defined by the function referenced in the `CRUSH ZX` command line.

- Any two of the following elastic constants are required:

  - Young's modulus for the fully compacted state is defined with the `YOUNGS MODULUS` command line. This is the elastic–perfectly plastic value of Young's modulus.
  - Poisson's ratio for the fully compacted state is defined with the `POISSONS RATIO` command line. This is the elastic–perfectly plastic value of Poisson's ratio.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.

- The yield stress for the fully compacted state is defined with the `YIELD STRESS` command line. This is the elastic–perfectly plastic value of the yield stress.

Note that several of the command lines in this command block (those beginning with `CRUSH`) reference functions. These functions must be defined in the SIERRA scope. Output variables available for this model are listed in Table 9.34. For information about the orthotropic crush model, consult Reference 14.

5.2.20 Orthotropic Rate Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    MODULUS TTTT = <real>modulus_tttt
    MODULUS TTLL = <real>modulus_ttll
    MODULUS TTWW = <real>modulus_ttww
    MODULUS LLLL = <real>modulus_llll
    MODULUS LLWW = <real>modulus_llww
    MODULUS WWWW = <real>modulus_wwww
    MODULUS TLTL = <real>modulus_tltl
    MODULUS LWLW = <real>modulus_lwlw
    MODULUS WTWT = <real>modulus_wtwt
    TX = <real>tx
    TY = <real>ty
    TZ = <real>tz
    LX = <real>lx
    LY = <real>ly
    LZ = <real>lz
    MODULUS FUNCTION = <string>modulus_function_name
    RATE FUNCTION = <string>rate_function_name
    T FUNCTION = <string>t_function_name
    L FUNCTION = <string>l_function_name
    W FUNCTION = <string>w_function_name
    TL FUNCTION = <string>tl_function_name
    LW FUNCTION = <string>lw_function_name
    WT FUNCTION = <string>wt_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE]
```

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The orthotropic rate model extends the functionality of the orthotropic crush constitutive model described in Section 5.2.19. The orthotropic rate model has been developed to describe the behavior of an aluminum honeycomb subjected to large deformation. The orthotropic rate model, like the original orthotropic crush model, has six independent yield functions that evolve with volume strain. Unlike the orthotropic crush model, the orthotropic rate model has yield functions that also depend on strain rate. The orthotropic rate model also uses an orthotropic elasticity tensor with nine elastic moduli in place of the orthotropic elasticity tensor with six elastic moduli used in the orthotropic crush model. A honeycomb orientation capability is included with the orthotropic rate model that allows users to prescribe initial honeycomb orientations that are not aligned with the original global coordinate system.

The command block for an orthotropic rate material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- In the following list of elastic constants, only the elastic modulus (Young's modulus) is required for this model. If two elastic constants are supplied, the elastic constants will be completed. However, only the elastic modulus is used in this model.
  - Young's modulus for the fully compacted honeycomb is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio for the fully compacted state is defined with the `POISSONS RATIO` command line.
  - The bulk modulus for the fully compacted state is defined with the `BULK MODULUS` command line.
  - The shear modulus for the fully compacted state is defined with the `SHEAR MODULUS` command line.
  - Lambda for the fully compacted state is defined with the `LAMBDA` command line.

- The yield stress for the fully compacted honeycomb is defined with the YIELD STRESS command line.
- The nine elastic moduli for the orthotropic uncompacted honeycomb are defined with the MODULUS TTT, MODULUS TTLL, MODULUS TTWW, MODULUS LLLL, MODULUS LLWW, MODULUS WWWW, MODULUS TLTL, MODULUS LWLW, and MODULUS WTWT command lines. The T-direction is usually associated with the generator axis for the honeycomb. The L-direction is in the ribbon plane (plane defined by flat sheets used in reinforced honeycomb) and orthogonal to the generator axis. The W-direction is perpendicular to the ribbon plane.
- The components of a vector defining the T-direction of the honeycomb are defined by the TX, TY, and TZ command lines. The values of tx, ty, and tz are components of a vector in the global coordinate system that define the orientation of the honeycomb's T-direction (generator axis).
- The components of a vector defining the L-direction of the honeycomb are defined by the LX, LY, and LZ command lines. The values of lx, ly, and lz are components of a vector in the global coordinate system that define the orientation of the honeycomb's L-direction. *Caution*: The vectors T and L must be orthogonal.
- The function describing the variation in moduli with compaction is given by the MODULUS FUNCTION command line. The moduli vary continuously from their initial orthotropic values to isotropic values when full compaction is obtained.
- The function describing the change in strength with strain rate is given by the RATE FUNCTION command line. Note that all strengths are scaled with the multiplier obtained from this function.
- The function describing the T-normal strength of the honeycomb as a function of compressive volumetric strain is given by the T FUNCTION command line.
- The function describing the L-normal strength of the honeycomb as a function of compressive volumetric strain is given by the L FUNCTION command line.
- The function describing the W-normal strength of the honeycomb as a function of compressive volumetric strain is given by the W FUNCTION command line.
- The function describing the TL-normal strength of the honeycomb as a function of compressive volumetric strain is given by the TL FUNCTION command line.
- The function describing the LW-normal strength of the honeycomb as a function of compressive volumetric strain is given by the LW FUNCTION command line.
- The function describing the WT-normal strength of the honeycomb as a function of compressive volumetric strain is given by the WT FUNCTION command line.

Note that several of the command lines in this command block reference functions. These functions must be defined in the SIERRA scope. Output variables available for this model are listed in Table 9.35.

5.2.21 Elastic Laminate Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC_LAMINATE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A11 = <real>a11_value
    A12 = <real>a12_value
    A16 = <real>a16_value
    A22 = <real>a22_value
    A26 = <real>a26_value
    A66 = <real>a66_value
    A44 = <real>a44_value
    A45 = <real>a45_value
    A55 = <real>a55_value
    B11 = <real>b11_value
    B12 = <real>b12_value
    B16 = <real>b16_value
    B22 = <real>b22_value
    B26 = <real>b26_value
    B66 = <real>b66_value
    D11 = <real>d11_value
    D12 = <real>d12_value
    D16 = <real>d16_value
    D22 = <real>d22_value
    D26 = <real>d26_value
    D66 = <real>d66_value
    COORDINATE SYSTEM = <string>coord_sys_name
    DIRECTION FOR ROTATION = 1|2|3
    ALPHA = <real>alpha_value_in_degrees
    THETA = <real>theta_value_in_degrees
    NTH11 FUNCTION = <string>nth11_function_name
    NTH22 FUNCTION = <string>nth22_function_name
    NTH12 FUNCTION = <string>nth12_function_name
    MTH11 FUNCTION = <string>mth11_function_name
    MTH22 FUNCTION = <string>mth22_function_name
    MTH12 FUNCTION = <string>mth12_function_name
  END [PARAMETERS FOR MODEL ELASTIC_LAMINATE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The elastic laminate model can be used to describe the overall linear elastic response of layered shells. The response of each layer is pre-integrated through the thickness under an assumed vari-

ation of strain through the thickness. That is, the user inputs laminate stiffness matrices directly, and the overall response is calculated appropriately. This model allows the user to input laminate stiffness matrices that are consistent with a state of generalized plane stress for each layer. Each layer can be orthotropic with a unique orientation. This model is primarily intended for capturing the response of fiber-reinforced laminated composites. The user inputs the laminate stiffness matrices calculated with respect to a chosen coordinate system and then specifies this coordinate system's definition relative to the global coordinate system. Thermal stresses are handled via the input of thermal-force and thermal-force-couple resultants for the laminate as a whole. At present, the user cannot get layer stresses out from this material model. However, the overall section-force and force-couple resultants can be computed from available output. The details of this model are described in References 15 and 16.

The command block for an elastic laminate material starts with the line:

```
BEGIN PARAMETERS FOR MODEL ELASTIC_LAMINATE
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL ELASTIC_LAMINATE]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- Two of the five elastic constants must be defined with the `YOUNGS MODULUS`, `POISSONS RATIO`, `SHEAR MODULUS`, `BULK MODULUS`, or `LAMBDA` commands
- The elastic constants are unrelated to the laminate stiffness matrix. They are used along with the shell section thickness (defined in the shell section command block, see section 6.2.4) to calculate drilling and hourglass stiffnesses. Otherwise these values have no physical significance in the elastic laminate material model.
- The extensional stiffnesses are defined with the A_{ij} command lines, where the values of ij are 11, 12, 16, 22, 26, 66, 44, 45, and 55.
- The coupling stiffnesses are defined with the B_{ij} command lines, where the values of ij are 11, 12, 16, 22, 26, and 66.
- The bending stiffnesses are defined with the D_{ij} command lines, where the values of ij are 11, 12, 16, 22, 26, and 66.
- The initial laminate coordinate system is defined with the `COORDINATE SYSTEM` command line.
- The rotation of the initial laminate coordinate system is defined with the `DIRECTION FOR ROTATION` and `ALPHA` command lines. The axis of initial laminate coordinate system is specified by the `DIRECTION FOR ROTATION` command line, while the angle of rotation is given by the `ALPHA` command line. This produces an intermediate laminate coordinate system that is then projected onto the surface of each shell element.

- The projected intermediate laminate coordinate system is rotated about the element normal by angle theta, which is specified by the `THETA` command line.
- The thermal-force resultants are defined by functions that are referenced on the `NTH11 FUNCTION`, `NTH22 FUNCTION`, and `NTH12 FUNCTION` command lines.
- The thermal-force-couple resultants are defined by functions that are referenced on the `MTH11 FUNCTION`, `MTH22 FUNCTION`, and `MTH12 FUNCTION` command lines.

5.2.22 Fiber Membrane Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL FIBER_MEMBRANE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CORD DENSITY = <real>cord_density
    CORD DIAMETER = <real>cord_diameter
    MATRIX DENSITY = <real>matrix_density
    TENSILE TEST FUNCTION = <string>test_function_name
    PERCENT CONTINUUM = <real>percent_continuum
    EPL = <real>epl
    AXIS X = <real>axis_x
    AXIS Y = <real>axis_y
    AXIS Z = <real>axis_z
    MODEL = <string>RECTANGULAR
    STIFFNESS SCALE = <real>stiffness_scale
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL FIBER_MEMBRANE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The fiber membrane model is used for modeling membranes that are reinforced with unidirectional fibers. Through the use of a non-zero `PERCENT CONTINUUM`, a background isotropic material response can also be incorporated and is added in a manner such that the response in the fiber direction is unchanged. The fiber membrane model can be used in both Presto and Adagio. When the fiber membrane model is used in Adagio, the model can be used with or without the control-stiffness option in Adagio's multilevel solver. The control-stiffness option is implemented via the `CONTROL STIFFNESS` command block and is discussed in Chapter 3.5. If the control-stiffness option is activated in Adagio, the response in the fiber direction is softened by lowering the fiber response. In all cases, the final material behavior that is used for equilibrium corresponds to the real material response. When the fiber membrane model is used in Presto, the fiber scaling, which is controlled by the `STIFFNESS SCALE` command line, is ignored.

The command block for a fiber membrane material starts with the line:

```
BEGIN PARAMETERS FOR MODEL FIBER_MEMBRANE
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL FIBER_MEMBRANE]
```

In the above command blocks, the following definitions are applicable. Usage requirements are identified both in this list of definitions and in the discussion that follows the list.

- The density of the material is defined with the `DENSITY` command line. This command line should be included, but its value will be recomputed (and hence replaced) if the `CORD DENSITY`, `CORD DIAMETER`, and `MATRIX DENSITY` command lines are specified.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required. These are used to compute values for the elastic preconditioner only.
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The density of the fibers is defined by the `CORD DENSITY` command line. This command line is optional. See the usage discussion below.
- The diameter of the fibers is defined by the `CORD DIAMETER` command line. This command line is optional. See the usage discussion below.
- The density of the matrix is defined by the `MATRIX DENSITY` command line. This command line is optional. See the usage discussion below.
- The `TENSILE TEST FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the fiber force versus strain data. This command line must be included.
- The fractional fiber stiffness to use in defining the background isotropic response is given by the `PERCENT CONTINUUM` command line. This command line must be included.
- The number of fibers per unit length is defined by the `EPL` command line. This command line must be included.

- The components of the vector defining the initial fiber direction is given by the AXIS X, AXIS Y, and AXIS Z command lines. These command lines must be included. See the usage discussion below.

- The coordinate system for specifying the fiber orientation is given by the MODEL command line. Only the option RECTANGULAR is available in this release. This command line must be included. See the usage discussion below.

- The fiber scaling is specified by the STIFFNESS SCALE command line. If the control-stiffness option is used in Adagio, this command line must be included. When the fiber membrane model is used in Presto, this command line is ignored.

- The reference strain is defined with the REFERENCE STRAIN command line. This command line is optional for Adagio and is not used in Presto. If the control-stiffness option is used in Adagio, this command line may be included. See the usage discussion below.

Certain command lines in the PARAMETERS FOR MODEL FIBER_MEMBRANE command block also have interdependencies or other factors that may impact their usage in Presto and Adagio, as discussed below.

The CORD DENSITY, CORD DIAMETER, and MATRIX DENSITY command lines are optional. When included, these three command lines are used for computation of the correct density corresponding to the fibers, the number of fibers per unit length, and the chosen matrix. When these three command lines are not included, the density is taken as that specified by the DENSITY command line.

The AXIS X, AXIS Y, and AXIS Z command lines must be specified if the value for the MODEL command line is RECTANGULAR. Currently, these axis-related command lines must be specified.

Specifying a reference strain (via the REFERENCE STRAIN command line) implies the use of strains for measuring part of the control-stiffness material constraint violation in Adagio. If this command line is not present, the material constraint violation is determined by comparing the change in the scaled fiber force over the current model problem.

5.2.23 Fiber Shell Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL FIBER_SHELL
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CORD DENSITY = <real>cord_density
    CORD DIAMETER = <real>cord_diameter
    MATRIX DENSITY = <real>matrix_density
    TENSILE TEST FUNCTION = <string>test_function_name
    PERCENT CONTINUUM = <real>percent_continuum
    EPL = <real>epl
    AXIS X = <real>axis_x
    AXIS Y = <real>axis_y
    AXIS Z = <real>axis_z
    MODEL = <string>RECTANGULAR
    ALPHA1 = <real>alpha1
    ALPHA2 = <real>alpha2
    ALPHA3 = <real>alpha3
    STIFFNESS SCALE = <real>stiffness_scale
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL FIBER_SHELL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The fiber shell model is used for modeling shells that are reinforced with unidirectional fibers. The input parameters for this model are generally identical to those for the fiber membrane, and its membrane response should be nearly identical to that of the fiber membrane. The salient difference between the fiber membrane and fiber shell models is that the latter includes the bending and torsional stiffnesses arising from the fiber thickness, where the former neglects these responses in order to gain a purely two-dimensional physical response. These additional shell stiffnesses for the fiber shell model can be modified via the three input parameters `ALPHA1`, `ALPHA2`, and `ALPHA3`, as described below.

The fiber shell model currently only works with the analytically-integrated four-node shell element. For this element, it is essential to allocate space for storage of internal material parameters, and this is accomplished by requiring the specification of a single integration point for the shell. The thickness of the shell element is taken to be identical to the cord diameter, and this consistency condition should be explicitly provided in the input data. All other relevant shell element data fields, *e.g.*, drilling stiffness factors, can be used as they would be for a conventional material model, *e.g.*, isotropic linear elasticity.

The command block for a fiber shell material starts with the line

```
BEGIN PARAMETERS FOR MODEL FIBER_SHELL
```

and terminates with the line

```
END [PARAMETERS FOR MODEL FIBER_SHELL]
```

In the above command blocks, the various material definitions are identical to those specified for a fiber membrane material. The additional effects of plate-bending response are mediated by the three parameters `ALPHA1`, `ALPHA2`, and `ALPHA3`, and these three variant parameters are described below.

- The parameters `ALPHA1`, `ALPHA2`, and `ALPHA3` provide a means to modify the plate-bending stiffnesses computed by a fiber shell element, so that a factored bending response can be obtained. For example, to attenuate the effects of plate bending and torsion, these three parameters could be set to small values such as 0.001, in order to capture only the membrane response of the material. In practice, setting these parameters to such small values can degrade the conditioning of the finite element equation set, and setting them to zero risks generating a singular stiffness matrix, but the use of these three parameters can be deduced from this simple example. Setting all three of these parameters to 1.0 provides a consistent computation of plate bending and plate membrane effects.

  - The `ALPHA1` command line provides a factor multiplying the bending response of the shell in the fiber direction, *i.e.*, setting this parameter equal to unity captures all the bending response of the fiber shell in the fiber direction.
  - The `ALPHA2` command line provides a factor multiplying the bending response of the shell in the perpendicular-to-fiber direction, *i.e.*, setting this parameter equal to unity captures all the bending response of the fiber shell in the direction parallel to the plane of the shell, but perpendicular to the fiber direction.
  - The `ALPHA3` command line provides a factor multiplying the plate torsional response of the shell, *i.e.*, setting this parameter equal to unity captures all the relevant out-of-plane torsional response.

Certain command lines in the `PARAMETERS FOR MODEL FIBER_SHELL` command block also have interdependencies, and these are described in the fiber membrane material, so they need not be replicated here.

5.2.24 Incompressible Solid Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    K SCALING = <real>k_scaling
    2G SCALING = <real>2g_scaling
    TARGET E = <real>target_e
    MAX POISSONS RATIO = <real>max_poissons_ratio
    REFERENCE STRAIN = <real>reference_strain
    SCALING FUNCTION = <string>scaling_function_name
  END [PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The incompressible solid model is a variation of the elastic model and can be used in both Presto and Adagio. In Adagio, the incompressible solid model is used with the control-stiffness option in the multilevel solver. The control-stiffness option is implemented via the `CONTROL STIFFNESS` command block and is discussed in Chapter 3.5. The model is used to model nearly incompressible materials where Poisson's ratio, ν, $\approx$ 0.5. In the course of solving a series of model problems in Adagio, the material response from this model incorporates scaling the bulk and/or shear behaviors to yield a material response that is more amenable to solution using Adagio's conjugate gradient solver. The final material behavior that is calculated corresponds to the actual moduli that are specified. When this model is used in Presto, the material scalings are ignored, and the model behaves like a linear elastic model.

The command block for an incompressible solid material starts with the line:

```
BEGIN PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID]
```

In the above command blocks, the following definitions are applicable. Usage requirements are identified both in this list of definitions and in the discussion that follows the list.

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- The `thermal strain option` is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required to define the unscaled material response:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following material-scaling command lines are used only in Adagio:
  - The nominal bulk scaling is defined with the `K SCALING` command line. This command line is optional. See the usage discussion below.
  - The nominal shear scaling is defined with the `2G SCALING` command line. This command line is optional. See the usage discussion below.
  - The target Young's modulus is defined with the `TARGET E` command line. This command line is optional. See the usage discussion below.
  - The maximum Poisson's ratio is defined with the `MAX POISSONS RATIO` command line. This command line is optional. See the usage discussion below.
  - The reference strain is defined with the `REFERENCE STRAIN` command line. This command line is optional. See the usage discussion below.
  - The `SCALING FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the time dependent scaling to be applied. This command line is optional. See the usage discussion below.

As noted previously, only two of the elastic constants are required to define the unscaled material response. This requirement applies to use of the incompressible solid model in Presto and in Adagio. Further, all the material-scaling command lines are only used in Adagio.

Several options exist for defining the bulk and/or shear scalings that can be used with the multilevel solver in Adagio.

- Option 1: You can provide the scalings directly by including both of the K SCALING and 2G SCALING command lines or either of them. When both command lines are input, the user-specified values for their parameters will be used. If only the K SCALING command line is input, the bulk scaling is as specified in the k_scaling parameter, and the value of the shear scaling parameter, 2g_scaling, is set to 1.0. On the other hand, if only the 2G SCALING command line is input, then the shear scaling is as specified in the 2g_scaling parameter, but the value of the bulk-scaling parameter, k_scaling, is not set to 1.0. Instead, the bulk scaling is determined by computing a scaled bulk modulus from the scaled shear modulus and a (scaled) Poisson's ratio of 0.3. Then, the bulk scaling is determined simply as the ratio of the scaled bulk modulus to the actual bulk modulus.
- Option 2: You can specify either or both of the TARGET E and MAX POISSONS RATIO command lines to define the scalings. If only the TARGET E command line is included, the bulk and shear scalings are computed by first finding scaled moduli using the value of the target_e parameter along with a (scaled) Poisson's ratio of 0.3. The bulk and shear scalings are then determined as the ratio of the appropriate scaled to unscaled modulus. If only the MAX POISSONS RATIO command line is included, the shear scaling is set to 1.0, and the bulk scaling is computed by first calculating a scaled bulk modulus from the actual shear modulus and the value of the max_poissons_ratio parameter. The bulk scaling is then calculated simply as the ratio of the scaled bulk modulus to the actual bulk modulus. If both the TARGET E and MAX POISSONS RATIO command lines are included, the bulk scaling (and shear scaling) is determined from the ratio of the bulk scaled modulus (and shear scaled modulus) computed using the values of the target_e and max_poissons_ratio parameters to the unscaled bulk (and shear) modulus.
- Option 3: You can choose not to include any of the K SCALING, 2G SCALING, TARGET E, and MAX POISSONS RATIO command lines. In such case, the shear scaling is set to 1.0, and the bulk scaling is computed as the ratio of the scaled bulk modulus coming from the real shear modulus and a (scaled) Poisson's ratio of 0.3 to the actual bulk modulus.

The function referenced by the value of the parameter scaling_function_name in the SCALING FUNCTION command line can be used to modify the bulk and shear scalings in solution time. The actual scalings used are computed by taking the scalings specified by the parameter values in the K SCALING, 2G SCALING, TARGET E, and MAX POISSONS RATIO command lines and simply multiplying them by the function value at the specified solution time. If the SCALING FUNCTION command line is not included, the bulk and shear scalings are fixed in time.

The REFERENCE STRAIN command line supplies a value for the reference strain that is used to create a normalized material constraint violation based on strains. Specifying a reference strain implies the use of strains for measuring the material constraint violation (or part of the control-stiffness error in Adagio). Otherwise, the material constraint violation is determined by using the change in the scaled stress response over the current model problem.

5.2.25 Mooney-Rivlin Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL MOONEY_RIVLIN
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    C10 = <real>c10
    C01 = <real>c01
    C10 FUNCTION = <string>c10_function_name
    C01 FUNCTION = <string>c01_function_name
    BULK FUNCTION = <string>bulk_function_name
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
    MAX POISSONS RATIO = <real>max_poissons_ratio(0.5)
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL MOONEY_RIVLIN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

Mooney-Rivlin is a hyperelastic model that is used to model rubber. The Mooney-Rivlin model incorporates temperature-dependent material moduli and can be used in both Presto and Adagio. When the model is used in Adagio, it can be used with or without the control-stiffness option in Adagio's multilevel solver. The control-stiffness option is implemented via the CONTROL STIFFNESS command block and is discussed in Chapter 3.5. The model is used to model nearly incompressible materials where Poisson's ratio, ν, ≈ 0.5. In the course of solving a series of model problems in Adagio, the material response from this model incorporates scaling the bulk and/or shear behaviors to yield a material response that is more amenable to solution using Adagio's conjugate gradient solver. The final material behavior that is calculated corresponds to the actual moduli that are specified. When this model is used in Presto, the material scalings are ignored.

The command block for a Mooney-Rivlin material starts with the line:

```
BEGIN PARAMETERS FOR MODEL MOONEY_RIVLIN
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL MOONEY_RIVLIN]
```

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.
- Any two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the YOUNGS MODULUS command line.
  - Poisson's ratio is defined with the POISSONS RATIO command line.
  - The bulk modulus is defined with the BULK MODULUS command line.
  - The shear modulus is defined with the SHEAR MODULUS command line.
  - Lambda is defined with the LAMBDA command line.
- The nominal value for C10 is defined with the C10 command line. This command line is required. See the usage discussion below.
- The nominal value for C01 is defined with the C01 command line. This command line is required. See the usage discussion below.
- The C10 FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the temperature dependence of the C10 material parameter. This command line is optional. If it is not present, there is no temperature dependence in the C10 parameter. See the usage discussion below.
- The C01 FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the temperature dependence of the C01 material parameter. This command line is optional. If it is not present, there is no temperature dependence in the C01 parameter. See the usage discussion below.
- The BULK FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the temperature dependence of the bulk modulus. This command line is optional. If it is not present, there is no temperature dependence in the bulk modulus. See the usage discussion below.
- The THERMAL EXPANSION FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the linear thermal expansion as function of temperature. This command line is optional. If it is not present, there is no thermal expansion. See the usage discussion below.
- The following material-scaling command lines are used only in Adagio:
  - The target Young's modulus is defined with the TARGET E command line. This command line is optional. See the usage discussion below.

- The `TARGET E FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the time variation of the target Young's modulus. This command line is optional. If it is not present, there is no time dependence in the Target E parameter. See the usage discussion below.
- The maximum Poisson's ratio is defined with the `MAX POISSONS RATIO` command line. This command line is optional and will default to 0.5 if not specified. See the usage discussion below.
- The reference strain is defined with the `REFERENCE STRAIN` command line. This command line is optional. See the usage discussion below.

As noted previously, only two of the elastic constants are required to define the unscaled bulk behavior. Together, the values for C10 and C01 determine the shear behavior, and thus the `C10` and `C01` command lines must be included in this model.

The command lines for functions that specify the temperature dependence of C10, C01, and bulk modulus are optional, e.g., the `C10 FUNCTION`, `C01 FUNCTION` and `BULK FUNCTION` command lines. If these command lines are not included, their corresponding material parameters are taken to be independent of temperature. Mooney-Rivlin, like other material models, allows for the specification of thermal strain behavior within the material model itself, via the `THERMAL EXPANSION FUNCTION` command line. This command line, like the other "function-type" command lines in this model requires that a function associated with the name be defined in the SIERRA scope.

The bulk and shear scalings that can be used with the multilevel solver in Adagio are specified via a combination of the `TARGET E`, `TARGET E FUNCTION`, and `MAX POISSONS RATIO` command lines. If the `TARGET E` command line is not included (and the `MAX POISSONS RATIO` command line is included), the shear scaling is set to 1.0, and the bulk scaling is determined from the ratio of the scaled bulk modulus to its unscaled value, where the scaled bulk modulus is computed using the value of the `max_poissons_ratio` parameter along with the unscaled initial shear modulus that is determined from the value of the parameters specified in the `C10` and `C01` command lines. On the other hand, if both the `TARGET E` command line and the `MAX POISSONS RATIO` command line are included, bulk and shear scaling values are computed using scaled moduli that are calculated from the `target_e` and `max_poissons_ratio` parameter values.

Including the `TARGET E FUNCTION` command line allows time-dependent bulk and shear scaling to be used. If this command line is not specified, the bulk and shear scalings remain constant in solution time. If the command line is specified, the target Young's modulus that is used for computing the scaled moduli is multiplied by the function value.

The `REFERENCE STRAIN` command line supplies a value for the reference strain used to create a normalized material constraint violation that is based on strains. Specifying a reference strain implies the use of strains for measuring the material constraint violation (or part of the control-stiffness error in Adagio). Otherwise, the material constraint violation is determined using the change in the scaled stress response over the current model problem.

Brief documentation on the theoretical basis for the Mooney-Rivlin model is given in Reference 17.

5.2.26 NLVE 3D Orthotropic Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL NLVE_3D_ORTHOTROPIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    COORDINATE SYSTEM = <string>coordinate_system_name
    DIRECTION FOR ROTATION = <real>1|2|3
    ALPHA = <real>alpha_in_degrees
    SECOND DIRECTION FOR ROTATION = <real>1|2|3
    SECOND ALPHA = <real>second_alpha_in_degrees
    FICTITIOUS LOGA FUNCTION = <string>fict_loga_function_name
    FICTITIOUS LOGA SCALE FACTOR = <real>fict_loga_scale_factor
    # In each of the five ``PRONY'' command lines and in
    # the RELAX TIME command line, the value of i can be from
    # 1 through 30
    1PSI PRONY <integer>i = <real>psi1_i
    2PSI PRONY <integer>i = <real>psi2_i
    3PSI PRONY <integer>i = <real>psi3_i
    4PSI PRONY <integer>i = <real>psi4_i
    5PSI PRONY <integer>i = <real>psi5_i
    RELAX TIME <integer>i = <real>tau_i
    REFERENCE TEMP = <real>tref
    REFERENCE DENSITY = <real>rhoref
    WLF C1 = <real>wlf_c1
    WLF C2 = <real>wlf_c2
    B SHIFT CONSTANT = <real>b_shift
    SHIFT REF VALUE = <real>shift_ref
    WWBETA 1PSI = <real>wwb_1psi
    WWTAU 1PSI = <real>wwt_1psi
    WWBETA 2PSI = <real>wwb_2psi
    WWTAU 2PSI = <real>wwt_2psi
    WWBETA 3PSI = <real>wwb_3psi
    WWTAU 3PSI = <real>wwt_3psi
    WWBETA 4PSI = <real>wwb_4psi
    WWTAU 4PSI = <real>wwt_4psi
    WWBETA 5PSI = <real>wwb_5psi
    WWTAU 5PSI = <real>wwt_5psi
    DOUBLE INTEG FACTOR = <real>dble_int_fac
    REF RUBBERY HCAPACITY = <real>hcapr
    REF GLASSY HCAPACITY = <real>hcapg
```

```
GLASS TRANSITION TEM = <real>tg
REF GLASSY C11 = <real>c11g
REF RUBBERY C11 = <real>c11r
REF GLASSY C22 = <real>c22g
REF RUBBERY C22 = <real>c22r
REF GLASSY C33 = <real>c33g
REF RUBBERY C33 = <real>c33r
REF GLASSY C12 = <real>c12g
REF RUBBERY C12 = <real>c12r
REF GLASSY C13 = <real>c13g
REF RUBBERY C13 = <real>c13r
REF GLASSY C23 = <real>c23g
REF RUBBERY C23 = <real>c23r
REF GLASSY C44 = <real>c44g
REF RUBBERY C44 = <real>c44r
REF GLASSY C55 = <real>c55g
REF RUBBERY C55 = <real>c55r
REF GLASSY C66 = <real>c66g
REF RUBBERY C66 = <real>c66r
REF GLASSY CTE1 = <real>cte1g
REF RUBBERY CTE1 = <real>cte1r
REF GLASSY CTE2 = <real>cte2g
REF RUBBERY CTE2 = <real>cte2r
REF GLASSY CTE3 = <real>cte3g
REF RUBBERY CTE3 = <real>cte3r
LINEAR VISCO TEST = <real>lvt
T DERIV GLASSY C11 = <real>dc11gdT
T DERIV RUBBERY C11 = <real>dc11rdT
T DERIV GLASSY C22 = <real>dc22gdT
T DERIV RUBBERY C22 = <real>dc22rdT
T DERIV GLASSY C33 = <real>dc33gdT
T DERIV RUBBERY C33 = <real>dc33rdT
T DERIV GLASSY C12 = <real>dc12gdT
T DERIV RUBBERY C12 = <real>dc12rdT
T DERIV GLASSY C13 = <real>dc13gdT
T DERIV RUBBERY C13 = <real>dc13rdT
T DERIV GLASSY C23 = <real>dc23gdT
T DERIV RUBBERY C23 = <real>dc23rdT
T DERIV GLASSY C44 = <real>dc44gdT
T DERIV RUBBERY C44 = <real>dc44rdT
T DERIV GLASSY C55 = <real>dc55gdT
T DERIV RUBBERY C55 = <real>dc55rdT
T DERIV GLASSY C66 = <real>dc66gdT
T DERIV RUBBERY C66 = <real>dc66rdT
T DERIV GLASSY CTE1 = <real>dcte1gdT
T DERIV RUBBERY CTE1 = <real>dcte1rdT
T DERIV GLASSY CTE2 = <real>dcte2gdT
```

```
T DERIV RUBBERY CTE2 = <real>dcte2rdT
T DERIV GLASSY CTE3 = <real>dcte3gdT
T DERIV RUBBERY CTE3 = <real>dcte3rdT
T DERIV GLASSY HCAPACITY = <real>dhcapgdT
T DERIV RUBBERY HCAPACITY = <real>dhcaprdT
REF PSIC = <real>psic_ref
T DERIV PSIC = <real>dpsicdT
T 2DERIV PSIC = <real>d2psicdT2
PSI EQ 2T = <real>psitt
PSI EQ 3T = <real>psittt
PSI EQ 4T = <real>psitttt
PSI EQ XX 11 = <real>psiXX11
PSI EQ XX 22 = <real>psiXX22
PSI EQ XX 33 = <real>psiXX33
PSI EQ XX 12 = <real>psiXX12
PSI EQ XX 13 = <real>psiXX13
PSI EQ XX 23 = <real>psiXX23
PSI EQ XX 44 = <real>psiXX44
PSI EQ XX 55 = <real>psiXX55
PSI EQ XX 66 = <real>psiXX66
PSI EQ XXT 11 = <real>psiXXT11
PSI EQ XXT 22 = <real>psiXXT22
PSI EQ XXT 33 = <real>psiXXT33
PSI EQ XXT 12 = <real>psiXXT12
PSI EQ XXT 13 = <real>psiXXT13
PSI EQ XXT 23 = <real>psiXXT23
PSI EQ XXT 44 = <real>psiXXT44
PSI EQ XXT 55 = <real>psiXXT55
PSI EQ XXT 66 = <real>psiXXT66
PSI EQ XT 1 = <real>psiXT1
PSI EQ XT 2 = <real>psiXT2
PSI EQ XT 3 = <real>psiXT3
PSI EQ XTT 1 = <real>psiXTT1
PSI EQ XTT 2 = <real>psiXTT2
PSI EQ XTT 3 = <real>psiXTT3
REF PSIA 11 = <real>psiA11
REF PSIA 22 = <real>psiA22
REF PSIA 33 = <real>psiA33
REF PSIA 12 = <real>psiA12
REF PSIA 13 = <real>psiA13
REF PSIA 23 = <real>psiA23
REF PSIA 44 = <real>psiA44
REF PSIA 55 = <real>psiA55
REF PSIA 66 = <real>psiA66
T DERIV PSIA 11 = <real>dpsiA11dT
T DERIV PSIA 22 = <real>dpsiA22dT
T DERIV PSIA 33 = <real>dpsiA33dT
```

```
      T DERIV PSIA 12 = <real>dpsiA12dT
      T DERIV PSIA 13 = <real>dpsiA13dT
      T DERIV PSIA 23 = <real>dpsiA23dT
      T DERIV PSIA 44 = <real>dpsiA44dT
      T DERIV PSIA 55 = <real>dpsiA55dT
      T DERIV PSIA 66 = <real>dpsiA66dT
      REF PSIB 1 = <real>psiB1
      REF PSIB 2 = <real>psiB2
      REF PSIB 3 = <real>psiB3
      T DERIV PSIB 1 = <real>dpsiB1dT
      T DERIV PSIB 2 = <real>dpsiB2dT
      T DERIV PSIB 3 = <real>dpsiB3dT
      PSI POT TT = <real>psipotTT
      PSI POT TTT = <real>psipotTTT
      PSI POT TTTT = <real>psipotTTTT
      PSI POT XT 1 = <real>psipotXT1
      PSI POT XT 2 = <real>psipotXT2
      PSI POT XT 3 = <real>psipotXT3
      PSI POT XTT 1 = <real>psipotXTT1
      PSI POT XTT 2 = <real>psipotXTT2
      PSI POT XTT 3 = <real>psipotXTT3
      PSI POT XXT 11 = <real>psipotXXT11
      PSI POT XXT 22 = <real>psipotXXT22
      PSI POT XXT 33 = <real>psipotXXT33
      PSI POT XXT 12 = <real>psipotXXT12
      PSI POT XXT 13 = <real>psipotXXT13
      PSI POT XXT 23 = <real>psipotXXT23
      PSI POT XXT 44 = <real>psipotXXT44
      PSI POT XXT 55 = <real>psipotXXT55
      PSI POT XXT 66 = <real>psipotXXT66
    END [PARAMETERS FOR MODEL NLVE_3D_ORTHOTROPIC]
  END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The NLVE three-dimensional orthotropic model is a nonlinear viscoelastic orthotropic continuum model that describes the behavior of fiber-reinforced polymer-matrix composites. In addition to being able to model the linear elastic and linear viscoelastic behaviors of such composites, it also can capture both “weak” and “strong” nonlinear viscoelastic effects such as stress dependence of the creep compliance and viscoelastic yielding. This model can be used in both Presto and Adagio.

Because the NLVE model is still under active development and also because it has an extensive list of command lines, we have not followed the typical approach in documenting this model.

5.2.27 Stiff Elastic

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL STIFF_ELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    SCALE FACTOR = <real>scale_factor
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL STIFF_ELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The stiff elastic model is a variation of the isotropic elastic model. The stiff elastic model can be used in both Presto and Adagio. When the model is used in Adagio, it is typically used with the control-stiffness option in Adagio's multilevel solver. The control-stiffness option is implemented via the `CONTROL STIFFNESS` command block and is discussed in Chapter 3.5. The stiff elastic model is used to lower the stiffness of the bulk and shear behaviors of relatively stiff materials to yield a material response more amenable to solution using Adagio's conjugate gradient solver. The final material behavior that is calculated corresponds to the actual moduli that are specified. When this model is used in Presto, the material scalings are ignored.

The command block for a stiff elastic material starts with the line:

```
BEGIN PARAMETERS FOR MODEL STIFF_ELASTIC
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL STIFF_ELASTIC]
```

In the above command blocks:

- The density of the material is defined with the DENSITY command line.
- The Biot's coefficient of the material is defined with the BIOTS COEFFICIENT command line.
- The thermal strain option is used to define thermal strains. See Section 5.1.3.1 and Section 5.1.3.2 for further information on defining and activating thermal strains.
- Any two of the following elastic constants are required to define the unscaled material response:
    - Young's modulus is defined with the YOUNGS MODULUS command line.
    - Poisson's ratio is defined with the POISSONS RATIO command line.
    - The bulk modulus is defined with the BULK MODULUS command line.
    - The shear modulus is defined with the SHEAR MODULUS command line.
    - Lambda is defined with the LAMBDA command line.
- The following command lines are used only in Adagio:
    - The material scaling is defined with the SCALE FACTOR command line.
    - The reference strain is defined with the REFERENCE STRAIN command line.

As noted previously, only two of the elastic constants are required to define the unscaled material response.

The scaled bulk and shear moduli are computed using a Young's modulus scaled by the value given by the SCALE FACTOR line command.

The REFERENCE STRAIN command line supplies a value for the reference strain used to create a normalized material constraint violation that is based on strains. Specifying a reference strain implies the use of strains for measuring the material constraint violation (or part of the control-stiffness error in Adagio). Otherwise, the material constraint violation is determined using the change in the scaled stress response over the current model problem.

5.2.28 Swanson Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL SWANSON
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    A1 = <real>a1
    P1 = <real>p1
    B1 = <real>b1
    Q1 = <real>q1
    C1 = <real>c1
    R1 = <real>r1
    CUT OFF STRAIN = <real>ecut
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
    MAX POISSONS RATIO = <real>max_poissons_ratio(0.5)
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL SWANSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Swanson model is a hyperelastic model that is used to model rubber. The Swanson model can be used in both Presto and Adagio. When the model is used in Adagio, it can be used with or without the control-stiffness option in Adagio's multilevel solver for nearly incompressible materials where Poisson's ratio, ν, $\approx$ 0.5. The control-stiffness option is implemented via the CONTROL STIFFNESS command block and is discussed in Chapter 3.5. In the course of solving a series of model problems in Adagio, the material response from this model incorporates scaling the bulk and/or shear behaviors to yield a material response that is more amenable to solution using Adagio's conjugate gradient solver. The final material behavior that is calculated corresponds to the actual moduli that are specified. When this model is used in Presto, the material scalings are ignored.

The command block for a Swanson material starts with the line:

```
BEGIN PARAMETERS FOR MODEL SWANSON
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL SWANSON]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- Any two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The material constant A1 is defined with the `A1` command line.
  - The material constant P1 is defined with the `P1` command line.
  - The material constant B1 is defined with the `B1` command line.
  - The material constant Q1 is defined with the `Q1` command line.
  - The material constant C1 is defined with the `C1` command line.
  - The material constant R1 is defined with the `R1` command line.
  - The small-strain value used for computing the initial shear modulus is defined with the `CUT OFF STRAIN` command line.
- The `THERMAL EXPANSION FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the linear thermal expansion as function of temperature. This command line is optional. If it is not present, there is no thermal expansion. See the usage discussion below.
- The following material-scaling command lines are used only in Adagio:
  - The target Young's modulus is defined with the `TARGET E` command line. This command line is optional. See the usage discussion below.
  - The `TARGET E FUNCTION` command line references the name of a function defined in a `DEFINITION FOR FUNCTION` command line in the SIERRA scope that describes the time variation of the target Young's modulus. This command line is optional. If it is not present, there is no time dependence in the Target E parameter. See the usage discussion below.
  - The maximum Poisson's ratio is defined with the `MAX POISSONS RATIO` command line. This command line is optional and will default to 0.5 if not specified. See the usage discussion below.
  - The reference strain is defined with the `REFERENCE STRAIN` command line. This command line is optional. See the usage discussion below.

As noted previously, only two of the elastic constants are required to define the unscaled bulk behavior. Together, the values for parameters in the A1, P1, B1, Q1, C1, and R1 command lines define the unscaled shear behavior, so these command lines must be present. The initial unscaled shear modulus is determined from those parameter values along with the value of the parameter in the CUT OFF STRAIN command line, so this command line must also be present.

The Swanson model, like a few of the material models, allows for the specification of thermal strain behavior within the material model itself, via the THERMAL EXPANSION FUNCTION command line. This command line, like the other "function-type" command lines in this model, requires that a function associated with the name be defined in the SIERRA scope.

The bulk and shear scalings that can be used with the multilevel solver in Adagio are specified via a combination of the TARGET E, TARGET E FUNCTION, and MAX POISSONS RATIO command lines. If the TARGET E command line is not included (and the MAX POISSONS RATIO command line is included), the shear scaling is set to 1.0, and the bulk scaling is determined from the ratio of the scaled bulk modulus to its unscaled value, where the scaled bulk modulus is computed using the value of the max_poissons_ratio parameter along with the unscaled shear modulus. On the other hand, if both the TARGET E command line and the MAX POISSONS RATIO are included, bulk and shear scaling values are computed using scaled moduli that are calculated from the target_e and max_poissons_ratio parameter values.

Including the TARGET E FUNCTION command line allows time-dependent bulk and shear scaling to be used. If this command line is not specified, the bulk and shear scalings remain constant in solution time. If the command line is specified, the target Young's modulus that is used for computing the scaled moduli is multiplied by the function value.

The REFERENCE STRAIN command line supplies a value for the reference strain used to create a normalized material constraint violation that is based on strains. Specifying a reference strain implies the use of strains for measuring the material constraint violation (or part of the control-stiffness error in Adagio). Otherwise, the material constraint violation is determined using the change in the scaled stress response over the current model problem.

Output variables available for this model are listed in Table 9.40. Brief documentation on the theoretical basis for the Swanson model is given in Reference 17.

5.2.29 Viscoelastic Swanson Model

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL VISCOELASTIC_SWANSON
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    A1 = <real>a1
    P1 = <real>p1
    B1 = <real>b1
    Q1 = <real>q1
    C1 = <real>c1
    R1 = <real>r1
    CUT OFF STRAIN = <real>ecut
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    PRONY SHEAR INFINITY = <real>ginf
    PRONY SHEAR 1 = <real>g1
    PRONY SHEAR 2 = <real>g2
    PRONY SHEAR 3 = <real>g3
    PRONY SHEAR 4 = <real>g4
    PRONY SHEAR 5 = <real>g5
    PRONY SHEAR 6 = <real>g6
    PRONY SHEAR 7 = <real>g7
    PRONY SHEAR 8 = <real>g8
    PRONY SHEAR 9 = <real>g9
    PRONY SHEAR 10 = <real>g10
    SHEAR RELAX TIME 1 = <real>tau1
    SHEAR RELAX TIME 2 = <real>tau2
    SHEAR RELAX TIME 3 = <real>tau3
    SHEAR RELAX TIME 4 = <real>tau4
    SHEAR RELAX TIME 5 = <real>tau5
    SHEAR RELAX TIME 6 = <real>tau6
    SHEAR RELAX TIME 7 = <real>tau7
    SHEAR RELAX TIME 8 = <real>tau8
    SHEAR RELAX TIME 9 = <real>tau9
    SHEAR RELAX TIME 10 = <real>tau10
    WLF COEF C1 = <real>wlf_c1
    WLF COEF C2 = <real>wlf_c2
    WLF TREF = <real>wlf_tref
    NUMERICAL SHIFT FUNCTION = <string>ns_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
```

```
    MAX POISSONS RATIO = <real>max_poissons_ratio(0.5)
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL VISCOELASTIC_SWANSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The viscoelastic Swanson model is a finite strain viscoelastic model that has an initial elastic response that matches the Swanson material model. The bulk response is elastic, while the shear response is viscoelastic. This model is commonly employed in simulating the response of rubber materials. The viscoelastic Swanson model can be used in both Presto and Adagio. When the model is used in Adagio, it can be used with or without the control-stiffness option in Adagio's multilevel solver for nearly incompressible materials where Poisson's ratio, ν, ≈ 0.5. The control-stiffness option is implemented via the `CONTROL STIFFNESS` command block and is discussed in Chapter 3.5. In the course of solving a series of model problems in Adagio, the material response from this model incorporates scaling the bulk and/or shear behaviors to yield a material response that is more amenable to solution using Adagio's conjugate gradient solver. The final material behavior that is calculated corresponds to the actual moduli that are specified. When this model is used in Presto, the material scalings are ignored.

The command block for a viscoelastic Swanson material starts with the line:

```
BEGIN PARAMETERS FOR MODEL VISCOELASTIC_SWANSON
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL VISCOELASTIC_SWANSON]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line.
- The Biot's coefficient of the material is defined with the `BIOTS COEFFICIENT` command line.
- Any two of the following elastic constants are required to define the unscaled bulk behavior:
  - Young's modulus is defined with the `YOUNGS MODULUS` command line.
  - Poisson's ratio is defined with the `POISSONS RATIO` command line.
  - The bulk modulus is defined with the `BULK MODULUS` command line.
  - The shear modulus is defined with the `SHEAR MODULUS` command line.
  - Lambda is defined with the `LAMBDA` command line.
- The following command lines are required:
  - The material constant A1 is defined with the `A1` command line.

- The material constant P1 is defined with the P1 command line.
- The material constant B1 is defined with the B1 command line.
- The material constant Q1 is defined with the Q1 command line.
- The material constant C1 is defined with the C1 command line.
- The material constant R1 is defined with the R1 command line.
- The small-strain value used for computing the glassy shear modulus is defined with the CUT OFF STRAIN command line.

- The THERMAL EXPANSION FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the linear thermal expansion as function of temperature. This command line is optional. If it is not present, there is no thermal expansion. See the usage discussion below.

- PRONY SHEAR INFINITY command line. This command line is required.

- The normalized relaxation spectra coefficients are specified with the PRONY SHEAR I command lines, where the value of I varies sequentially from 1 to 10. These command lines are optional.

- The normalized relaxation spectra time constants are specified with the SHEAR RELAX TIME I command lines, where the value of I varies sequentially from 1 to 10. These command lines are optional.

- WLF COEF C1 command line. This command line is required.

- WLF COEF C2 command line. This command line is required.

- WLF TREF command line. This command line is required.

- NUMERICAL SHIFT FUNCTION command line. This command line is optional.

- The following material-scaling command lines are used only in Adagio:

  - The target Young's modulus is defined with the TARGET E command line. This command line is required. See the usage discussion below.
  - The TARGET E FUNCTION command line references the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope that describes the time variation of the target Young's modulus. This command line is optional. If it is not present, there is no time dependence in the Target E parameter. See the usage discussion below.
  - The maximum Poisson's ratio is defined with the MAX POISSONS RATIO command line. This command line is optional and will default to 0.5 if not specified. See the usage discussion below.
  - The reference strain is defined with the REFERENCE STRAIN command line. This command line is required. See the usage discussion below.

As noted previously, only two of the elastic constants are required to define the unscaled bulk behavior. Together, the values for parameters in the A1, P1, B1, Q1, C1, and R1 command lines define the unscaled glassy shear behavior, so these command lines must be present. The unscaled glassy shear modulus is determined from those parameter values along with the value of the parameter in the CUT OFF STRAIN command line, so this command line must also be present.

The viscoelastic Swanson model, like a few of the material models, allows for the specification of thermal strain behavior within the material model itself, via the THERMAL EXPANSION FUNCTION command line. This command line, like the other "function-type" command lines in this model requires that a function associated with the name be defined in the SIERRA scope.

The bulk and shear scalings that can be used with the multilevel solver in Adagio are specified via a combination of the TARGET E, TARGET E FUNCTION, and MAX POISSONS RATIO command lines. If the TARGET E command line is not included (and the MAX POISSONS RATIO command line is included), the shear scaling is set to 1.0, and the bulk scaling is determined from the ratio of the scaled bulk modulus to its unscaled value, where the scaled bulk modulus is computed using the value of the max_poissons_ratio parameter along with the unscaled shear modulus. On the other hand, if both the TARGET E command line and the MAX POISSONS RATIO command line are included, bulk and shear scaling values are computed using scaled moduli that are calculated from the target_e and max_poissons_ratio parameter values.

Including the TARGET E FUNCTION command line allows time-dependent bulk and shear scaling to be used. If this command line is not specified, the bulk and shear scalings remain constant in solution time. If the command line is specified, the target Young's modulus that is used for computing the scaled moduli is multiplied by the function value.

The REFERENCE STRAIN command line supplies a value for the reference strain used to create a normalized material constraint violation that is based on strains. Specifying a reference strain implies the use of strains for measuring the material constraint violation (or part of the control-stiffness error in Adagio). Otherwise, the material constraint violation is determined using the change in the scaled stress response over the current model problem.

Output variables available for this model are listed in Table 9.41. Brief documentation on the theoretical basis for the viscoelastic Swanson model is given in References 17, 18, 19, and 20.

5.3 Cohesive Zone Material Models

Several material models are available for use with cohesive zone elements or with Dash contact, and are described in this section.

Traction separation models used for cohesive surface elements are input within BEGIN PROPERTY SPECIFICATION FOR MATERIAL blocks in the same manner as continuum models. Although density is not a property used by cohesive zone elements, because of their specification within this block, all of these models currently require a density to be provided as part of their input.

5.3.1 Traction Decay

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL TRACTION_DECAY
    NORMAL DECAY LENGTH = <real>
    TANGENTIAL DECAY LENGTH = <real>
  END [PARAMETERS FOR MODEL TRACTION_DECAY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Traction Decay cohesive model is a simple model that get initialized with a traction upon activation or insertion of the cohesive element, and decays that traction to zero over specified values of normal and tangential separation. This model is only valid to use in conjunction with dynamic cohesive zone activation through MPC deactivation or dynamic insertion of cohesive surface elements.

The command block for a traction decay material starts with the line:

```
BEGIN PARAMETERS FOR MODEL TRACTION_DECAY
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL TRACTION_DECAY]
```

In the above command block:

- The density of the material is defined with the DENSITY command line. Although this is not used in the model, it is currently a required input parameter.
- The separation length over which the normal traction decays to zero is set by the NORMAL DECAY LENGTH command line.
- The separation length over which the tangential traction decays to zero is set by the TANGENTIAL DECAY LENGTH command line.

5.3.2 Tvergaard Hutchinson

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON
    INIT TRACTION METHOD = IGNORE|ADD (IGNORE)
    LAMBDA_1 = <real>
    LAMBDA_2 = <real>
    NORMAL LENGTH SCALE = <real>
    TANGENTIAL LENGTH SCALE = <real>
    PEAK TRACTION = <real>
    PENETRATION STIFFNESS MULTIPLIER = <real>
    NORMAL INITIAL TRACTION DECAY LENGTH = <real>
    TANGENTIAL INITIAL TRACTION DECAY LENGTH = <real>
    USE ELASTIC UNLOADING = NO|YES (YES)
  END [PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Tvergaard Hutchinson cohesive model combines the normal and tangential separation into a single normalized separation and calculates a traction per unit length based on this value. This model then calculates normal and tangential traction based on the ratio of the normal and tangential length scales.

For a Tvergaard Hutchinson surface model, a Tvergaard Hutchinson command block starts with the input line: The command block for a traction decay material starts with the line:

```
BEGIN PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON]
```

In the above command blocks:

- The density of the material is defined with the `DENSITY` command line. Although this is not used in the model, it is currently a required input parameter.
- For dynamically activated or dynamically inserted cohesive surface elements, an initial traction can be added to the calculated traction based on element properties specified in the `ELEMENT DEATH` block and is set via the `INIT TRACTION METHOD`. The default behavior is to ignore any initial tractions and let the traction-separation law dictate the behavior.
- `LAMBDA_1` indicates the normalized separation at which the traction response flattens with an additional increase in separation.
- `LAMBDA_2` indicates the normalized separation at which the traction begins to degrade with an additional increase in separation.

- The separation at which failure occurs in the normal direction is prescribed using the `NORMAL LENGTH SCALE` command.
- The maximum traction is specified through the `PEAK TRACTION` command.
- `NORMAL INITIAL TRACTION DECAY LENGTH` and `TANGENTIAL INITIAL TRACTION DECAY LENGTH` specify the length over which the initial traction will decay to zero in the normal and tangential direction respectively if the cohesive elements are initialized during element death. This decay length is independent of the `NORMAL LENGTH SCALE` and `TANGENTIAL LENGTH SCALE` specified for the calculated traction.
- The separation at which failure occurs in the tangential direction is prescribed using the `TANGENTIAL LENGTH SCALE` command.
- To help prevent interpenetration of the cohesive faces, use the `PENETRATION STIFFNESS MULTIPLIER` command to artificially increase the normal traction when penetration occurs. WARNING: cohesive elements are not well equipped to handle compression. This is an ad-hoc method to handle contact.
- Set the `USE ELASTIC UNLOADING` command to `YES` to force this model to unload elastically.

5.3.3 Thouless Parmigiani

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value

  BEGIN PARAMETERS FOR MODEL THOULESS_PARMIGIANI
    INIT TRACTION METHOD = IGNORE|ADD (IGNORE)
    LAMBDA_1_N = <real>
    LAMBDA_1_T = <real>
    LAMBDA_2_N = <real>
    LAMBDA_2_T = <real>
    NORMAL LENGTH SCALE = <real>
    PEAK NORMAL TRACTION = <real>
    TANGENTIAL LENGTH SCALE = <real>
    PEAK Tangential TRACTION = <real>
    PENETRATION STIFFNESS MULTIPLIER = <real>
    USE ELASTIC UNLOADING = NO|YES (YES)
  END [PARAMETERS FOR MODEL THOULESS_PARMIGIANI]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The Thouless Parmigiani model support mixed-mode fracture more accurately than the Tvergaard Hutchinson model by separating the normal and tangential components, allowing one to fail independently of the other. Failure of this model is dependent on the energy release of both normal and tangential components. The shape of the traction-separation curve for this model is hardening, followed by a plateau, followed by softening.

The command block for a Thouless Parmigiani material starts with the line:

```
BEGIN PARAMETERS FOR MODEL THOULESS_PARMIGIANI
```

and terminates with the line:

```
END [PARAMETERS FOR MODEL THOULESS_PARMIGIANI]
```

In the above command block:

- The density of the material is defined with the `DENSITY` command line. Although this is not used in the model, it is currently a required input parameter.
- For dynamically activated or dynamically inserted cohesive surface elements, an initial traction can be added to the calculated traction based on element properties specified in the `ELEMENT DEATH` block and is set via the `INIT TRACTION METHOD`. The default behavior is to ignore any initial tractions and let the traction-separation law dictate the behavior.
- `LAMBDA_1_N` indicates the normalized normal separation at which the traction response flattens with an additional increase in separation.
- `LAMBDA_1_T` indicates the normalized tangential separation at which the traction response flattens with an additional increase in separation.

- LAMBDA_2_N indicates the normalized normal separation at which the traction begins to decrease with an additional increase in separation.

- LAMBDA_2_T indicates the normalized tangential separation at which the traction begins to decrease with an additional increase in separation.

- The separation at which failure occurs in the normal direction is prescribed using the NORMAL LENGTH SCALE command.

- The maximum normal traction is specified through the PEAK NORMAL TRACTION command.

- The separation at which failure occurs in the tangential direction is prescribed using the TANGENTIAL LENGTH SCALE command.

- The maximum tangential traction is specified through the PEAK TANGENTIAL TRACTION command.

- To help prevent interpenetration of the cohesive faces, use the PENETRATION STIFFNESS MULTIPLIER command to artificially increase the normal traction when penetration occurs. WARNING: cohesive elements are not well equipped to handle compression. This is an ad-hoc method to handle contact.

- Set the USE ELASTIC UNLOADING command to YES to force this model to unload elastically.

5.3.4 Compliant Joint

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value

  BEGIN COMPLIANT JOINT MODEL <string>name
    NORMAL MODULUS = <real>norm_mod
    SHEAR MODULUS = <real>shear_mod
    APERTURE = <real>aperture
    APERTURE LIMIT = <real>aperture_limit
    FRICTION COEFFICIENT = <real>fric_coef
    COHESION = <real>cohesion
  END [PARAMETERS FOR MODEL COMPLIANT_JOINT]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

The compliant joint model is formulated for use in modeling of geologic faults and joints. The joint has an elastic response when both opening and closing defined by `norm_mod` and `shear_mod`. Normal and shear modulus have units of stress.

When the joint is closing, the modulus of the joint varies from `norm_mod` to infinity over a gap distance of `aperture`. The `aperture_limit` is a number between 0.0 and 1.0 that defines how far along the aperture curve the model will go while still increasing the stiffness. A small value causes the model to have a nearly constant stiffness during joint closing. A larger number causes more stiffening of the joint as the overclosure approaches the aperture. A value of 1.0 allows the normal stiffness to go all the way to infinity. Values of aperture limit near one are not recommended as they tend to cause the joint model to become extremely stiff and cause the model to be unstable. The aperture has units of length, while the aperture limit is a dimensionless constant.

When the joint is open, it has a shear stiffness specified by `shear_mod`. When the joint is closing, Coulomb frictional effects are added to the shear force. These are computed as `fric_coef` times the compressive force, where `fric_coef` is a dimensionless friction coefficient.

Additional cohesive resistance to joint opening is provided by `cohesion`. This parameter has units of stress.

5.4 References

1. Stone, C. M. *SANTOS – A Two-Dimensional Finite Element Program for the Quasistatic, Large Deformation, Inelastic Response of Solids*, SAND90-0543. Albuquerque, NM: Sandia National Laboratories, 1996. pdf.

2. Bammann, D. J., M. L. Chiesa, and G. C. Johnson. "Modelling Large Deformation and Failure in Manufacturing Processes." In *Proceedings of the 19th International Congress of Theoretical and Applied Mechanics*, edited by T. Tatsumi, E. Watanabe, and T. Kambe, 359–376. Amsterdam: Elsevier Science Publishers, 1997.

3. Bammann, D. J., M. L. Chiesa, M. F. Horstemeyer, and L. E. Weingarten. "Failure in Ductile Materials Using Finite Element Methods." In *Structural Crashworthiness and Failure*, edited by N. Jones and T. Wierzbicki, 1–53. London: Elsevier Applied Science, 1993.

4. Bammann, D. J. "Modeling Temperature and Strain Dependent Large Deformations in Metals." *Applied Mechanics Reviews* 43, no. 5 (1990): S312–319. doi.

5. Simo, J. C., and T. J. R. Hughes. *Computational Inelasticity*, Springer-Verlag, New York, NY, 1998.

6. Taylor, L. M., and D. P. Flanagan. *PRONTO3D: A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912. Albuquerque, NM: Sandia National Laboratories, March 1989. pdf.

7. Krieg, R. D. *A Simple Constitutive Description for Cellular Concrete*, SAND SC-DR-72-0883. Albuquerque, NM: Sandia National Laboratories, 1978. pdf.

8. Swenson, D. V., and L. M. Taylor. "A Finite Element Model for the Analysis of Tailored Pulse Stimulation of Boreholes." *International Journal for Numerical and Analytical Methods in Geomechanics* 7 (1983): 469–484. doi.

9. Attaway, S. W., R. V. Matallucci, S. W. Key, K. B. Morrill, L. J. Malvar, and J. E. Crawford. *Enhancements to PRONTO3D to Predict Structural Response to Blast*, SAND2000-1017. Albuquerque, NM: Sandia National Laboratories, 2000.

10. *ACI318-08: Building Code Requirements for Structural Concrete and Commentary*. Farmington Hills, MI: American Concrete Institute, 2008.

11. Neilsen, M. K., and Morgan, H. S., and Krieg, R. D. *A Phenomenological Constitutive Model for Low Density Polyurethane Foams*, SAND86-2927. Albuquerque, NM: Sandia National Laboratories, April 1987. pdf.

12. Neilsen, M. K., and Pierce, J. D., and Krieg, R. D. *A Constitutive Model for Layered Wire Mesh and Aramid Cloth Fabric*, SAND91-2850. Albuquerque, NM: Sandia National Laboratories, 1993. pdf.

13. Green, A. E., and W. Zerna. *Theoretical Elasticity, 2nd Edition*. Oxford: Clarendon Press, 1968.

14. Whirley, R. G., B. E. Engelmann, and J. O. Halquist. *DYNA3D Users Manual*. Livermore, CA: Lawrence Livermore Laboratory, 1991.

15. Hammerand, D. C. *Laminated Composites Modeling in ADAGIO/PRESTO*, SAND2004-2143. Albuquerque, NM: Sandia National Laboratories, 2004. pdf.

16. Hammerand, D. C. *Critical Time Step for a Bilinear Laminated Composite Mindlin Shell Element*, SAND2004-2487. Albuquerque, NM: Sandia National Laboratories, 2004. pdf.

17. Scherzinger, W. M., and D. C. Hammerand. *Constitutive Models in LAME*, SAND2007-5873. Albuquerque, NM: Sandia National Laboratories, September 2007. pdf.

18. HKS. *ABAQUS Version 6.6, Theory Manual*. Providence, RI: Hibbitt, Karlsson and Sorensen, 2006.

19. Hammerand, D. C. “ABAQUS Style Finite Strain Viscoelasticity in Adagio.” Memo. Albuquerque, NM: Sandia National Laboratories, March 2003.

20. Hammerand, D. C. “Finite Strain Viscoelasticity in Adagio and ABAQUS.” Memo. Albuquerque, NM: Sandia National Laboratories, July 2003.

21. Swegle, J. W. *SIERRA: PRESTO Theory Documentation: Energy Dependent Materials Version 1.0*. Albuquerque, NM: Sandia National Laboratories, October 2001.

22. Johnson, G. R., and Cook, W. H. “A constitutive model and data for metals subjected to large strains, high strain rates and high temperatures” *Proc. 7th. Int. Symp. on Ballistics, The Hague, The Netherlands* (1983): 541–547.

Chapter 6

Elements

This chapter explains how material, geometric, and other properties are associated with the various element blocks in a mesh file. A mesh file contains, for the most part, only topological information about elements. For example, there may be a group of elements in the mesh file that consists of four nodes defining a planar facet in three-dimensional space. Whether or not these elements are used as shells or membranes in our actual model of an object is determined by command lines in the input file. The specifics of a material type associated with these four node facets are also set in the input file.

Most elements can be used in either explicit or implicit calculations. If an element is available for one but not the other, this information will be noted for the element. In particular, there are special element implementations available with explicit calculations for peridynamics and for smoothed particle hydrodynamics (SPH). This chapter also includes descriptions of the commands for mass property calculations, element death, and mesh rebalancing (mesh rebalancing is available for explicit calculations only). Two "element-like" capabilities are discussed in Chapter 6—torsional springs (explicit only) and rigid bodies.

Highlights of chapter contents follow. Section 6.1 discusses the `FINITE ELEMENT MODEL` command block, which provides the description of a mesh that will be associated with the elements. Section 6.2 presents the section command blocks that are used to define the different element sections. Next in Section 6.3 are descriptions of command blocks that exhibit element-like functionality. Section 6.3.1 explains the use of rigid bodies. Section 6.3.2 describes how to implement a torsional spring mechanism (explicit only). Section 6.4 describes the `MASS PROPERTIES` command block, which lets the user compute the total mass of the model or the mass of sub-parts of the model once the element blocks are completely defined in terms of geometry and material. Section 6.5 details the `ELEMENT DEATH` command block, which lets the user delete (kill) elements based on various criteria during an analysis. Section 6.6 describes particle embedding (explicit only), which specifies element blocks in which each element is embedded with particles during initialization. A command block for derived quantities that are to be used with transfers or error estimation is discussed in Section 6.7. Section 6.8 presents various options available in explicit calculations for rebalancing of the initial mesh decomposition for parallel runs, which may improve the parallel performance of some problems. Finally, Section 6.9 describes the procedure for adaptively remeshing a portion of the model.

Most of the command blocks and command lines described next appear within the SIERRA scope. There are some exceptions, and these exceptions are noted.

6.1 Finite Element Model

```
BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
  DATABASE NAME = <string>mesh_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  ALIAS <string>mesh_identifier AS <string>user_name
  OMIT BLOCK <string>block_list
  COMPONENT SEPARATOR CHARACTER = <string>separator
  BEGIN PARAMETERS FOR BLOCK [<string list>block_names]
    #
    # Command lines that define attributes for
    # a particular element block appear in this
    # command block.
    #
  END [PARAMETERS FOR BLOCK <string list>block_names]
END [FINITE ELEMENT MODEL <string>mesh_descriptor]
```

The input file must point to a mesh file that is to be used for an analysis. The name of the mesh file appears within a `FINITE ELEMENT MODEL` command block, which appears in the SIERRA scope. In this command block, you will identify the particular mesh file that describes your model. Also within this command block, there will be one or more `PARAMETERS FOR BLOCK` command blocks. (All the `PARAMETERS FOR BLOCK` command blocks are embedded in the `FINITE ELEMENT MODEL` command block.) Within the `PARAMETERS FOR BLOCK` command block, you will set a material type and model, a section, and various other parameters for the element block. The concept of "section" is explained in Section 6.1.5.

The current element library is as follows:

- Eight-node, uniform-gradient hexahedron: Both a midpoint-increment formulation [1] and a strongly objective formulation are implemented [2]. These elements can be used with any of the material models described in Chapter 5.
- Eight-node, selective-deviatoric hexahedron: Only a strongly objective formulation is provided. This element can be used with any of the material models described in Chapter 5.
- Four-node tetrahedron: There is now the regular element formulation for the four-node tetrahedron and a node-based formulation for the four-node tetrahedron. For the regular element formulation, only a strongly objective formulation is implemented. The concept of a node-based four-node tetrahedron is described in Reference 3. The regular four-node tetrahedron can be used with any of the material models described in Chapter 5. The node-based tetrahedron can be used with any of the material models described in Chapter 5. When

using node-based tetrahedron it is important that nodal quantities are used where other element types use element quantities. For example, you evaluate element variable stress with a regular four-node tetrahedron but in node-based tetrahedron one should use nodal variable element_stress_## where ## is a number that increments from 1. Nodal variables for node-based tets will live on all nodes but are zero where they are not used. Element variables for node-based elements serve as intermediate variables and should not be used in post processing in the same way as other regular element variables.

- Eight-node tetrahedron: This tetrahedral element has nodes at the four vertices and nodes on the four faces. The eight-node tetrahedron has only a strongly objective formulation [4]. The eight-node tetrahedron uses a mean quadrature formulation even though it has the additional nodes. This element can be used with any of the material models described in Chapter 5.

- Ten-node tetrahedron: Only a strongly objective formulation is implemented. This element can be used with any of the material models described in Chapter 5.

- Four-node, quadrilateral, uniform-gradient membrane: Both a midpoint-increment formulation and a strongly objective formulation are implemented. This element is derived from the Key-Hoff shell formulation [5]. The strongly objective formulation has not been extensively tested, and it is recommended that the midpoint-increment formulation, which is the default, be used for this element type. These elements can be used with any of the following material models described in Chapter 5:

  - Elastic
  - Elastic-plastic
  - Elastic-plastic power-law hardening
  - Multilinear elastic-plastic hardening (no failure)

- Three-node, triangular shell: This shell uses the same formulation as the three-node triangular shell in Pronto [1]. A midpoint-increment formulation is implemented. This element can be used with any of the following material models described in Chapter 5:

  - Elastic
  - Elastic-plastic
  - Elastic-plastic power-law hardening
  - Multilinear elastic-plastic hardening without failure
  - Multilinear elastic-plastic hardening with failure

- Four-node, quadrilateral shell: This shell uses the Key-Hoff formulation [5]. Both a midpoint-increment formulation and a strongly objective formulation are implemented. The strongly objective formulation has not been extensively tested, and it is recommended that the midpoint-increment formulation, which is the default, be used for this element type. These elements can be used with any of the following material models described in Chapter 5:

  - Elastic

- Elastic-plastic
  - Elastic-plastic power-law hardening
  - Multilinear elastic-plastic hardening without failure
  - Multilinear elastic-plastic hardening with failure

- Four-node, quadrilateral, selective-deviatoric membrane: Only a midpoint-increment formulation is implemented. These elements can be used with any of the following material models described in Chapter 5:
  - Elastic
  - Elastic-plastic
  - Elastic-plastic power-law hardening
  - Multilinear elastic-plastic hardening (no failure)

- Linear elastic shell element: The linear elastic shell element is linear in both a material and geometric sense. The linear elastic shell element can be used with any material, however it will use only the density and elastic material constants of that material. Use of the linear elastic shell element is specified with the `FORMULATION = NQUAD` command in the section command block.

- Two-node beam: The beam element is a uniform result model. Strains and stresses are computed only at the midpoint of the element. These midpoint values determine the forces and moments for the beam. The beam element is based on an incremental kinematic formulation that is accurate for large strains and rotations (e.g. this element exactly agrees with a logarithmic strain formulation under large strain axial loading). Thinning of the cross section is taken into account through a constant volume assumption. There are five different sections currently implemented for the beam: rod, tube, bar, box, and I. This element can be used with any of the following material models described in Chapter 5:
  - Elastic
  - Elastic-plastic

- Two-node truss: The two-node truss element carries only a uniform axial stress. Currently, there is a linear-elastic material model for the truss element.

- Two-node spring: The two-node spring element computes a uniaxial resistance force based on a non-linear force-engineering strain function. This element can handle preloads, mass per unit length, resetting of the initial length after preload and any arbitrary loading function.

- Two-node damper: (Code Usage: Presto only) The two-node damping element computes a damping force based on the relative velocity of the two nodes along the axis of the element. This element uses only a damping parameter for a material property.

- Point mass: The point mass element allows the user to put a specified mass and/or rotational inertia at a node. This element requires input for density, but does not make use of any other material properties.

- Particle elements: These are one-dimensional elements. These elements can be used with any of the material models described in Chapter 5.

The command block to describe a mesh file begins with

```
BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
```

and is terminated with:

```
END [FINITE ELEMENT MODEL <string>mesh_descriptor]
```

where `mesh_descriptor` is a user-selected name for the mesh. In this section, we will first discuss the command lines within the scope of the `FINITE ELEMENT MODEL` command block but outside the scope of the `PARAMETERS FOR BLOCK` command block. We will then discuss the `PARAMETERS FOR BLOCK` command block and the associated command lines for this particular block.

6.1.1 Identification of Mesh File

Nested within the `FINITE ELEMENT MODEL` command block are two command lines (`DATABASE NAME` and `DATABASE TYPE`) that give the mesh name and define the type for the mesh file, respectively. The command line

```
DATABASE NAME = <string>mesh_file_name
```

gives the name of the mesh file with the string `mesh_file_name`. If the current mesh file is in the default directory and is named `job.g`, then this command line would appear as:

```
DATABASE NAME = job.g
```

If the mesh file is in some other directory, the command line would have to show the path to that directory. For parallel runs, the string `mesh_file_name` is the base name for the spread of parallel mesh files. For example, for a four-processor run, the actual mesh files associated with a base name of `job.g` would be `job.g.4.0`, `job.g.4.1`, `job.g.4.2`, and `job.g.4.3`. The database name on the command line would be `job.g`.

Two metacharacters can appear in the name of the mesh file. If the `%P` character is found in the name, it will be replaced with the number processors being used for the run. For example, if you are running on 1024 processors and use the name `mesh-%P/job.g`, then the name would be expanded to `mesh-1024/job.g` and the actual mesh files would be `mesh-1024/job.g.1024.0000` to `mesh-1024/job.g.1024.1023`. The other recognized metacharacter is `%B` which is replaced with the base name of the input file containing the input commands. For example, if the commands are in the file `my_analysis_run.i` and the mesh database name is specified as `%B.g`, then the mesh would be read from the file `my_analysis_run.g`.

If the mesh file does not use the Exodus II format, you must specify the format for the mesh file using the command line:

```
DATABASE TYPE = <string>database_type(exodusII)
```

Currently, only the Exodus II database format is supported by Presto and Adagio for mesh input. Other options may be added in the future.

6.1.2 Alias

It is possible to associate a user-defined name with some mesh entity. The mesh entity names for Exodus II entities are typically the concatenation of the entity type (for example, “block”, “nodelist”, or “surface”), an underscore (“_”), and the entity id. This generated name can be aliased to a more descriptive name by using the `ALIAS command line`:

```
ALIAS <string>mesh_identifier AS <string>user_name
```

This alias can then be used in other locations in the input file in place of the Exodus II name.

Examples of this association are as follows:

```
Alias block_1   as Case
Alias block_10  as Fin
Alias block_12  as Nose
Alias surface_1 as Nose_Case_Interface
Alias surface_2 as OuterBoundary
```

The above examples use the Exodus II naming convention described in Section 1.5.

6.1.3 Omit Block

If the finite element mesh contains element blocks that should be omitted from the finite element analysis, the `OMIT BLOCK` line command is used.

```
OMIT BLOCK <string>block_list
```

The element blocks listed in the command are removed from the model. Any node sets or surfaces only existing on nodes or elements in the omitted element blocks are also omitted. When using this command, it is necessary to deactivate (comment out) or remove any references to the omitted blocks in the rest of the input file. Note that if this command is used in a parallel analysis, it is possible for the resulting model to become unbalanced if, for example, the omitted element blocks make up a large portion of the elements on one or more processors. In this case with explicit calculations, the mesh can be rebalanced using the `REBALANCE` command described in Section 6.8.1.

Examples of omitting element blocks are:

```
Omit Block block_1 block_2
Omit Block block_10
```

6.1.4 Component Separator Character

A variable defined on the mesh database can be used as an initial condition, or a prescribed temperature with the `READ VARIABLE` command. If the variable is a vector or a tensor, then the base name of the variable will be separated from the suffixes with a separator character. The default separator character is an underscore, but it can be changed with the `COMPONENT SEPARATOR CHARACTER` command.

```
COMPONENT SEPARATOR CHARACTER = <string>character|NONE
```

For example, the variable `displacement` can have the suffixes `x`, `y`, etc. By default, the base name is separated from the suffixes with an underscore character so that we have `displacement_x`, `displacement_y`, etc. in the mesh file. The underscore can be replaced as the default separator by using the above command line. If the data used the period as the separator, then the command would be

```
COMPONENT SEPARATOR CHARACTER = .
```

For the displacement example the components would then appear in the mesh file as `displacement.x`, `displacement.y`, etc.

The separator can be eliminated with an empty string or `NONE`.

6.1.5 Descriptors of Element Blocks

```
BEGIN PARAMETERS FOR BLOCK [<string list>block_names]
  MATERIAL <string>material_name
  SOLID MECHANICS USE MODEL <string>model_name
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  SECTION = <string>section_id
  LINEAR BULK VISCOSITY =
    <real>linear_bulk_viscosity_value(0.06)
  QUADRATIC BULK VISCOSITY =
    <real>quad_bulk_viscosity_value(1.20)
  HOURGLASS STIFFNESS =
    <real>hour_glass_stiff_value(solid = 0.05,
      shell/membrane = 0.0)
  HOURGLASS VISCOSITY =
    <real>hour_glass_visc_value(solid = 0.0,
      shell/membrane = 0.0)
  MEMBRANE HOURGLASS STIFFNESS =
    <real>memb_hour_glass_stiff_value(0.0)
  MEMBRANE HOURGLASS VISCOSITY =
    <real>memb_hour_glass_visc_value(0.0)
  BENDING HOURGLASS STIFFNESS =
    <real>bend_hour_glass_stiff_value(0.0)
  BENDING HOURGLASS VISCOSITY =
    <real>bend_hour_glass_visc_value(0.0)
  TRANSVERSE SHEAR HOURGLASS STIFFNESS =
    <real>tshr_hour_glass_stiff_value(0.0)
  TRANSVERSE SHEAR HOURGLASS VISCOSITY =
    <real>tshr_hour_glass_visc_value(0.0)
  EFFECTIVE MODULI MODEL = <string>PRESTO|PRONTO|CURRENT|
      ELASTIC(PRONTO)
  ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
```

```
    ACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
      <string list>period_names
    INACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
      <string list>period_names
  END [PARAMETERS FOR BLOCK <string list>block_names]
```

The finite element model consists of one or more element blocks. Associated with an element block or group of element blocks will be a PARAMETERS FOR BLOCK command block, which is also referred to in this document as an *element-block command block*. The basic information about the element blocks (number of elements, topology, connectivity, etc.) is contained in a mesh file. Specific attributes for an element block must be specified in the input file. If for example, a block of eight-node hexahedra is to use the selective-deviatoric versus mean-quadrature formulation, then the selective-deviatoric formulation must be specified in the input file. The element library is listed at the beginning of Section 6.1.

The element-block command block begins with the input line

```
BEGIN PARAMETERS FOR BLOCK [<string list>block_names]
```

and is terminated with the input line:

```
END [PARAMETERS FOR BLOCK <string list>block_names]
```

Here block_names is a list of element blocks assigned to the element-block command block. Such a list must be included on the BEGIN PARAMETERS FOR BLOCK input line if the INCLUDE ALL BLOCKS line command is not used. If the format for the mesh file is Exodus II, the typical form of a block_name is block_integerID, where integerID is the integer identifier for the block. If the element block is 280, the value of block_name would be block_280. It is also possible to generate an alias identifier for the element block and use this for the block_name. If block_280 is aliased to AL6061, then block_name becomes AL6061.

All the element blocks listed on the PARAMETERS FOR BLOCK command line (or all the element blocks included using the line commands INCLUDE ALL BLOCKS and REMOVE BLOCK) will have the same mechanics properties. The mechanics properties are set by use of the various command lines. One of the key command lines, i.e., MATERIAL, will let you associate a material with the elements in the block. Another key command line is the SECTION command line. This command line lets you differentiate between elements with the same topology but different formulations. For example, assume that the topology of the elements in a block is a four-node quadrilateral. With the SECTION command line you can specify whether the element block will be used as a membrane or a shell. The SECTION command line also lets you assign a variety of parameters to an element, depending on the element formulation.

It is important to state here that the SECTION command line only specifies an identifier that maps to a section command block that is defined by the user. There are currently several kinds of section command blocks for the different elements: SOLID SECTION, COHESIVE SECTION, SHELL SECTION, MEMBRANE SECTION, BEAM SECTION, TRUSS SECTION, SPRING SECTION, DAMPER SECTION, POINT MASS SECTION, PARTICLE SECTION, PERIDYNAMICS SECTION, and SUPERELEMENT SECTION. It is within a section command block that the formulation-specific entities related to a particular element are specified. If no SECTION com-

mand line is present in an element-block command block, the code assumes the element block is a block of eight-node hexahedra using mean quadrature and the midpoint-increment formulation.

All the command lines that can be used for the element-block command block are described in Section 6.1.5.1 through Section 6.1.5.9.

6.1.5.1 Material Property

```
MATERIAL <string>material_name
SOLID MECHANICS USE MODEL <string>model_name
```

The material property specification for an element block is done by using the above two command lines. The property specification references both a PROPERTY SPECIFICATION FOR MATERIAL command block and a material-model command block, which has the general form PARAMETERS FOR MODEL model_name. These command blocks are described in Chapter 5. The PROPERTY SPECIFICATION FOR MATERIAL command block contains all the parameters needed to define a material, and is associated with an element block (PARAMETERS FOR BLOCK command block) by use of the MATERIAL command line. Some of the material parameters inside the property specification are grouped on the basis of material models. A material-model command block is associated with an element block by use of the SOLID MECHANICS USE MODEL command line.

Consider the following example. Suppose there is a PROPERTY SPECIFICATION FOR MATERIAL command block with a material_name of steel. Embedded within this command block for steel is a material-model command block for an elastic model of steel and an elastic-plastic model of steel. Suppose that for the current element block we would like to use the material steel with the elastic model. Then the element-block command block would contain the input lines:

```
MATERIAL steel
SOLID MECHANICS USE MODEL elastic
```

If, on the other hand, we would like to use the material steel with the elastic-plastic model, the element-block command block would contain the input lines:

```
MATERIAL steel
SOLID MECHANICS USE MODEL elastic_plastic
```

The user should remember that not all material types can be used with all element types.

6.1.5.2 Include All Blocks

The INCLUDE ALL BLOCKS line command is used to associate all element blocks with the same element parameters (which minimizes input).

```
INCLUDE ALL BLOCKS
```

6.1.5.3 Remove Block

The `REMOVE BLOCK` command line allows you to delete blocks from the set specified in the `PARAMETERS FOR BLOCK` command block or `INCLUDE ALL BLOCKS` command line(s) through the string list `block_names`.

```
REMOVE BLOCK <string>block_list
```

6.1.5.4 Section

```
SECTION = <string>section_id
```

The section specification for an element-block command block is done by using the above command line. The `section_id` is a string associated with a section command block. The various section command blocks are described in Section 6.2.

Suppose you wanted the current element-block command block to use the membrane formulation. You would define a `MEMBRANE SECTION` command block with some name, such as `membrane_rubber`. Inside the current element-block command block you would have the command line:

```
SECTION = membrane_rubber
```

The thickness of the membrane would be described in the `MEMBRANE SECTION` command block and then associated with the current element-block command block.

There can be only one `SECTION` command line in an element-block command block. Each element-block command block within the model description can reference a unique section command block, or several element-block command blocks can reference the same section command block. For example, in Figure 6.1, the section named `membrane_rubber` appears in two different `PARAMETERS FOR MODEL` command blocks, but there is only one specification for their associated `MEMBRANE SECTION` command block. When several element-block command blocks reference the same section, the input file is less verbose, and it is easier to maintain the input file.

6.1.5.5 Linear and Quadratic Bulk Viscosity

```
LINEAR BULK VISCOSITY =
  <real>linear_bulk_viscosity_value(0.06)
QUADRATIC BULK VISCOSITY =
  <real>quad_bulk_viscosity_value(1.20)
```

The linear and quadratic bulk viscosity are set with these two command lines. Consult the documentation for the elements [6] for a description of the bulk viscosity parameters.

6.1.5.6 Hourglass Control

```
HOURGLASS STIFFNESS = <real>hour_glass_stiff_value(solid
```

```
BEGIN FINITE ELEMENT MODEL mesh1
.
.
  BEGIN PARAMETERS FOR BLOCK block1
     .
    SECTION membrane_rubber
    .
  END PARAMETERS FOR BLOCK block1
  BEGIN PARAMETERS FOR BLOCK block2
    .
    SECTION membrane_rubber
    .
  END PARAMETERS FOR BLOCK block2
  .
  .
END FINITE ELEMENT MODEL mesh1

BEGIN MEMBRANE SECTION membrane_rubber
  .
  .
END MEMBRANE SECTION membrane_rubber
```

Figure 6.1: Association between `SECTION` command lines and a section command block.

```
  = 0.05, shell/membrane = 0.0)
HOURGLASS VISCOSITY = <real>hour_glass_visc_value(solid
  = 0.0, shell/membrane = 0.0)
MEMBRANE HOURGLASS STIFFNESS =
  <real>memb_hour_glass_stiff_value(0.0)
MEMBRANE HOURGLASS VISCOSITY =
  <real>memb_glass_visc_value(0.0)
BENDING HOURGLASS STIFFNESS =
  <real>bend_hour_glass_stiff_value(0.0)
BENDING HOURGLASS VISCOSITY =
  <real>bend_glass_visc_value(0.0)
TRANSVERSE SHEAR HOURGLASS STIFFNESS =
  <real>tshr_hour_glass_stiff_value(0.0)
TRANSVERSE SHEAR HOURGLASS VISCOSITY =
  <real>tshr_glass_visc_value(0.0)
```

These command lines set the hourglass control parameters for elements that use hourglass control. Currently, the included elements are the eight-node, uniform-gradient hexahedral elements; the eight-node and ten-node tetrahedral elements; and the four-node membrane and shell elements. Consult the element documentation [6] for a description of the hourglass parameters.

Hourglass stiffness and viscosity parameters for hexahedral and tetrahedral elements are set using the `HOURGLASS STIFFNESS` and `HOURGLASS VISCOSITY` commands, respectively. If either of these commands are used for shell elements, they set the hourglass stiffness or viscosity for all

three modes (membrane, bending, and transverse shear).

Hourglass parameters for the membrane, bending, and transverse shear modes can be set individually for shell elements. The membrane hourglass stiffness and viscosity can be set with the `MEMBRANE HOURGLASS STIFFNESS` and `MEMBRANE HOURGLASS VISCOSITY` commands. These membrane hourglass commands can also be used with membrane elements. The bending hourglass stiffness and viscosity are set with the `BENDING HOURGLASS STIFFNESS` and `BENDING HOURGLASS VISCOSITY` commands, and transverse shear hourglass stiffness and viscosity are set with the `TRANSVERSE SHEAR HOURGLASS STIFFNESS` and `TRANSVERSE SHEAR HOURGLASS VISCOSITY` commands. All of these commands will override either the default values and any value set in the generic `HOURGLASS STIFFNESS` and/or `HOURGLASS VISCOSITY` commands for the particular mode that is specified.

The hourglass stiffness parameter defaults to 0.05 for solids using hourglass control; it defaults to 0.0 for shell and membrane elements. A reasonable user defined hourglass stiffness (if needed) for shells and membranes is 0.005 (approximately an order of magnitude lower than for solid elements). The hourglass viscosity parameter defaults to 0.0 for all elements currently using hourglass control.

The hourglass stiffness is the same as the dilatational hourglass parameter, and the hourglass viscosity is the same as the deviatoric hourglass parameter.

The computation of the hourglass parameters can be strongly affected by the method that computes the effective moduli. The command line in Section 6.1.5.7 selects the method for computing the effective moduli.

6.1.5.7 Effective Moduli Model

```
EFFECTIVE MODULI MODEL =
  <string>PRESTO|PRONTO|CURRENT|ELASTIC(PRONTO)
```

The hourglass force computations require a measure of the material moduli to ensure appropriate scaling of the hourglass forces. For elastic, isotropic material models, the moduli are constant throughout the analysis. However, for nonlinear materials, the moduli are typically computed numerically from the stresses. For models with softening regimes or that approach perfect plasticity, the moduli may be difficult to define, and the way in which they are computed may adversely affect the analysis. Through the `EFFECTIVE MODULI MODEL` command line, Presto provides several methods for the computation of these effective moduli:

- `PRESTO`: This method includes a number of techniques for returning reasonable moduli for softening and perfectly plastic materials. The effective moduli that this approach produces are stiffer than those computed by the `PRONTO` approach.
- `PRONTO`: This method is the default and is identical to the method of computing effective moduli present in the Pronto3D code. It is similar to the `PRESTO` approach but generally produces moduli that are softer than the `PRESTO` approach.

- `CURRENT`: This method computes the effective moduli without any extra handling of negative or near-zero moduli cases. It generally provides the softest response but is also less stable.
- `ELASTIC`: This method simply uses the initial elastic moduli for the entire analysis. It is the most robust but also the most stiff, and may produce an overly stiff global response.

The `EFFECTIVE MODULI MODEL` command line should be used with caution because it can strongly affect the analysis results.

6.1.5.8 Element Numerical Formulation

```
ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
```

For calculation of the critical time step, it is necessary to determine a characteristic length for each element. In one dimension, the correct characteristic element length is obviously the distance between the two nodes of the element. In higher dimensions, this length is usually taken to be the minimum distance between any of the nodes in the element. However, some finite element codes, primarily those based on Pronto3D [1], use as a characteristic length an eigenvalue estimate based on work by Flanagan and Belytschko [7]. That characteristic length provides a stable time step, but in many cases is far more conservative than the minimum distance between nodes. For a cubic element with side length equal to 1, and thus also surface area of each face and volume equal to 1, the minimum distance between nodes is 1. However, the eigenvalue estimate is $1 \big/ \sqrt{3}$, which is only 58% of the minimum distance. As the length of the element is increased in one direction while keeping surfaces in the lateral direction squares of area 1, the eigenvalue estimate asymptotes to $1 \big/ \sqrt{2}$ for very long elements. If the length is decreased, the eigenvalue estimate asymptotes to the minimum distance between nodes for very thin elements. In this case, the eigenvalue estimate is always more conservative than the minimum distance between nodes. However, consider an element whose cross section in one direction is not a square but a trapezoid with one side length much greater than the other. Assume the large side length is 1 and the other side length is arbitrarily small, ε. In this case, the minimum distance between nodes becomes ε, creating a very small and inefficient time step. However, the eigenvalue estimate is related to the length across the middle of the trapezoid, which for the conditions stated is $1/2$. Since both distances provide stable time steps, and one or the other can be much larger in various circumstances, the most efficient calculation is obtained by using the maximum of the two lengths, either the eigenvalue estimate or the minimum distance between nodes, to determine the time step.

By using the maximum of the lengths, the computed critical time step should be at the edge of instability, and the `TIME STEP SCALE FACTOR` command line should be used to provide a margin of safety. In this case the scale factor for the time step should not be greater than 0.9, and in some cases it may have to be reduced further. Thus, although the maximum of the lengths provides a time step that is closer to the critical value and provides better accuracy and efficiency, you may need to specify a smaller-than-expected scale factor for stability. For this reason, the choice of which approach to use is left to the user and is determined by the command line:

```
ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
```

If the input parameter is OLD, only the eigenvalue estimate is used; NEW means that the maximum of the two lengths is used. The default is OLD so that users will have to specifically choose the new approach and be aware of the scale factor for the time step.

The ELEMENT NUMERICAL FORMULATION command line is applicable to both the energy-dependent and purely mechanical material models. If this command line is applied to blocks using energy-dependent materials, only the determination of the characteristic length is affected. If this command line is applied to an element block with a purely mechanical model and the OLD option is used, the Pronto3D-based artificial viscosity, time step, and eigenvalue estimate will be used in the element calculations. If, however, the NEW option is used, the artificial viscosity and time step will be computed from equations associated with the energy-dependent models. You should consult Reference 8 for further details about the critical time-step calculations and the use of this command line.

6.1.5.9 Activation/Deactivation of Element Blocks by Time

```
ACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
  <string list>period_names
INACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
  <string list>period_names
```

This command line permits the activation and deactivation of element blocks by time period. The time periods are defined in the TIME STEPPING BLOCK command block (Section 3.1.1) within a specific procedure named in a PRESTO PROCEDURE or an ADAGIO PROCEDURE command block (Section 2.2.1).

The ACTIVE FOR PROCEDURE or INACTIVE FOR PROCEDURE command lines can optionally be used to deactivate element blocks for a portion of the analysis. If the ACTIVE FOR PROCEDURE command is used, the element block is active for all periods listed for the named procedure, and is deactivated for all time periods that are absent from the list. If the INACTIVE FOR PROCEDURE command is used, the element block is deactivated for all periods listed for the named procedure. The element block is active for all time periods that are absent from the list. If neither command line is used, by default the block is active during all time periods. This command line controls the activation and deactivation of all elements in a block. Alternatively, individual elements can be deactivated with the ELEMENT DEATH command block (see Section 6.5).

Warning: Inactive element's corresponding nodes will not move while they are inactive except in the case that the nodes are shared with active elements. It is therefore possible that inactive elements can be inverted while inactive and trigger a fatal error when activated.

Known Issue: Deactivation of element blocks does not currently work in conjunction with the full tangent preconditioner (see Section 4.3) in Adagio. To use this capability, one of the nodal preconditioners must be used.

6.2 Element Sections

Element sections are defined by section command blocks. There are currently nine different types of section command blocks. The section command blocks appear in the SIERRA scope, at the same level as the FINITE ELEMENT MODEL command block. In general, a section command block has the following form:

```
BEGIN section_type SECTION <string>section_name
  command lines dependent on section type
END [section_type SECTION <string>section_name]
```

Currently, section_type can be SOLID, COHESIVE, SHELL, MEMBRANE, BEAM, TRUSS, SPRING, DAMPER, POINT MASS, PARTICLE, PERIDYNAMICS, or SUPERELEMENT. These various section types are identified as individual section command blocks and are described below. The corresponding section_name parameter in each of these command blocks, e.g., truss_section_name in the TRUSS SECTION command block, is selected by the user. The method used to associate these names with individual SECTION command lines in PARAMETERS FOR BLOCK command blocks is discussed in Section 6.1.5.4.

6.2.1 Solid Section

```
BEGIN SOLID SECTION <string>solid_section_name
  COORDINATE SYSTEM = <string>Coordinate_system_name
  FORMULATION = <string>MEAN_QUADRATURE|
    SELECTIVE_DEVIATORIC|FULLY_INTEGRATED|VOID(MEAN_QUADRATURE)
  DEVIATORIC PARAMETER = <real>deviatoric_param
  STRAIN INCREMENTATION = <string>MIDPOINT_INCREMENT|
    STRONGLY_OBJECTIVE|NODE_BASED(MIDPOINT_INCREMENT)
  NODE BASED ALPHA FACTOR = <real>bulk_stress_weight(0.01)
  NODE BASED BETA FACTOR = <real>shear stress_weight(0.35)
  NODE BASED STABILIZATION METHOD = <string>EFFECTIVE_MODULI|
    MATERIAL(MATERIAL)
  HOURGLASS FORMULATION = <string>TOTAL|INCREMENTAL(INCREMENTAL)
  HOURGLASS INCREMENT = <string>ENDSTEP|MIDSTEP (ENDSTEP)
  HOURGLASS ROTATION = <string> APPROXIMATE|SCALED (APPROXIMATE)
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [SOLID SECTION <string>solid_section_name]
```

The SOLID SECTION command block is used to specify the properties for solid elements (hexahedra and tetrahedra). This command block is to be referenced by an element block made up of solid elements. The two types of solid-element topologies currently supported are hexahedra and tetrahedra. The parameter solid_section_name is user-defined and is referenced by a SECTION command line in a PARAMETERS FOR BLOCK command block.

The COORDINATE SYSTEM command line specifies the name of a coordinate system definition block command that will be used to define a local coordinate system on each element of a block that uses this SOLID SECTION. The coordinate system can then be used to transform element stresses for solid elements from the global coordinate system to this local element coordinate system through the use of a BEGIN USER OUTPUT command block as described in Section out:user. It is also used for defining a local element coordinate system for representative volume analyses (see Section spec:rve).

The FORMULATION command line specifies whether the element will use a single-point integration rule (mean quadrature), a selective-deviatoric rule, or a fully-integrated rule, or act as a void element. The selective-deviatoric and fully-integrated integration rules are higher-order integration schemes, which are discussed below.

If the user wishes to use the selective-deviatoric rule, the DEVIATORIC PARAMETER command line must also appear in the SOLID SECTION command block. The selective-deviatoric parameter, deviatoric_param, which is valid from 0.0 to 1.0, indicates how much of the deviatoric response should be taken from a uniform-gradient integration and how much should be taken from a full integration of the element. A value of 0.0 will give a pure uniform-gradient response with no hourglass control. Thus, this value is of little practical use. A value of 1.0 will give a fully integrated deviatoric response. Although any value between 0.0 and 1.0 is perfectly valid, lower values are generally preferred.

The selective-deviatoric elements, when used with a value greater than 0.0, provide hourglass control without artificial hourglass parameters.

Alternatively, a fully-integrated rule can be defined by setting FORMULATION = FULLY_INTEGRATED. This will apply an eight-point integration scheme for eight-noded hexahedral elements and a four-point integration scheme for ten-noded tetrahedral elements. This definition is currently unavailable for four-noded and eight-noded tetrahedral elements.

The VOID formulation is valid for 8-node hexahedral and 4-node tetrahedral element blocks. Void elements only compute volume. They do not contribute internal forces to the model. The material model and density associated with void elements are ignored. The volume and the first and second derivatives of the volume for each element are stored in the element variables volume, volume_first_derivative, and volume_second_derivative. The volume derivatives are computed using least squares fits of the volume history, which is stored for the previous five time steps.

In addition to the per-element volume and derivatives, the total volume and derivatives of that total volume for all elements in each void element block are written to the results file as global variables. The names for these variables are voidvol_blockID, voidvol_first_deriv_blockID, and voidvol_second_deriv_blockID. In these global variable names, blockID is the ID of the block. For example, the void volume for block 8 would be stored in voidvol_8.

Some of the solid elements support different strain-incrementation formulations. See the element summary at the beginning of Section 6.1 to determine which strain-incrementation formulations are available for which elements. The STRAIN INCREMENTATION command line lets you specify a midpoint-increment strain formulation (MIDPOINT_INCREMENT), a strongly objective strain formulation (STRONGLY_OBJECTIVE), or a node-based formulation (NODE_BASED) for some of the

elements. Consult the element documentation [2,6] for a description of these strain formulations.

The node-based formulation can only be used with four-node tetrahedral elements. If your element-block command block (i.e., a `PARAMETERS FOR BLOCK` command block) has a `SECTION` command line that references a `SOLID SECTION` command block that uses:

```
STRAIN INCREMENTATION = NODE_BASED
```

then the element block must be a block of four-node tetrahedral elements.

The node-based formulation lets you calculate a solution that is some mixture of an element-based formulation (information from the center of an element) and a node-based formulation (information at a node that is based on all elements attached to the node). The node-based tetrahedron allows the user to model with four-node tetrahedral elements and avoid the main problems with regular tetrahedral elements. Regular tetrahedral elements are much too stiff and can produce very inaccurate results.

You can adjust the mixture of node-based versus element-based information incorporated into your solution with the `NODE BASED ALPHA FACTOR` and `NODE BASED BETA FACTOR` command lines. These two lines apply only if you have selected the `NODE_BASED` option on the `STRAIN INCREMENTATION` command line. The value for `bulk_stress_weight` on the `NODE BASED ALPHA FACTOR` command line sets the element bulk stress weighting factor, while the value for `shear_stress_weight` on the `NODE BASED BETA FACTOR` command line sets the element shear stress weighting factor. You should consult Reference 3 to better understand the use of these weighting factors. If both of these factors are set to 0.0, you will be using a strictly node-based formulation. If both of these factors are set to 1.0, you will be using a strictly element-based formulation.

The mixing of node-based and element-based solutions for the node-based formulation is a stabilization technique to eliminate zero energy modes of a pure node-based formulation. With this in mind, two options are provided for the stress update of the element-based solution through the `NODE BASED STABILIZATION METHOD` command (applicable only with `STRAIN INCREMENTATION = NODE_BASED`). The default option, `MATERIAL`, runs the same material model in the element as is run at the nodes, with independent material states. The other option, `EFFECTIVE_MODULI`, uses a linearized update for the element stress state based on the element average value of the nodal effective moduli. The nodal effective moduli are computed from the nodal stress and strain increments at each time step. This second option avoids potential issues due to having independent node and element material states.

The `HOURGLASS FORMULATION` command is used to switch between total and incremental forms of hourglass control. This option can only be used with eight-noded uniform-gradient hexahedral elements using strongly objective strain incrementation (`STRAIN INCREMENTATION = STRONGLY_OBJECTIVE`). One of the following two arguments can be used with this command: `TOTAL` or `INCREMENTAL`. The total formulation performs stiffness hourglass force updates based on the rotation tensor from the polar decomposition of the total deformation gradient. The incremental formulation is the default and performs stiffness hourglass force updates based on the hourglass velocities and the incremental rotation tensor. The viscous hourglass forces and the hourglass parameters are unchanged by this command. Consult the element documentation [6] for a description of the hourglass forces and the incremental hourglass formulation.

The HOURGLASS INCREMENT and HOURGLASS ROTATION commands control the speed and accuracy of the hourglass control computation. These commands are only applicable to the uniform gradient hex with midpoint strain incrementation (STRAIN INCREMENTATION = MIDPOINT_INCREMENT). The HOURGLASS INCREMENT line command specifies whether the hourglass resistance increment is to be computed at the middle or end of the time step. The end-step calculation has a slightly reduced computational cost while the mid-step computation is more accurate. The default is ENDSTEP. The HOURGLASS ROTATION command controls whether the hourglass resistance will be scaled after rotation to preserve the magnitude. Scaling requires additional computation time but will be more accurate, particularly when very large rotations are present in the analysis. The default is APPROXIMATE, meaning no scaling is done.

Rigid elements in a section are indicated by including the RIGID BODY command line. The RIGID BODY command line specifies an identifier that maps to a rigid body command block. See Section 6.3.1 for a full discussion of how to create rigid bodies and Section 6.3.1.1 for information on the use of the RIGID BODIES FROM ATTRIBUTES command.

Known Issue: For problems with large rotations, the hourglass energies are known to spike exponentially with increases in total rotation. This behavior is observed for both midpoint and strongly objective strain incrementation, and for both incremental and total hourglass formulations. The large rotation issue is a known limitation in solid mechanics for all non-hyperelastic material models.

6.2.2 Cohesive Section

```
BEGIN COHESIVE SECTION <string>cohesive_section_name
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points(1)
END [COHESIVE SECTION <string>cohesive_section_name]
```

Warning: In order for the cohesive elements to contribute to the global stable time step, a node based time step must be used. See Section 3.2.3 for details.

The COHESIVE SECTION command block is used to specify the properties for cohesive zone elements (quadrilateral and triangular). The name of this block (given by cohesive_section_name) is referenced by the element block for cohesive elements. If the option for adaptive insertion of cohesive zone elements is used, the name of this block is referenced by the COHESIVE SECTION command defined in Section 6.5.5.

```
NUMBER OF INTEGRATION POINTS = <integer>num_int_points(1)
```

The default number of integration points for a cohesive element is one. However, it should be noted that with a single integration point, spurious hour-glass like modes can be introduced to the deformation of the cohesive element. Currently, the quadrilateral cohesive element supports one and four integration points while the triangular cohesive element supports one and three integration points.

6.2.3 Localization Section

```
BEGIN LOCALIZATION SECTION <string>localization_section_name
  MEAN DILATATION FORMULATION
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points
  THICKNESS = <real>thickness
END [LOCALIZATION SECTION <string>cohesive_section_name]
```

Warning: Currently the localization elements do not contribute to the global stable time step. As such, the user must specify a time step via the `INITIAL TIME STEP` and `TIME STEP INCREASE FACTOR` command lines. See Section 3.1 for details.

Localization elements are planar elements that lie between bulk (volumetric) elements and can employ the same underlying bulk material model. Topologically, localization elements are identical to cohesive surface elements. The reason this 2D element can access 3D material models is due to the multiplicative decomposition of the deformation gradient $\boldsymbol{F}$ such that $\boldsymbol{F} = \boldsymbol{F}^{\parallel}\boldsymbol{F}^{\perp}$ where $\boldsymbol{F}^{\parallel}$ encapsulates in-plane stretching and $\boldsymbol{F}^{\perp}$ is defined to be

$$\boldsymbol{F}^{\perp} = \boldsymbol{I} + \frac{\boldsymbol{\Delta}}{h} \otimes \boldsymbol{N} \tag{6.1}$$

where $\boldsymbol{N}$ is the normal to the mid-plane, $\boldsymbol{\Delta}$ is the gap vector, and h is the element “thickness” or the length scale governing the sub-grid separation progress. We note that because the length scale h is independent of the discretization, the methodology is regularized and ideal for employing local, softening material models to simulate the failure process. For full details on the theory, please see 15.

The `LOCALIZATION SECTION` command block is used to specify the properties for localization elements (quadrilateral and triangular). The name of this block (given by `localization_section_name`) is referenced by the element block for localization elements.

`MEAN DILATATION FORMULATION` This command line will yield a constant pressure formulation. The average pressure is obtained at all integration points through a modification of the kinematic quantities. For hypo-elastic materials, we volume average the tr[$\boldsymbol{D}$]. For uncoupled hyperelastic models, we volume average det[$\boldsymbol{F}$]. After voluming averaging tr[$\boldsymbol{D}$] and det[$\boldsymbol{F}$], we remove local dilatational contributions and additively (hypoelastic) or multiplicatively (hyperelastic) include the average response. This results in an average pressure (by construction) and has been shown to be effective in avoiding element locking during isochoric deformations. Note that one can use a single integration point to achieve a constant pressure but we do not provide any hourglass control to suppress spurious modes. If isochoric deformations are of concern, we recommend using a fully-integrated element with `MEAN DILATATION FORMULATION`.

`NUMBER OF INTEGRATION POINTS = <integer>num_int_points` The default number of integration points for a localization element depends on the topology, but is always sufficient for full integration. For an 8 noded hex (planar 4 node quad) the default is 4 integration points while for a 6 noded wedge (planar 3 node tri) the default is 1 integration point. However, it should be noted that with a single integration point, spurious hourglass modes can be introduced into the deformation of the quadrilateral localization element. Currently, the quadrilateral localization

element supports one and four integration points while the triangular localization element supports one integration point.

`THICKNESS = <real>thickness` The `THICKNESS` command line sets the localization element thickness *h*. In many respects, *h* should be considered a material parameter as it governs the evolution of surface separation. We note that the introduction of *h* generates the true length scale in the problem, the process zone size. Care must be taken to adequately resolve the process zone size or the methodology will not be regularized.

6.2.4 Shell Section

```
BEGIN SHELL SECTION <string>shell_section_name
  THICKNESS = <real>shell_thickness
  THICKNESS MESH VARIABLE =
    <string>THICKNESS|<string>var_name
  THICKNESS TIME STEP = <real>time_value
  THICKNESS SCALE FACTOR = <real>thick_scale_factor(1.0)
  INTEGRATION RULE = TRAPEZOID|GAUSS|LOBATTO|SIMPSONS|
    USER|ANALYTIC|DEFAULT(DEFAULT)
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points(5)
  FORMULATION = KH_SHELL|BT_SHELL|NQUAD(BT_SHELL)
  BEGIN USER INTEGRATION RULE
    <real>location_1 <real>weight_1
    <real>location_2 <real>weight_2
    .
    .
    <real>location_n <real>weight_n
  END [USER INTEGRATION RULE]
  LOFTING FACTOR = <real>lofting_factor(0.5)
  OFFSET MESH VARIABLE = <string>var_name
  ORIENTATION = <string>orientation_name
  DRILLING STIFFNESS FACTOR = <real>stiffness_factor(1.0e-5)
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [SHELL SECTION <string>shell_section_name]
```

The `SHELL SECTION` command block is used to specify the properties for a shell element. If this command block is referenced in an element block of three-dimensional, four-node elements, the elements in the block will be treated as shell elements. The parameter, `shell_section_name`, is user-defined and is referenced by a `SECTION` command line in a `PARAMETERS FOR BLOCK` command block.

Either a `THICKNESS` command line or a `THICKNESS MESH VARIABLE` command line must appear in the `SHELL SECTION` command block.

If a shell element block references a `SHELL SECTION` command block with the command line:

```
THICKNESS = <real>shell_thickness
```

then all the shell elements in the block will have their thickness initialized to the value `shell_thickness`.

Sierra/SM can also initialize the thickness using an attribute defined on elements in the mesh file. Meshing programs such as PATRAN and CUBIT typically set the element thickness as an attribute on the elements. If the elements have one and only one attribute defined on the mesh, the `THICKNESS MESH VARIABLE` command line should be specified as:

```
THICKNESS MESH VARIABLE = THICKNESS
```

which causes the thickness of the element to be initialized to the value of the attribute for that element. If there are zero attributes or more than one attribute, the thickness variable will not be automatically defined, and the command will fail.

The thickness may also be initialized by any other field present on the input mesh. To specify a field other than the single-element attribute, use this form of the `THICKNESS MESH VARIABLE` command line:

```
THICKNESS MESH VARIABLE = <string>var_name
```

Here, the string `var_name` is the name of the initializing field.

The input mesh may have values defined at more than one point in time. To choose the point in time in the mesh file that the variable should be read, use the command line:

```
THICKNESS TIME STEP = <real>time_value
```

The default time point in the mesh file at which the variable is read is 0.0.

Once the thickness of a shell element is initialized by using either the `THICKNESS` command line or the `THICKNESS MESH VARIABLE` command line, this initial thickness value can then be scaled using the scale-factor command line:

```
THICKNESS SCALE FACTOR = <real>thick_scale_factor
```

If the initial thickness of the shell is 0.15 inch, and the value for `thick_scale_factor` is 0.5, then the scaled thickness of the membrane will be 0.075.

The thickness mesh variable specification may be coupled with the `THICKNESS SCALE FACTOR` command line. In this case, the thickness mesh variable is scaled by the specified factor.

The shell formulation can be selected via the `FORMULATION` command line. Several options are available:

- The `BT_SHELL` option is used to select the Belytschko-Tsay (BT) shell formulation. This is the default. The BT shell formulation has constant transverse shear, and is the fastest shell element available.
- The `KH_SHELL` option specifies the the Key-Hoff shell formulation, which includes a non-linear transverse shear term. The Key-Hoff shell can better solve problems with high gradients of transverse shear (such as twisted beams), but distorted elements can be excessively stiff.

- The NQUAD option selects a formulation for a completely elastic plate.
- The SEVEN_PARAMETER option selects an experimental shell formulation.
- The HEXSHELL option selects a experimental shell element that treats hexahedral elements with a shell-based formulation.

For shell elements, the user can select from a number of integration rules, including a user-defined integration option. The integration rule is selected with the command line:

```
  INTEGRATION RULE =
<string>TRAPEZOID|GAUSS|LOBATTO|SIMPSONS|ANALYTIC|DEFAULT
       USER(DEFAULT)
```

Consult the element documentation [6] for a description of different integration schemes for shell elements.

The default integration scheme for most material models is TRAPEZOID with five integration points through the thickness. The default integration scheme for the elastic material model and the fiber shell material model is ANALYTIC, which is an analytic through-thickness integration scheme. This scheme is faster and more accurate than any of the pointwise rules. Currently the analytic integration scheme is only available for the elastic material model when using the BT_SHELL shell formulation for quad shell elements, and for the fiber shell using the BT_SHELL formulation with one integration point allocated for storage of the inelastic fiber material parameters.

The number of integration points for the piecewise integration point rules can be set to any number greater than one by using the following command line:

```
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points(5)
```

The SIMPSONS, GAUSS, and LOBATTO integration schemes in the INTEGRATION RULE command line all default to five integration points. The number of integration points for these three schemes can be reset by using the NUMBER OF INTEGRATION POINTS command line. There are limitations on the number of integration points for some of these integration rules. The SIMPSONS rule can be set to any number greater than one, the GAUSS scheme can be set to one through seven integration points, and the LOBATTO integration scheme can be set to two through seven integration points.

In addition to these standard integration schemes, you may also define an integration scheme by using the USER INTEGRATION RULE command block.

```
    BEGIN USER INTEGRATION RULE
      <real>location_1 <real>weight_1
      <real>location_2 <real>weight_2
      .
      .
      <real>location_n <real>weight_n
    END [USER INTEGRATION RULE]
```

You may NOT specify both a standard integration scheme and a user scheme. If the USER option is specified in the INTEGRATION RULE command line, a set of integration locations with associated weight factors must be specified. This is done with tabular input command lines inside the USER INTEGRATION RULE command block. The number of command lines inside this command block should match the number of integration points specified in the NUMBER OF INTEGRATION POINTS command line. For example, suppose we wish to use a user-defined scheme with three integration points. The NUMBER OF INTEGRATION POINTS command line should specify three (3) integration points and the number of command lines inside the USER INTEGRATION RULE command block should be three (to give three locations and three weight factors).

For the user-defined rule, the integration point locations should fall between –1 and +1, and the weights should sum to 1.0.

The command line

```
LOFTING FACTOR = <real>lofting_factor(0.5)
```

allows the user to shift the location of the mid-surface of a shell element relative to the geometric location of the shell element. The lofting factor must be greater than or equal to 0.0 and less than or equal to 1.0. By default, the geometric location of a shell element in a mesh represents the mid-surface of the shell. If a shell has a thickness of 0.2 inch, the top surface of the shell is 0.1 inch above the geometric surface defined by the shell element, and the bottom surface of the shell is 0.1 inch below the geometric surface defined by the shell element. (The top surface of the shell is the surface with a positive element normal; the bottom surface of the shell is the surface with a negative element normal.)

Figure 6.2 shows an edge-on view of shell elements with a thickness of t and the location of the geometric plane in relation to the shell surfaces for three different values of the lofting factor—0.0, 0.5, and 1.0. For a lofting factor of 0.0, the geometric surface defined by the shell corresponds to the top surface of the shell element. A lofting factor of 1.0 puts the geometric surface at the bottom surface of the shell element. The geometric surface is midway between the top and bottom surfaces for a lofting factor of 0.5, which is the default. Lofting factors greater than 1.0 or less than 0.0 would put the geometric surface outside the shell element and are not allowed.

Consider the example of a lofting factor set to 1.0 for a shell with thickness of 0.2 inch. In this case, the top surface of the shell will be located at a distance of 0.2 inch from the geometric surface (measuring in the direction of the positive shell normal), and the bottom surface will be located at the geometric surface.

Both the shell mechanics and contact use shell lofting. See Section 8.2 for a discussion of lofting surfaces for shells and contact. It is recommended that shell lofting values other than 0.5 not be used if the shell is thick. If the shell is thicker than its in-plane width, the shell lofting algorithms may become unstable.

Lofting as described above may also be implemented through

```
OFFSET MESH VARIABLE = <string>var_name.
```

This command allows an offset value to be read from an attribute (or variable) on the mesh file. This attribute (or variable) must be named but may have any name, and this name is specified in place of var_name in the command. An offset is a dimensional value (i.e. not scaled by the shell

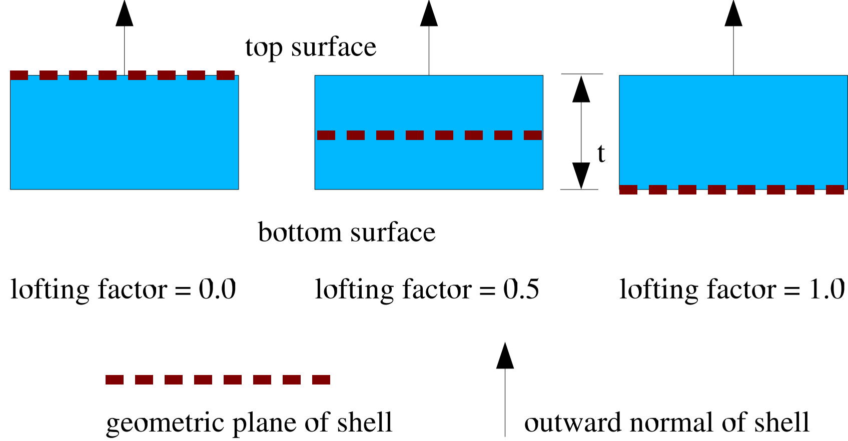


Figure 6.2: Location of geometric plane of shell for various lofting factors.

thickness) that gives the shell mid-plane shift in the positive shell normal direction. Internal to Sierra/SM the offset value is converted to an equivalent lofting factor by dividing by the initial thickness and adding 0.5. Thus, an offset of zero gives a lofting factor of 0.5, an offset of one half of the thickness gives a lofting factor of one, and an offset of negative one half of the thickness gives a lofting factor of zero. There is no check that given offset values produce lofting values between zero and one, which allows any offset to be specified but requires that care be exercised to avoid unstable lofted shells.

Warning: An offset and a lofting factor may not be specified in the same shell section. Both determine the shell lofting and may conflict.

The `ORIENTATION` command line lets you select a coordinate system for output of in-plane stresses and strains. The `ORIENTATION` option makes use of an embedded coordinate system *rst* associated with each shell element. The *rst* coordinate system for a shell element is shown in Figure 6.3. The *r*-axis extends from the center of the shell to the midpoint of the side of the shell defined by nodes 1 and 2. The *t*-axis is located at the center of the shell and is normal to the surface of the shell at the center point. The *s*-axis is the cross-product of the *t*-axis and the *r*-axis. The *rst*-axes form a local coordinate system at the center of the shell; this local coordinate system moves with the shell element as the element deforms.

The `ORIENTATION` command line in the `SHELL SECTION` command block references an `ORIENTATION` command block that appears in the SIERRA scope. As described in Chapter 2 of this document, the `ORIENTATION` command block can be used to define a local co-rotational coordinate system $X''Y''Z''$ at the center of a shell element. In the original shell configuration (time 0), one of the axes—X'', Y'', or Z''—is projected onto the plane of the shell element. The angle between this projected axis of the $X''Y''Z''$ coordinate system and the *r*-axis is used to establish the transformation for in-plane stresses and strains. We will illustrate this with an example.

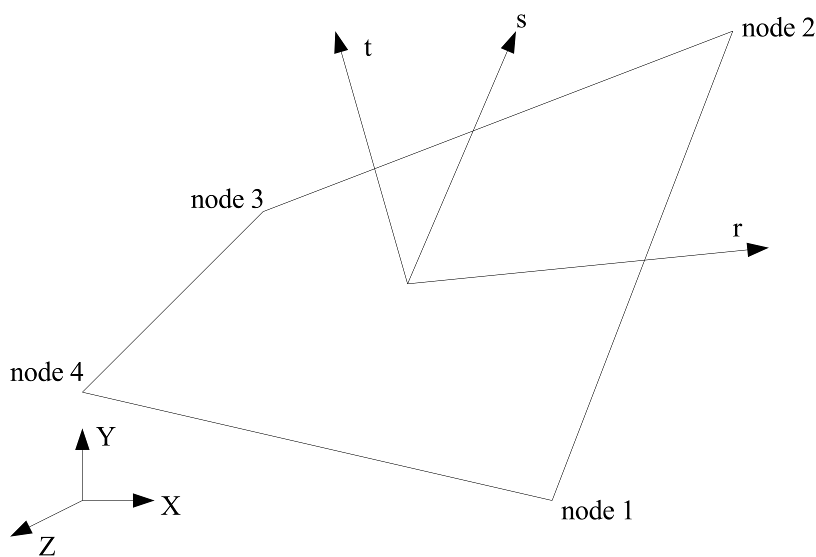


Figure 6.3: Local *rst* coordinate system for a shell element.

Suppose that in our `ORIENTATION` command block we have specified a rotation of 30 degrees about the 1-axis (X'-axis). The command line for this rotation in the `ORIENTATION` command block would be:

```
ROTATION ABOUT 1 = 30
```

For this case, we project the Y''-axis onto the plane of the shell (Figure 6.4). The angle between this projection and the r-axis establishes a transformation for the in-plane stresses of the shell (the stresses in the center of the shell lying in the plane of the shell). What will be output as the in-plane stress σ_{xx}^{ip} will be in the Y''-direction; what will be output as the in-plane stress σ_{yy}^{ip} will be in the Z''-direction. The in-plane stress σ_{xy}^{ip} is a shear stress in the $Y''Z''$-plane. The $X''Y''Z''$ coordinate system maintains the same relative position in regard to the rst coordinate system. This means that the $X''Y''Z''$ coordinate system is a local coordinate system that moves with the shell element as the element deforms.

The following permutations for output of the in-plane stresses occur depending on the axis (1, 2, or 3) specified in the `ROTATION ABOUT` command line:

- Rotation about the 1-axis (X'-axis): The in-plane stress σ_{xx}^{ip} will be in the Y''-direction; the in-plane stress σ_{yy}^{ip} will be in the Z''-direction. The in-plane stress σ_{xy}^{ip} is a shear stress in the $Y''Z''$-plane.

- Rotation about the 2-axis (Y'-axis): The in-plane stress σ_{xx}^{ip} will be in the Z''-direction; the in-plane stress σ_{yy}^{ip} will be in the X''-direction. The in-plane stress σ_{xy}^{ip} is a shear stress in the $Z''X''$-plane.

- Rotation about the 3-axis (Z'-axis): The in-plane stress σ_{xx}^{ip} will be in the X''-direction; the in-plane stress σ_{yy}^{ip} will be in the Y''-direction. The in-plane stress σ_{xy}^{ip} is a shear stress in the $X''Y''$-plane.

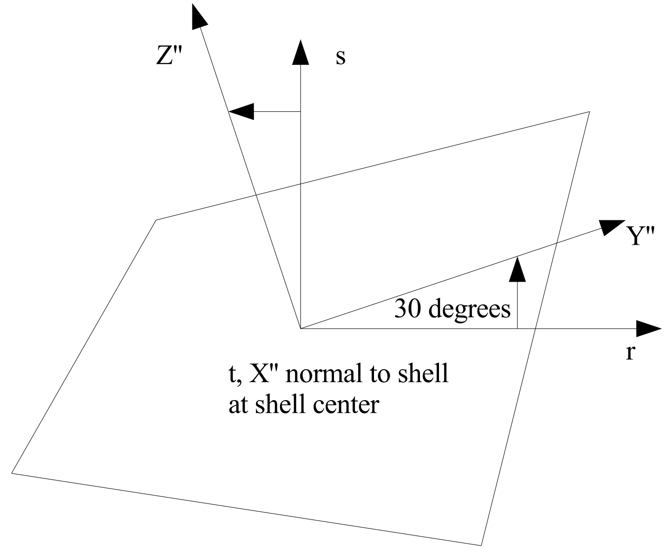


Figure 6.4: Rotation of 30 degrees about the 1-axis (X'-axis).

If no orientation is specified, the in-plane stresses and strains are output in the consistent axes from a default orientation, which is a rectangular system with the X' and Y' axes taken as the global X and Y axes, respectively, and `ROTATION ABOUT 1 = 0.0`.

The command line

```
DRILLING STIFFNESS FACTOR = <real>stiffness_factor
```

adds stiffness in the drilling degrees of freedom to quadrilateral shells. Drilling degrees of freedom are rotational degrees of freedom in the direction orthogonal to the plane of the shell at each node. The formulation used for the quadrilateral shells has no rotational stiffness in this direction. This can lead to spurious zero-energy modes of deformation similar in nature to hourglass deformation. This makes obtaining a solution difficult in quasistatic problems and can result in singularities when using the full tangent preconditioner.

The `stiffness_factor` should be chosen as a quantity small enough to add enough stiffness to allow the solve to be successful without unduly affecting the solution. The default value for `stiffness_factor` is 1.0e-5, which places drilling stiffness about 10000X lower than the bending stiffness. If singularities are encountered in the solution or hourglass-like deformation is observed in the drilling degrees of freedom, setting a larger value may be appropriate.

Elements in a section can be made rigid by including the `RIGID BODY` command line. The `RIGID BODY` command line specifies an indenter that maps to a rigid body command block. See Section 6.3.1 for a full discussion of how to create rigid bodies and Section 6.3.1.1 for information on the use of the `RIGID BODIES FROM ATTRIBUTES` command.

6.2.5 Membrane Section

```
BEGIN MEMBRANE SECTION <string>membrane_section_name
```

```
    THICKNESS = <real>mem_thickness
    THICKNESS MESH VARIABLE =
      <string>THICKNESS|<string>var_name
    THICKNESS TIME STEP = <real>time_value
    THICKNESS SCALE FACTOR = <real>thick_scale_factor(1.0)
    FORMULATION = <string>MEAN_QUADRATURE|
      SELECTIVE_DEVIATORIC(MEAN_QUADRATURE)
    DEVIATORIC PARAMETER = <real>deviatoric_param
    LOFTING FACTOR = <real>lofting_factor(0.5)
    RIGID BODY = <string>rigid_body_name
    RIGID BODIES FROM ATTRIBUTES = <integer>first_id
      TO <integer>last_id
  END [MEMBRANE SECTION <string>membrane_section_name]
```

The MEMBRANE SECTION command block is used to specify the properties for a membrane element. If a section defined by this command block is referenced in the parameters for a block of four-noded elements, the elements in that block will be treated as membranes. The parameter membrane_section_name is user-defined and is referenced by a SECTION command line in a PARAMETERS FOR BLOCK command block.

Either a THICKNESS command line or a THICKNESS MESH VARIABLE command line must appear in the MEMBRANE SECTION command block.

If a membrane element block references a MEMBRANE SECTION command block with the command line:

```
THICKNESS = <real>mem_thickness
```

then all the membrane elements in the block will have their thickness initialized to the value mem_thickness.

Sierra/SM can also initialize the thickness using an attribute defined on elements in the mesh file. Meshing programs such as PATRAN and CUBIT typically set the element thickness as an attribute on the elements. If the elements have one and only one attribute defined on the mesh, the THICKNESS MESH VARIABLE command line should be specified as:

```
THICKNESS MESH VARIABLE = THICKNESS
```

which causes the thickness of the element to be initialized to the value of the attribute for that element. If there are zero attributes or more than one attribute, the thickness variable will not be automatically defined, and the command will fail.

The thickness may also be initialized by any other field present on the input mesh. To specify a field other than the single-element attribute, use this form of the THICKNESS MESH VARIABLE command line:

```
THICKNESS MESH VARIABLE = <string>var_name
```

where the string var_name is the name of the initializing field.

The input mesh may have values defined at more than one point in time. To choose the point in time in the mesh file that the variable should be read, use the command line:

```
THICKNESS TIME STEP = <real>time_value
```

The default time point in the mesh file at which the variable is read is 0.0.

Once the thickness of a membrane element is initialized by using either the `THICKNESS` command line or the `THICKNESS MESH VARIABLE` command line, this initial thickness value can then be scaled by using the scale-factor command line:

```
THICKNESS SCALE FACTOR = <real>thick_scale_factor
```

If the initial thickness of the membrane is 0.15 inch, and the value for `thick_scale_factor` is 0.5, then the scaled thickness of the membrane will be 0.075.

The `FORMULATION` command line specifies whether the element will use a single-point integration rule (mean quadrature) or a selective-deviatoric integration rule:

```
FORMULATION = <string>MEAN_QUADRATURE|SELECTIVE_DEVIATORIC
  (MEAN_QUADRATURE)
```

If the user wishes to use the selective-deviatoric rule, the `DEVIATORIC PARAMETER` command line must also appear in the `MEMBRANE SECTION` command block:

```
DEVIATORIC PARAMETER = <real>deviatoric_param
```

The selective-deviatoric elements, when used with a parameter greater than 0.0, provide hourglass control without artificial hourglass parameters. The selective-deviatoric parameter, `deviatoric_param`, which is valid from 0.0 to 1.0, indicates how much of the deviatoric response should be taken from a uniform-gradient integration and how much should be taken from a full integration of the element. A value of 0.0 will give a pure uniform-gradient response with no hourglass control. Thus, this value is of little practical use. A value of 1.0 will give a fully integrated deviatoric response. Although any value between 0.0 and 1.0 is perfectly valid, lower values are generally preferred.

The command line

```
LOFTING FACTOR = <real>lofting_factor(0.5)
```

allows the user to shift the location of the mid-surface of a membrane element relative to the geometric location of the membrane element. By default, the geometric location of a membrane element in a mesh represents the mid-surface of the membrane. If a membrane has a thickness of 0.2 inch, the top surface of the membrane is 0.1 inch above the geometric surface defined by the membrane element, and the bottom surface of the membrane is 0.1 inch below the geometric surface defined by the membrane element. (The top surface of the membrane is the surface with a positive element normal; the bottom surface of the membrane is the surface with a negative element normal.)

Figure 6.2, which shows lofting for shells, is also applicable to membranes. For membranes, Figure 6.2 represents an edge-on view of membrane elements with a thickness of *t* and the location of the geometric plane in relation to the membrane surfaces for three different values of the lofting factor—0.0, 0.5, and 1.0. For a lofting factor of 0.0, the geometric surface defined by the membrane corresponds to the top surface of the membrane element. A lofting factor of 1.0 puts the geometric

surface at the bottom surface of the membrane element. The geometric surface is midway between the top and bottom surfaces for a lofting factor of 0.5, which is the default.

Consider the example of a lofting factor set to 1.0 for a membrane with thickness of 0.2 inch. In this case, the top surface of the membrane will be located at a distance of 0.2 inch from the geometric surface (measuring in the direction of the positive shell normal), and the bottom surface will be located at the geometric surface.

Both the membrane mechanics and contact use membrane lofting. See Section 8.2 for a discussion of lofting surfaces for membranes and contact.

Elements in a section can be made rigid by including the `RIGID BODY` command line. The `RIGID BODY` command line specifies an indenter that maps to a rigid body command block. Consult Section 6.3.1 for a full description of how to create rigid bodies and Section 6.3.1.1 for information on the use of the `RIGID BODIES FROM ATTRIBUTES` command.

6.2.6 Beam Section

```
BEGIN BEAM SECTION <string>beam_section_name
  SECTION = <string>ROD|TUBE|BAR|BOX|I
  WIDTH = <real>section_width
  WIDTH VARIABLE = <string>width_var
  HEIGHT = <real>section_width
  HEIGHT VARIABLE= <string>height_var
  WALL THICKNESS = <real>wall_thickness
  WALL THICKNESS VARIABLE = <string>wall_thickness_var
  FLANGE THICKNESS = <real>flange_thickness
  FLANGE THICKNESS VARIABLE = <string>flange_thickness_var
  T AXIS = <real>tx <real>ty <real>tz
  T AXIS VARIABLE = <string>t_axis_var
  REFERENCE AXIS = <string>CENTER|RIGHT|
    TOP|LEFT|BOTTOM(CENTER)
  AXIS OFFSET = <real>s_offset <real>t_offset
  AXIS OFFSET GLOBAL = <real>x_offset <real>y_offset <real>z_offset
  AXIS OFFSET VARIABLE = <string>axis_offset_var
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [BEAM SECTION <string>beam_section_name]
```

The BEAM SECTION command block is used to specify the properties for a beam element. If this command block is referenced in an element block of three-dimensional, two-node elements, the elements in the block will be treated as beam elements. The parameter, beam_section_name, is user-defined and is referenced by a SECTION command line in a PARAMETERS FOR BLOCK command block.

Five different cross sections can be specified for the beam—ROD, TUBE, BAR, BOX, and I—via use of the SECTION command line. Each section requires a specific set of command lines for a complete geometric description. The command lines related to section geometry are WIDTH, HEIGHT, WALL THICKNESS, and FLANGE THICKNESS. We present a summary of the geometric parameter command lines required for each section as a quick reference.

- If the section is ROD, the following geometry command lines are required:

  ```
  WIDTH or WIDTH VARIABLE
  HEIGHT or HEIGHT VARIABLE
  ```

- If the section is TUBE, the following geometry command lines are required:

  ```
  WIDTH or WIDTH VARIABLE
  HEIGHT or HEIGHT VARIABLE
  WALL THICKNESS or WALL THICKNESS VARIABLE
  ```

- If the section is `BAR`, the following geometry command lines are required:

```
WIDTH or WIDTH VARIABLE
HEIGHT or HEIGHT VARIABLE
```

- If the section is `BOX`, the following geometry command lines are required:

```
WIDTH or WIDTH VARIABLE
HEIGHT or HEIGHT VARIABLE
WALL THICKNESS or WALL THICKNESS VARIABLE
```

- If the section is `I`, the following geometry command lines are required:

```
WIDTH or WIDTH VARIABLE
HEIGHT or HEIGHT VARIABLE
WALL THICKNESS or WALL THICKNESS VARIABLE
FLANGE THICKNESS or FLANGE THICKNESS VARIABLE
```

Most of the sections require the `T AXIS` or `T AXIS VARIABLE` command line. If the beam has a circular or tube cross section, and the width of the beam exactly equals the height, then the `T_AXIS` need not be specified. If the `T_AXIS` is not specified for one of these circularly symmetric cross sections the code will arbitrarily pick a t axis at each beam that is perpendicular to the the beam.

Beam section parameters can be specified as constant for all beams in the section with commands such as `WIDTH` or `T AXIS`. Alternatively a set of beam parameters that vary from element to element can be specified with variants of these commands with `VARIABLE` at the end, such as `WIDTH VARIABLE` or `T AXIS VARIABLE`. When the `VARIABLE` variants of commands are used, the command specifies the name of an attribute field on the input mesh that contains the parameter. For `WIDTH VARIABLE`, `HEIGHT VARIABLE`, `WALL THICKNESS VARIABLE`, and `FLANGE THICKNESS VARIABLE`, the field should contain one entry per element. For `AXIS OFFSET VARIABLE` the field should contain either two or three entries per element. If the field for the axis offset contains two entries, it specifies the offset in the local s, t coordinate system. If it contains three entries, it specifies the offset in the global x, y, z coordinate system. For `T AXIS VARIABLE` the field should contain three entries per element.

Before presenting details about the various sections, we will discuss the local coordinate system for the beam. (The geometric properties are related to this local coordinate system.) For the beam, it is necessary to specify a local Cartesian coordinate system, which will be designated as r, s, and t. The r-axis lies along the length of the beam and passes through the centroid of the beam. The t-axis is specified by the user as a vector in the global coordinate system. The s-axis is computed from the cross product of the t-axis and the r-axis. The t-axis is then recomputed as the cross product of the r-axis and the s-axis to ensure that the t-axis is orthogonal to the r-axis. These local direction vectors are all normalized, so the user-input vectors do not have to be unit vectors.

If we want the initial position of the t-axis to be parallel to the global Z-axis, then we would use the command line:

```
T AXIS = 0 0 1
```

If we wanted the initial position of the *t*-axis to be parallel to a vector (0.5, 0.8660, 0) in the global coordinate system, then we would use the command line:

```
T AXIS = 0.5 0.8660 0.0
```

The *t*-axis will change position as the beam deforms (rotates about the *r*-axis). This change in position is tracked internally by the computations for the beam element. The `HEIGHT` for the beam cross section is in the direction of the *t*-axis, and the `WIDTH` of the beam cross section is in the direction of the *s*-axis.

Now that the local coordinate system for the beam has been defined, we can describe the definition of each section.

- The `ROD` section is a solid elliptical section. The diameter along the height is specified by the `HEIGHT` command line, and the diameter along the width is specified by the `WIDTH` command line.
- The `TUBE` section is a hollow elliptical section. The diameter along the height is specified by the `HEIGHT` command line, and the diameter along the width is specified by the `WIDTH` command line. The wall thickness for the tube is specified by the `WALL THICKNESS` command line.
- The `BAR` section is a solid rectangular section. The height is specified by the `HEIGHT` command line, and the width is specified by the `WIDTH` command line.
- The `BOX` section is a hollow rectangular section. The height is specified by the `HEIGHT` command line, and the width is specified by the `WIDTH` command line. The wall thickness for the box is specified by the `WALL THICKNESS` command line.
- The `I` section is the standard I-section associated with a beam. The height of the I-section is given by the `HEIGHT` command line, and the width of the flanges is given by the `WIDTH` command line. The thickness of the vertical member is given by the `WALL THICKNESS` command line, and the thickness of the flanges is given by the `FLANGE THICKNESS` command line.

By default, the *r*-axis coincides with the geometric centerline of the beam. The geometric centerline of the beam is defined by the location of the two nodes defining the beam connectivity. It is possible to offset the local *r*-axis, *s*-axis, and *t*-axis from the geometric centerline of the beam. To do this, one can use either the `REFERENCE AXIS` command line or the `AXIS OFFSET` command line, but not both.

The `REFERENCE AXIS` command line has the options `CENTER`, `TOP`, `RIGHT`, `BOTTOM`, and `LEFT`. The `CENTER` option is the default, which means that the *r*-axis coincides with the geometric centerline of the beam. If the `TOP` option is used, the *r*-axis is moved in the direction of the original *t*-axis by a positive distance `HEIGHT/2` from the centroid so that it passes through the top of the beam section (top being defined in the direction of the positive *t*-axis). If the `RIGHT` option is used, the *r*-axis is moved in the direction of the original *s*-axis by a positive distance `WIDTH/2` so that it passes through the right side of the beam section (the section being viewed in the direction of the

negative r-axis). If the BOTTOM option is used, the r-axis is moved in the direction of the original t-axis by a distance HEIGHT/2 so that it passes through the bottom of the beam section (bottom being defined in the direction of the negative t-axis). If the LEFT option is used, the r-axis is moved in the direction of the original s-axis by a negative distance WIDTH/2 so that it passes through the left side of the beam section (the section being viewed in the direction of the negative r-axis). For all options, the s-axis and the t-axis remain parallel to their original positions before the translation of the r-axis.

The AXIS OFFSET command line allows the user to offset the local coordinate system from the geometric centerline by an arbitrary distance. The first parameter on the command line moves the r-axis a distance s_offset from the centroid of the section along the original s-axis. The second parameter on the command line moves the r-axis a distance t_offset from the centroid of the section along the original t-axis. The s-axis and t-axis remain parallel to their original positions before the translation of the r-axis.

Alternatively, the axis offset can be specified in global coordinates using the AXIS OFFSET GLOBAL command. This command takes three parameters, which are the x, y, and z components of the offset in the global coordinate system.

Strains and stresses are computed at the midpoint of the beam. The integration of the stresses over the cross section at the midpoint is used to compute the internal forces in the beam. Each beam section has its own integration scheme. The integration scheme for each of the sections is shown in Figure 6.5 through Figure 6.7. The numbers in these figures show the relative location of the integration points in regard to the centroid of the section and the s-axis and the t-axis.

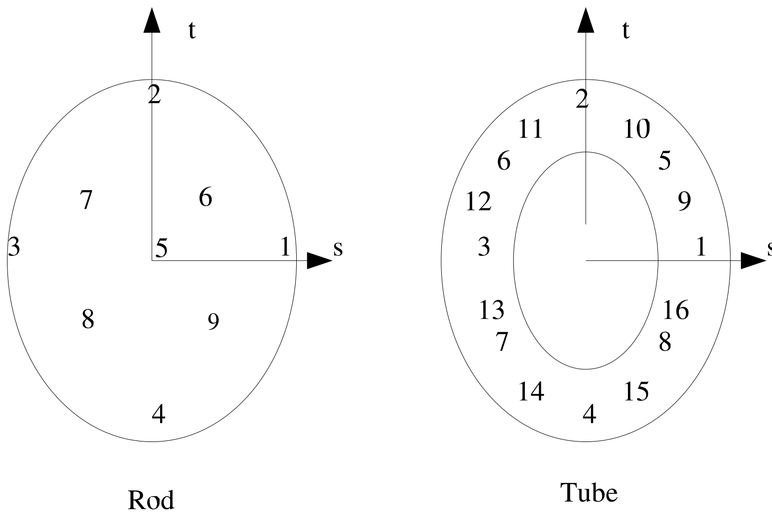


Figure 6.5: Integration points for rod and tube

At each integration point, there is an axial strain (with a corresponding axial stress) and an in-plane (in the plane of the cross section) shear strain (with a corresponding shear stress). The user can output this stress and strain information by using the RESULTS OUTPUT commands described in Chapter 9. The variable that will let users access the strain at the beam integration points is

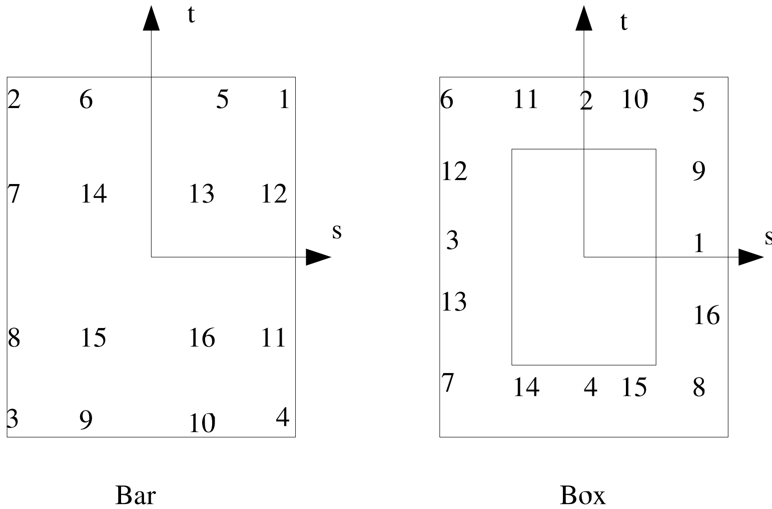


Figure 6.6: Integration points for bar and box.

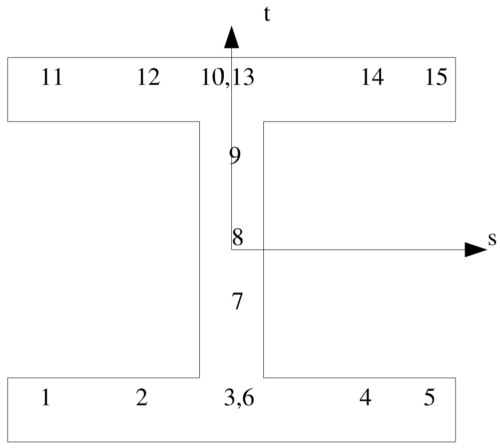


Figure 6.7: Integration points for I-section.

`beam_strain_inc`, and the variable that will let users access the stress at the beam integration points is `stress`. If the user requests output for the beam strain, 32 values are given for the strain. The first value (designated in the output as 01) is the axial strain at the first integration point, the second value (designated in the output as 02) is the shear strain at the first integration point, etc. The odd values for the strain output (01, 03, 05, etc.) are the axial strains at the integration points. The even values of the strain output (02, 04, 06, etc.) are the shear strains at the integration points. For the case where there are only nine integration points (the rod), only the first 18 values for strain have any meaning for the section (the values 19 through 32 are zero). For the I-section, only the first 30 of the strain values have meaning since this section only has 15 integration points. For all other sections, all 32 values have meaning. Output of stress is slightly different than the output

of strain because the stress is stored as a symmetric tensor that contains six components, although four of these components are never used. The axial stress at the first integration point is designated by `stress_xx_01` and the shear stress at the first integration point is `stress_xy_01`. The other four components, `stress_yy_01`, `stress_zz_01`, `stress_yz_01` and `stress_xz_01`, are unused and are set to zero. The stresses at other integration points are named `stress_xx_NN` and `stress_xy_NN`, where `NN` is a number from 01 to 16.

As an alternative for the stress output, you may use the variables `beam_stress_axial` and `beam_stress_shear`. The variable `beam_stress_axial` contains only the axial stresses. The first value associated with `beam_stress_axial` (designated as 01) corresponds to the axial stress at integration point 1, the second value associated with `beam_stress_axial` (designated as 02) corresponds to the axial stress at integration point 2, and so on. The variable `beam_stress_shear` contains only shear stresses. The correlation between numbering the values for `beam_stress_shear` (01, 02, ...) and the integration points is the same as for `beam_stress_axial`.

It is possible to access mean values for the internal forces at the midpoint of the beam. The axial force at the midpoint of the beam is obtained by referencing the variable `beam_axial_force`. The transverse forces at the midpoint of the beam in the *s*-direction and the *t*-direction are obtained by referencing `beam_transverse_force_s` and `beam_transverse_force_t`, respectively. The torsion at the midpoint of the beam (the moment about the *r*-axis), is obtained by referencing `beam_moment_r`. The moments about the *s*-axis and the *t*-axis are obtained by referencing `beam_moment_s` and `beam_moment_t`, respectively.

Elements in a section can be made rigid by including the `RIGID BODY` command line. The `RIGID BODY` command line specifies an identifier that maps to a rigid body command block. See Section 6.3.1 for a full discussion of how to create rigid bodies and Section 6.3.1.1 for information on the use of the `RIGID BODIES FROM ATTRIBUTES` command.

6.2.7 Truss Section

```
BEGIN TRUSS SECTION <string>truss_section_name
  AREA = <real>cross_sectional_area
  INITIAL LOAD = <real>initial_load
  PERIOD = <real>period
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [TRUSS SECTION <string>truss_section_name]
```

The `TRUSS SECTION` command block is used to specify the properties for a truss element. If this command block is referenced in an element block of three-dimensional, two-node elements, the elements in the block will be treated as truss elements. The parameter, `truss_section_name`, is user-defined and is referenced by a `SECTION` command line in a `PARAMETERS FOR BLOCK` command block.

The cross-sectional area for truss elements is specified by the `AREA` command line. The value `cross_sectional_area` is the cross-sectional area of the truss members in the element block.

The truss can be given some initial load over some given time period. The magnitude of the load is specified by the `INITIAL LOAD` command line. If the load is compressive, the sign on the value `initial_load` should be negative; if the load is tensile, the sign on the value `initial_value` should be positive. The period is specified by the `PERIOD` command line.

The initial load is applied over some period by specifying the axial strain rate in the truss, $\dot{\varepsilon}$, over some period p. At some given time t, the strain rate is

$$\dot{\varepsilon} = \frac{ap}{2}\left[1 - \cos\left(\pi t/p\right)\right], \tag{6.2}$$

where

$$a = \frac{2F_i}{EAp}. \tag{6.3}$$

In Equation (6.3), F_i is the initial load, E is the modulus of elasticity for the truss, and A is the area of the truss. Over the period p, the total strain increment generates the desired initial load in the truss.

During the initial load period, the time increments should be reasonably small so that the integration of $\dot{\varepsilon}$ over the period is accurate. The period should be set long enough so that if the model was held in a steady state after time p, there would only be a small amount of oscillation in the load in the truss.

When doing an analysis, you may not want to activate certain boundary conditions until after the prestressing is done. During the prestressing, time-independent boundary conditions such as fixed displacement will most likely be turned on. Time-dependent boundary conditions such as prescribed acceleration or prescribed force will most likely be activated after the prestressing is complete.

Elements in a section can be made rigid by including the `RIGID BODY` command line. The `RIGID BODY` command line specifies an indenter that maps to a rigid body command block. See Section 6.3.1 for a full discussion of how to create rigid bodies and Section 6.3.1.1 for information on the use of the `RIGID BODIES FROM ATTRIBUTES` command.

It should be noted that the axial stress, which is the only stress component for a truss element, is output as a symmetric tensor with six components to be compatible with the notion of volumetric stress. Because of this, for every truss element, six values of stress are stored and output, but only the fist value is ever used. The axial stress is therefore output as `stress_xx` and the other five stress components, `stress_yy`, `stress_zz`, `stress_xy`, `stress_yz`, `stress_zx` are all unused and set to zero.

6.2.8 Spring Section

```
BEGIN SPRING SECTION <string>spring_section_name
  FORCE STRAIN FUNCTION = <string>force_strain_function
```

```
    DEFAULT STIFFNESS <real>default_stiffness
    PRELOAD = <real>preload_value
    PRELOAD DURATION = <real>preload_duration
    RESET INITIAL LENGTH AFTER PRELOAD = <string>NO|YES
    MASS PER UNIT LENGTH = <real>mass_per_unit_length
  END [SPRING SECTION <string>spring_section_name]
```

The SPRING SECTION command block is used to specify the properties for a spring element. If this command block is referenced in an element block of three-dimensional, two node elements, the elements in the block will be treated as spring elements. The parameter, spring_section_name, is user-defined and is referenced by a SECTION command line in a PARAMETERS FOR BLOCK command block.

The spring behavior is governed by the force-engineering strain function which is specified by the FORCE STRAIN FUNCTION command line. The force generated by the spring element is based on the evaluation of the user specified force_strain_function, which has units of F vs dL/L_0 with the current engineering strain of the spring, dL/L_0. This allows the force-strain function to be length independent.

The DEFAULT STIFFNESS command block specifies the spring stiffness used during the first time step and during preload. In all other situations the spring stiffness is based on the slope of the force-strain function evaluated at the previous timestep and the current timestep. The unit of default_stiffness is force.

To specify a preload on the spring both the PRELOAD and PRELOAD DURATION command line must be specified. The PRELOAD command line specifies the magnitude of the preload force, while the PRELOAD DURATION command line specifies how long the preload application should take, in seconds.

An optional preload input, RESET INITIAL LENGTH AFTER PRELOAD, is used when the user would like the initial length of the spring to reset to the displaced length after the preload has occurred. If this command line is not specified, or is set to NO, the initial length of the spring is the undeformed length as calculated from the input mesh.

Springs can optionally have mass through the MASS PER UNIT LENGTH command line. However, this input parameter is required if the spring elements are not individual two-node elements, but rather a string of spring elements, where the inter-spring nodes are only connected to the two adjacent springs. It should be noted, if the mass per unit length is not specified there is no critical timestep calculated for the spring elements.

6.2.9 Damper Section

```
  BEGIN DAMPER SECTION <string>damper_section_name
    AREA = <real>damper_cross_sectional_area
  END [DAMPER SECTION <string>damper_section_name]
```

The DAMPER SECTION command block is used to specify the properties for a damper element. If this command block is referenced in an element block of three-dimensional, two-node elements, the elements in the block will be treated as damper elements. The parameter, damper_section_

name, is user-defined and is referenced by a SECTION command line in a PARAMETERS FOR BLOCK command block.

The cross-sectional area for damper elements is specified by the DAMPER AREA command line. The value damper_cross_sectional_area is the cross-sectional area of the dampers in the element block.

The damper area is used only to generate mass associated with the damper element. The mass is the density for the damper element multiplied by the original volume of the element (original length multiplied by the damper area).

The force generated by the damper element depends on the relative velocity along the current direction vector for the damper element. If $\boldsymbol{n}$ is a unit normal pointing in the direction from node 1 to node 2, if $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the velocity vectors at nodes 1 and 2, respectively, then the force generated by the damper element is

$$F_d = \eta \mathbf{n} \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1) , \tag{6.4}$$

where η is the damping parameter. Currently, the damping parameter must be specified by using an elastic material model for the damper element. The value for Young's modulus in the elastic material model is used for the damping parameter η.

6.2.10 Point Mass Section

```
BEGIN POINT MASS SECTION <string>pointmass_section_name
  VOLUME = <real>volume
  MASS = <real>mass
  IXX = <real>Ixx
  IYY = <real>Iyy
  IZZ = <real>Izz
  IXY = <real>Ixy
  IXZ = <real>Ixz
  IYZ = <real>Iyz
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
  MASS VARIABLE = <string>mass_variable_name
  INERTIA VARIABLE = <string>inertia_variable_name
  OFFSET VARIABLE = <string>offset_variable_name
  ATTRIBUTES VARIABLE NAME = <string>attrib_variable_name
END [POINT MASS SECTION <string>pointmass_section_name]
```

A point mass element is simply a mass at a node, which can be a convenient modeling tool in certain instances. The user can create an element block with one or more point masses. A point mass is a sphere element attached to a single node. A point mass will have its mass added to the mass at the connected node. (Other mass at the node will be derived from mass due to elements attached to the node.) The mass at a node due to a point mass is treated like any other mass at a node derived from an element. The mass due to point mass will be included in body force calculations and kinetic energy calculations, for example.

Point masses are a convenient modeling tool to be used in conjunction with rigid bodies. An element block including one or more point masses can be included in a collection of element blocks used to define a rigid body. The element block of point masses can be used to adjust the total mass and inertia properties for the rigid body. (The point mass element does not have to be used only in conjunction with rigid bodies. One can place a point mass at a node associated with solid or structural elements.)

An element block in which the connectivity for each element contains only one node can be used as a collection of point masses. This command block would have the following form:

```
BEGIN PARAMETERS FOR BLOCK <string>block_id
  MATERIAL = <string>material_name
  SOLID MECHANICS USE MODEL <string> material_model_name
  SECTION = <string>point_mass_section_name
END PARAMETERS FOR BLOCK <string>block_id
```

The element block associated with the point mass must reference a material command block just like any other element block. If the point mass section specifies a volume, then the product of the density specified in the material block and the volume specified in the section block is used to calculate the element mass.

The `VOLUME` command specifies the volume of a point mass element. The mass of the element is then found by multiplying that volume by the material model density.

The `MASS` command explicitly specifies the mass.

The `IXX`, `IYY`, `IZZ`, `IXY`, `IXZ`, `IYZ` commands are used to explicitly define the symmetric inertia tensor entries for the point mass. Note that currently, the trace of the inertial tensor is added to the nodal rotational mass and the rest of the inertia tensor entries are ignored.

The `OFFSET` is used to define the offset of the mass with respect to the connected node. Currently the offset effects only the rotational properties of the point mass. The rotational mass of the point mass is increased by mass times offset distance squared.

The `MASS VARIABLE` specifies that the mass of the point mass elements is to be read from the input mesh file from the variable with the specified name (use the name ”attribute” to read the mass from the mesh block attributes.)

The `INERTIA VARIABLE` specifies that the inertia tensor of the point mass elements is to be read from the input mesh file from the variable with the specified name (use the name ”attribute” to read the inertia tensor from the mesh block attributes.) When this command is used, the mesh file variable should contain six entries per element, ordered as follows: Ixx, Iyy, Izz, Ixy, Ixz, Iyz.

The `OFFSET VARIABLE` specifies that the offset vector of the point mass elements is to be read from the input mesh file from the variable with the specified name (use the name ”attribute” to read the offset vector from the mesh block attributes.) When this command is used, the mesh file variable should contain three entries per element, ordered as follows: Offset_x, Offset_y, Offset_z.

The `ATTRIBUTES VARIABLE NAME` specifies that all properties of the point mass element are to be read from the input mesh file of the variable with the specified name (use the name ”attribute”

to read from the mesh block attributes.) When this command is used, the mesh file variable should contain ten entries per element, ordered as follows: Mass, Ixx, Iyy, Izz, Ixy, Ixz, Iyz, Offset_x, Offset_y, Offset_z.

Multiple ways are provided to specify the raw attributes of mass, inertia and offset for the point mass element. The user must be careful to only use one of these ways to define each property. Using multiple ways to define the mass will result in errors or warnings because of ambiguities.

If you have access to the SEACAS codes from Sandia National Laboratories, you may use the codes in this library to generate element blocks with point mass elements. See Reference 11 for an overview of the SEACAS codes. By using various SEACAS codes, you can easily generate an element block with one or more point masses. For each point mass in the element block, you should create an eight-node hexahedral element that is centered at the point where you want the point mass located. The hexahedron can have arbitrary dimensions, but it is best to work with a unit cube. Suppose you wanted a point mass at (13.5, 27.0, 3.1415). You could create a unit hexahedron (1 by 1 by 1) centered on (13.5, 27.0, 3.1415). The SEACAS program SPHGEN3D will convert the hexahedron to a zero-dimensional element in three-dimensional space located at the center of the hexahedron. For our specific example, the program SPHGEN3D would create a zero-dimensional element, basically an element consisting of a single node, at (13.5, 27.0, 3.1415).

Elements in a section can be made rigid by including the `RIGID BODY` command line. The `RIGID BODY` command line specifies an indenter that maps to a rigid body command block. See Section 6.3.1 for a full discussion of how to create rigid bodies and Section 6.3.1.1 for information on the use of the `RIGID BODIES FROM ATTRIBUTES` command.

6.2.11 Particle Section

```
BEGIN PARTICLE SECTION <string>sph_section_name
  RADIUS MESH VARIABLE = <string>var_name|<string>attribute|
    SPHERE INITIAL RADIUS = <real>rad
  RADIUS MESH VARIABLE TIME STEP = <string>time
  PROBLEM DIMENSION = <integer>1|2|3(3)
  CONSTANT SPHERE RADIUS
  FINAL RADIUS MULTIPLICATION FACTOR = <real>factor(1.0)
  FORMULATION = <string>MASS_PARTICLE|SPH(SPH)
  MONAGHAN EPSILON = <real>monaghan_epsilon(0.0)
  MONAGHAN N = <real>monaghan_n(0.0)
  SPH ALPHAQ PARAMETER = <real>alpha(1.0)
  SPH BETAQ PARAMETER = <real>beta(2.0)
  DENSITY FORMULATION = <string>MATERIAL|KERNEL(MATERIAL)
END [PARTICLE SECTION <string>sph_section_name]
```

SPH (smoothed particle hydrodynamics) is useful for modeling fluids or for modeling materials that undergo extremely large distortions. One must be careful when using SPH for modeling. SPH tends to exhibit both accuracy and stability problems, particularly in tension. An SPH particle interacts with other nearest-neighbor SPH particles based on radius properties of all the elements involved; SPH particles react with other elements, such as tetrahedra, hexahedra, and shells, through contact. You should consult Reference 10 regarding the theoretical background for SPH.

You can define contact interaction between the particles in a particle element block and other element types. In order to do this, you must use the CONTACT NODE SET option described in Section 8.2.2. All of the particles in the particle element block must be included in a node set if you use the CONTACT NODE SET option.

The particle section should only be associated with elements with the SPHERE topology. Sphere elements are point elements that are connected to only a single node.

All of the particles contained in a particle element block must be given some initial radius. There are two options for setting the initial radius for each particle. First, each particle can be given the same radius. To set the radius for each particle in an element block to the same value, use the SPHERE INITIAL RADIUS command line. The parameter rad on this command line sets the radius for all the particles in the element block.

Alternatively, the radius for each particle can be read from a mesh file. The radii can be read from a variable on the mesh file from attributes associated with the element block. If you want to read some variable from the mesh file for the radii, then you would use:

```
RADIUS MESH VARIABLE = sph_radius
```

where sph_radius is the variable name on the mesh file. If you want to use the variable associated with a specific time on the mesh file, you should use the RADIUS MESH VARIABLE TIME STEP command line to select the specific time. If you want to read the attributes associated with the particles, then you should insert the command line

```
RADIUS MESH VARIABLE = attribute
```

(as shown) into the PARTICLE SECTION command block. Pronto3d [1] only offers the attribute option. To compare Presto and Pronto3d results, you should use the attribute option.

If the attribute option is used, each particle should have two attributes defined per element. The first attribute is the particle radius, and the second attribute is the particle volume. The most common way to generate a particle mesh is to first generate a solid element mesh and then use the utility 'sphgen3d' to convert the solid elements into sphere elements. Sphgen3d will convert each solid element into a sphere and place it at the source element centroid. In the output mesh file generated by sphgen3d The radius attribute of the element blocks will be proportional to the source element size and the volume attribute will be the volume of the source element.

The FINAL RADIUS MULTIPLICATION FACTOR command may be used to increase or decrease the initial particle radii. The factor specified will scale the particle radii input with either a constant or a mesh file attribute.

After the radii are initialized, you may determine whether the radii are to remain constant or are to change throughout the analysis. The CONSTANT SPHERE RADIUS command line is an optional command line that prevents the sphere radius from changing over the course of the calculation. By default, the sphere radii will expand or contract based on the changing density in the elements to satisfy the relation that element mass (a constant) equals element volume times element density. If the CONSTANT SPHERE RADIUS command line appears, then the radii for all particles will remain constant.

The DENSITY FORMULATION command controls the method used to update the density, and thus SPH radii, during the analysis. The default MATERIAL option uses the material density and nodal mass to compute a volume for each particle at a given time. The radius is then computed as the cube root of that volume. The alternative KERNEL option computes the current density for each particle using the mass of that particle and the SPH kernel density function. The KERNEL option may be necessary when large expansions of particles are expected (such as when modeling large density changes in gases). The MATERIAL option generally changes particle densities, and thus radii, less than the KERNEL option, so is appropriate for analyses that do not have large density fluctuations.

An analysis problem using SPH may be inherently one-, two-, or three-dimensional. You may indicate whether or not there is some inherent dimensionality in the problem by using the PROBLEM DIMENSION command line. The possible value for this command line is 1 (one-dimensional), 2 (two-dimensional), or 3 (three-dimensional). The default value is 3 for three-dimensional. The internal SPH kernel functions are modified depending on the value set on the PROBLEM DIMENSION command line. If, for example, your problem is inherently two-dimensional in nature, you may get more accurate results by specifying the dimension for your problem as 2 (as opposed to 1 or 3).

The FORMULATION command defines the SPH formulation to use, and has the following options:

- SPH, general smoothed particle hydrodynamics.
- MASS_PARTICLE, Particles only have mass, and no stiffness. Particles do not interact with each other in any way.

The MONAGHAN EPSILON and MONAGHAN N commands set the Monaghan viscosity coefficients. Monaghan viscosity may allow the SPH algorithm to behave better, particularly in tension. Standard values to use are 0.2 for epsilon and 4 for n. See (Reference 14) for more information on the usage of these parameters.

The SPH ALPHAQ PARAMETER and SPH BETAQ PARAMETER commands control the SPH viscosity terms. Larger values lead to more viscosity. Alpha is associated with a viscosity parameter that is proportional to the dilatational modulus of the material. Betaq is associated with a viscosity parameter that is proportional to the change in material density over a time step.

Utility Commands. In addition to the SPH-related command lines just described (which appear in the PARTICLE SECTION command block) there is one other SPH-related command block:

```
BEGIN SPH OPTIONS
  SPH SYMMETRY PLANE <string>+X|+Y|+Z|-X|-Y|-Z
    <real>position_on_axis(0.0)
  SPH DECOUPLE STRAINS: <string>material1 <string>material2
END [SPH OPTIONS]
```

If used, this command block should be placed in the Presto region scope. (All other SPH-related command lines must be nested within the PARTICLE SECTION command block; the PARTICLE

SECTION command block, like all other section command blocks, is in the SIERRA scope.) The symmetry conditions are applied to all SPH element blocks.

The SPH SYMMETRY PLANE command line may be used to reduce model sizes by specifying symmetry planes and modeling only a portion of the model. Due to the nonlocal nature of SPH element integration, symmetry planes cannot be defined with boundary conditions alone; these planes must be explicitly defined. A plane is defined by an outward normal vector aligned with one of the axes (+X, +Y, +Z, -X, -Y, -Z) and some point on the axis, which represents a point in the plane. Suppose for example, the outward normal to the plane of symmetry is in the negative Y-direction (-Y) and the plane of symmetry passes through the y-axis at $y = +2.56$. Then the definition for the symmetry plane would be:

```
SPH SYMMETRY PLANE -Y +2.56
```

The SPH DECOUPLE STRAINS command line prevents two dissimilar materials from directly interacting. Generally, the material properties at a particle are the average of the material properties from nearby particles. If particles with very dissimilar material properties are interacting, this interaction can create problems. The SPH DECOUPLE STRAINS command line ensures that particles with very dissimilar material properties do not directly interact by material-property averaging, but instead just interact with a contact-like interaction. The two material types that are not to interact are specified by the parameters material1 and material2. These parameters will appear as material names on a PROPERTY SPECIFICATION FOR MATERIAL command block.

Display. For purposes of visualizing the element stresses, it may be necessary to copy these element variables into nodal variables. This can easily be done by defining a USER VARIABLE command block (Section 11.2.4) in conjunction with a USER OUTPUT command block (Section 9.2.2). Once the nodal variable is defined, it can be output in a RESULTS OUTPUT command block (Section 9.2.1). An example is provided below. The SPH element blocks for the problem are element blocks 20, 21, and 22. All other element blocks are non-SPH elements.

- In the SIERRA scope:

```
BEGIN USER VARIABLE nodal_stress
  TYPE = NODE SYM_TENSOR LENGTH = 1
END
```

- In the region scope:

```
BEGIN USER OUTPUT
  BLOCK = block_20 block_21 block_22
  COPY ELEMENT VARIABLE stress TO NODAL VARIABLE
    nodal_stress
  END

  BEGIN RESULTS OUTPUT output_presto
    DATABASE NAME = sph.e
```

```
    DATABASE TYPE = exodusII
    AT TIME 0.0 INCREMENT = 1.0e-04
    NODAL nodal_stress
  END RESULTS OUTPUT output_presto
```

Known Issue: For problems with rotations, the SPH algorithm generates rotational spring like forces on the boundary because of known deficiencies in the algorithm in distinguishing between rigid body rotation and deformation. The SPH algorithm does not conserve rotational or angular momentum.

6.2.12 Superelement Section

```
BEGIN SUPERELEMENT SECTION <string>section_name
  BEGIN MAP
    <integer>node_index_1 <integer>component_index_1
    <integer>node_index_2 <integer>component_index_2
    ...
    <integer>node_index_n <integer>component_index_n
  END
  BEGIN STIFFNESS MATRIX
    <real>k_1_1 <real>k_1_2 ...  <real>k_1_n
    <real>k_2_1 <real>k_2_2 ...  <real>k_2_n
    ...         ...         ...  ...
    <real>k_n_1 <real>k_n_2 ...  <real>k_n_n
  END
  BEGIN DAMPING MATRIX
    <real>c_1_1 <real>c_1_2 ...  <real>c_1_n
    <real>c_2_1 <real>c_2_2 ...  <real>c_2_n
    ...         ...         ...  ...
    <real>c_n_1 <real>c_n_2 ...  <real>c_n_n
  END
  BEGIN MASS MATRIX
    <real>m_1_1 <real>m_1_2 ...  <real>m_1_n
    <real>m_2_1 <real>m_2_2 ...  <real>m_2_n
    ...          ...        ...  ...
    <real>m_n_1 <real>m_n_2 ...  <real>m_n_n
  END
  FILE = <string>netcdf_file_name
END [SUPERELEMENT SECTION <string>section_name]
```

A superelement allows definition of an element with a user defined stiffness, damping, and mass matrix. The superelement stiffness is linear and remains constant in time.

The superelement must be represented by an element in the mesh file. A block of elements used to define a superelement must contain exactly one finite element. The finite element that represents the superelement may have any topology. The topology may either be a valid geometric topology (hex, rod, tet, etc.) or may be be an arbitrary topology as defined in the input mesh file. The nodes of a superelement can be shared with other elements, or can be attached only to the superelement. In addition, the superelement can have additional internal degrees of freedom that are not present in the mesh file. If output values are desired on a node, that node must be present in the input mesh file.

Superelement nodes have all the same variables as regular nodes (mass, displacement, velocity, etc.) Only the element time step is defined as an output variable on the superelement itself.

6.2.12.1 Input Commands

```
BEGIN MAP
```

The `MAP` command block defines the mapping from nodal degrees of freedom to the local degrees of freedom in the stiffness and mass matrix for the superelement. The map should contain N pairs of integers, where N is the number of nodes in the superelement. The first integer of each pair is the index of the node in the superelement in the range $0 \ldots N$. The second integer is the component in the range $0 \ldots 6$.

A node index of 0 is a special value and marks the degree of freedom that is internal to the superelement. A superelement may have any number of internal degrees of freedom. Internal degrees of freedom are created internal to the code do not correspond to any nodes actually present in the mesh. A node index greater than zero represents that node index in the element. For example if the superelement was represented by an eight node hex element then node indexes could vary from one to eight and would match the first through eighth nodes in the hex element.

If an internal degree of freedom is used, the component index should be set to 0 along with the node index. If a regular node is used, components 1, 2, and 3 correspond to the X, Y, and Z translational degrees of freedom. Components 4, 5, and 6 correspond to the X, Y, and Z rotational degrees of freedom.

The following is an example superelement definition for a three degree of freedom truss element lying along the x-axis. The superelement is defined in the mesh file using a two node rod element. Degrees of freedom 1 and 3 are mapped to x degrees of freedom of the end nodes of the rod element. Degree of freedom 2 is internal to the superelement.

```
BEGIN SUPERELEMENT SECTION truss_x3
  BEGIN MAP
    1 1
    0 0
    2 1
  END
  BEGIN STIFFNESS MATRIX
    100 -100    0
   -100  200 -100
```

```
      0 -100  100
    END
    BEGIN DAMPING MATRIX
      1 -1  0
     -1  2 -1
      0 -1  1
    END
    BEGIN MASS MATRIX
      0.25 0.00 0.00
      0.00 0.50 0.00
      0.00 0.00 0.025
    END
  END [SUPERELEMENT SECTION <string>section_name]
```

```
BEGIN STIFFNESS MATRIX
```

The `STIFFNESS MATRIX` command block defines the *NxN* stiffness matrix for the superelement. The number of rows and columns in the stiffness matrix must be the same as the number of rows in the `MAP` command block. The stiffness matrix should be symmetric. If the input matrix is not symmetric, it will be made symmetric by $K_{sym} = 0.5 * (K_{input} + K_{input}^T)$. To guarantee stability for explicit dynamics and solution convergence for implicit statics/dynamics, the stiffness matrix should be positive definite.

```
BEGIN DAMPING MATRIX
```

The `DAMPING MATRIX` command block defines the *NxN* damping matrix for the superelement. The number of rows and columns in the damping matrix must be the same as the number of rows in the `MAP` command block. The damping matrix should generally be symmetric. This command block is optional. If a damping matrix is not defined, no damping will be used by default.

```
BEGIN MASS MATRIX
```

The `MASS MATRIX` command block defines the *NxN* mass matrix for the superelement. The mass matrix must have the same dimensions as the stiffness matrix. The mass matrix need not be symmetric, however to guarantee stability for explicit dynamics and solution convergence for implicit statics/dynamics the mass matrix should be positive definite.

```
FILE = <string>netcdf_file_name
```

As an alternative to defining stiffness and mass matrices in the input file, the stiffness and mass matrices may be imported from a NetCDF file. External codes (such as Salinas) are able to compress larger models into superelements and output the matrices in the NetCDF format. The `netcdf_file_name` defines the path to the file that contains the mass, damping, and stiffness matrix definitions. All superelement matrices should be defined either in the input deck or in the NetCDF file. The damping matrix is optional, so if it is not defined in the NetCDF file, no damping will be used by default. The connectivity map needs to be specified in the input file in either case.

In the netcdf file the stiffness matrix must be named `Kr`, the mass matrix `Mr`, and the damping matrix `Cr`.

Known Issue: Superelements are not compatible with several modeling capabilities. They cannot be used with element death. They cannot be used with node-based, power method, or Lanczos critical time step estimation methods. They are also not compatible with some preconditioners (such as FETI) for implicit solutions.

6.2.13 Peridynamics Section

```
BEGIN PERIDYNAMICS SECTION <string>section_name
  MATERIAL MODEL FORMULATION = <string>CLASSICAL|PERIDYNAMICS
  HORIZON = <real>horizon [SCALE BY ELEMENT RADIUS]
  INFLUENCE FUNCTION = <string>influence_function
  BOND DAMAGE MODEL = <string>CRITICAL_STRETCH|NONE(NONE)
     [<real>critical_stretch]
  HOURGLASS STIFFNESS = <real>stiffness(0.0)
  BOND CUTTING BLOCK = <string list>block_names
  BOND VISUALIZATION = <string>OFF|ON(OFF)
END [PERIDYNAMICS SECTION <string>section_name]
```

Peridynamics is an extension of classical solid mechanics that allows for the modeling of bodies in which discontinuities occur spontaneously. For this reason, peridynamics is well suited for modeling phenomena such as cracking. This section describes the `PERIDYNAMICS SECTION` command block. Details regarding peridynamics methodology and suggested usage guidelines can be found in Section 10.2.3.

A `PERIDYNAMICS SECTION` should only be associated with elements with the `SPHERE` topology. Sphere elements are point elements that are connected to only a single node. Sphere elements may be created, for example, using the `sphgen3d` utility.

Warning: The default parallel decomposition for models containing sphere elements is typically very poor. It is recommended that an initial rebalance be carried out for parallel analyses with peridynamics. See Section 6.8 for details regarding the `INITIAL REBALANCE` command.

Within a `PERIDYNAMICS SECTION`, the `MATERIAL MODEL FORMULATION` command line specifies the application of either a peridynamics material model or a classical material model adapted for use with peridynamics as described in Section 10.2.3. Currently, only the `ELASTIC` material model is available with the `PERIDYNAMICS` material model formulation. Setting `MATERIAL MODEL FORMULATION = CLASSICAL` enables the interface between Sierra/SM peridynamics and the `LAME` material library. This interface allows the material models described in Section 5.2 to be used with peridynamics.

The peridynamic horizon is set with the `HORIZON` line command. As described in Section 10.2.3, the horizon determines the region of nonlocality for a given peridynamic element. A peridynamic element interacts directly with all peridynamic elements that are within a distance equal to the horizon in the reference configuration. The horizon may be specified as either a constant value or

as a multiple of the element radius. By default, the command applies a constant horizon equal to the specified value of horizon to all elements associated with the given peridynamics section. If the optional SCALE BY ELEMENT RADIUS is included at the end of this command, the horizon for each peridynamic element is set to horizon times a given element's radius.

A user-defined peridynamic influence function may be defined using the INFLUENCE FUNCTION command line. The influence function provides a means for controlling the relative influence of a neighbor within the family of a given peridynamic element as a function of the neighbor's distance from the element. Distance from the element is normalized by the horizon and ranges from 0.0 to 1.0. A value of 0.0 corresponds to a distance of zero from the element, and a value of 1.0 corresponds to a distance from the element equal to the horizon. By default, the influence function is set to a constant value of 1.0, which results in all neighbors being weighted equally. An arbitrary influence function may be defined with the INFLUENCE FUNCTION command line by providing a function name corresponding to a DEFINITION OF FUNCTION command block in the Sierra scope (see Section 2.1.5). The input to this function is the distance in the reference configuration from an element to a given neighbor, divided by that element's horizon. The output is the weighting value, ω.

The bond damage model is specified by the BOND DAMAGE MODEL command. The available options for BOND DAMAGE MODEL are CRITICAL_STRETCH, ELEMENT_VARIABLE, and NONE. The default option, NONE, disables bond breaking. The critical stretch bond breaking law is activated by specifying CRITICAL_STRETCH. If this model is selected, the user must specify critical_stretch, the value of stretch at which bonds are broken, where stretch is computed as bond elongation divided by the initial bond length. The ELEMENT_VARIABLE option allows bonds to be broken based on the maximum value of a specified element variable (e.g., an element variable associated with a material model, such as equivalent plastic strain). In the case of a bond that connects two peridynamics elements for which the specified element variable is defined, the average of the two element values is compared against the threshold value to determine bond breakage. If the specified element variable is defined on only one of the two elements connected by a given bond, the single available value is compared against the threshold value.

Hourglass stiffness for peridynamics is controlled with the HOURGLASS STIFFNESS command. Within the peridynamics hourglass stiffness routine, the parameter stiffness is multiplied by a material's bulk modulus to determine the resistance to deformation in potential zero-energy modes. By default, no hourglass stiffness is computed for peridynamics; hourglass stiffness must be activated by including a HOURGLASS STIFFNESS command line. Note that peridynamics hourglass control is completely independent from the hourglass control applied to underintegrated elements (Section 6.1.5.6).

By default, a peridynamic bond is formed between a given peridynamic element and all peridynamic elements within its horizon. In some cases, this results in the formation of bonds that are nonphysical. For example, bonds may be incorrectly formed between two components that are within a distance equal to the horizon in the reference configuration but that are not physically connected. The bond cutting feature provides a mechanism for controlling the formation of bonds. The BOND CUTTING BLOCK command designates blocks in the block_names list as bond cutting blocks. Any bond that would pass through a surface of a bond cutting block is omitted from the analysis. For example, to prevent the creation of bonds between two components that are in

close proximity but are not physically connected, a user may include a block in the input mesh file that lies between these components and designate that block as a bond cutting block. The bond cutting block may be any type of block (e.g., a shell, hex, or tet mesh), provided that Sierra/SM is capable of skinning it (see Section 8.2). If a sphere element mesh for use with peridynamics has been created from a hex-element mesh (for example, with `sphgen3d`), it may be convenient to simply include the original hex element mesh as a bond cutting block. Any block designated as a bond cutting block is automatically removed from the analysis following the initialization of bonds.

The `BOND VISUALIZATION` command provides a mechanism for visualizing bonds through the representation of bonds as bar elements in the output database. This feature is set to `OFF` by default and may be activated with the command `BOND VISUALIZATION = ON`. The bond visualization feature is intended as a verification tool for mesh creation and is not recommended for used for production runs.

Warning: Setting `BOND VISUALIZATION = ON` results in the creation of additional elements in the output file. Because each bond is represented by an additional element, the resulting output file may grow in size by several orders of magnitude relative to the standard output file. For this reason, `BOND VISUALIZATION` is recommended only as a mesh verification tool for small- to medium-sized problems and should not be used for production runs.

Known Issue: Peridynamics is not compatible with several modeling capabilities. Peridynamics is currently not compatible with restart. Also, the only type of rebalancing that can be done with peridynamics is an initial rebalance.

6.3 Element-like Functionality

This section describes functionality in Sierra/SM that behaves similarly to finite elements, although is not implemented as such. The functionality described in this section—rigid bodies and the torsional spring mechanism (explicit only)—is specified through command blocks that appear in the SIERRA scope and region scope, respectively.

6.3.1 Rigid Body

```
BEGIN RIGID BODY <string>rb_name
  MASS = <real>mass
  POINT MASS = <real>mass [AT <real>X <real>Y <real>Z]
  REFERENCE LOCATION = <real>X <real>Y <real>Z
  INERTIA = <real>Ixx <real>Iyy <real>Izz <real>Ixy
    <real>Iyz <real>Izx
  POINT INERTIA = <real>Ixx <real>Iyy <real>Izz <real>Ixy
    <real>Iyz <real>Izx
  MAGNITUDE = <real>magnitude_of_velocity
  DIRECTION = <string>direction_definition
  ANGULAR VELOCITY = <real>omega
  CYLINDRICAL AXIS = <string>axis_definition
  INCLUDE NODES IN <string>surface_name
    [IF <string>field_name <|<=|=|>=|> <real>value]
END [RIGID BODY <string>rb_name]
```

A rigid body can consist of any combination of elements—solid elements, structural elements, and point masses—except SPH or peridynamics elements. All nodes associated with a rigid body maintain their relative position to each other as determined at time 0 when there is no deformation of the body. This means that the elements associated with the rigid body can translate and rotate through space, but they cannot deform. Element blocks defining the rigid body do not have to be contiguous. They can also adjoin deformable element blocks. Multiple rigid bodies are allowed in a model.

The non-deformable nature of the rigid body can put it in conflict with other constraints. Refer to Appendix E for information on how conflicting constraints are handled.

Warning: Kinematic boundary conditions prescribed on a rigid body must be applied to the rigid body reference node. Kinematic boundary conditions prescribed on a rigid body that are not prescribed on the rigid body reference node will be unenforced. To apply kinematic boundary conditions to a rigid body, use the `BLOCK =` or the `RIGID BODY =` line command with the rigid body's block id or name, respectively, in the appropriate boundary condition command block. Refer to Chapter 7.

Sierra/SM creates a new node for each rigid body in the analysis. The new nodes are true nodes in that they are associated with solution fields such as displacement, velocity, and rotational velocity. These nodes will appear in a results file along with other nodes. The global node number given to the new nodes is simply the total number of nodes in the mesh plus one, repeated for each new rigid body node.

If an element block is declared to be a part of a rigid body, the internal force calculations are not called for the elements in that block and element time steps for explicit dynamics are not computed. Hence, for a model where all the element blocks are included in rigid bodies, there is no estimate for a global explicit dynamics time step. In this case an initial time must be specified in the TIME CONTROL command block (Section 3.1.1) with the command line

Explicit
```
INITIAL TIME STEP = <real>time_step
```

For the completely rigid explicit dynamics case, one also should not override the default value of 1.0 for the time step scale factor (see Section 3.1.1). A time step value of 1.0×10^{-6} should work well for most such problems, though any time step may be specified. However since a completely rigid problem does not impose a stability limit on the explicit dynamics time step, specifying too large of a time step could cause loss of accuracy in the solution of the problem (due to numerical integration error, large rotations over a time step, etc.) even though the problem runs to completion.

Specification of a rigid body requires the above command block, which appears in the SIERRA scope, plus the RIGID BODY command line that appears in the various SECTION command blocks described in this chapter. Suppose, for example, rigidbody_1 consists of element blocks 100, 110, and 280. The PARAMETERS FOR BLOCK command blocks for element blocks 100, 110, and 280 must all contain a SECTION command line. In each case, the Section must contain a line such as:

```
RIGID BODY = rigidbody_1
```

Once you have declared an element block or some collection of element blocks to be a rigid body and created a rigid body name (through the Section chosen), that rigid body name must appear as the name in a RIGID BODY command block. In our example, we must have a RIGID BODY command block with the value for rb_name set to rigidbody_1. Therefore, at a minimum, you must have a command block in the SIERRA scope with the form

```
BEGIN RIGID BODY rigidbody_1
END RIGID BODY rigidbody_1
```

for our example.

The RIGID BODY command block has several optional command lines, composing four groups of commands. One group consists of the MASS, POINT MASS, REFERENCE LOCATION, POINT INERTIA, and INERTIA command lines, a second group consists of the paired MAGNITUDE and DIRECTION command lines, a third group consists of the paired ANGULAR VELOCITY and CYLINDRICAL AXIS command lines, and a final group consists of the INCLUDE NODES IN command line. The command block can include none or any combination of these groups. If none of these commands is included, the command block simply supplies the value for rb_name.

Input to the MASS command line consists of a single real number that defines the total mass of the rigid body. If this line command is not present, the mass of the rigid body will be computed using

the elements in the rigid body and their densities. This is the translational mass at the reference location. This command does not change the inertia terms.

The POINT MASS command line requires a real number specifying the amount of mass added to the rigid body. Optionally, the location of that mass may be specified with three real numbers. If the location is not given, the additional mass will be placed at the reference location (and will not affect the moments and products of inertia).

The REFERENCE LOCATION command line requires three real numbers defining the reference location for the rigid body. If this line command is not present, the center of mass will be used. The center of mass is calculated from the mesh and the element densities.

Input to the INERTIA command line consists of six real numbers. If present, this command line will set the inertia for the rigid body. If it is not present, moments and products of inertia are computed for the rigid body based on the reference location of the rigid body and the element masses.

Input to the POINT INERTIA command line consists of six real numbers that define moments (Ixx, Iyy, Izz) and products (Ixy, Iyx, Izx) of inertia to be added to the inertia tensor of the rigid body. This modified inertia tensor (rather than the inertia tensor based solely on element mass) is then used to calculate the motion of the rigid body.

It should be noted, if a rigid body contains an element that has rotational resistance, i.e. shell, beam, etc, the nodal moment of inertia of that element is added to the diagonal components of the rigid body's inertia tensor.

An initial, translational-only velocity for the rigid body reference location should be specified with the MAGNITUDE and DIRECTION command lines. The MAGNITUDE command line gives the magnitude of the initial velocity applied to the reference location, and the DIRECTION command line gives a defined direction.

An initial rotational and translational velocity for a rigid body can be specified with the ANGULAR VELOCITY and CYLINDRICAL AXIS command lines. The ANGULAR VELOCITY command line gives the initial translational velocity of and rotational velocity about the reference location due to an angular velocity about some defined axis given on the CYLINDRICAL AXIS command line. If the defined cylindrical axis passes through the reference location, this command will impart only an initial rotational velocity to the rigid body reference node. If the axis does not pass through the reference location, this command will impart an initial translational velocity on the rigid body node as well as the initial rotational velocity.

The INCLUDE NODES IN command line allows a rigid body to include nodes of a surface or block of the mesh. Optionally, the nodes in the surface or block will be included in the rigid body only if the value of a field on the nodes meets a given criterion. For example, consider the rigid body block below.

```
begin rigid body rigid1
  include nodes in block_1
  include nodes in surface_3 if height = -1.0
end
```

Rigid body rigid1 will include all nodes in block_1 and those nodes on surface_3 whose value of the `height` field are equal to -1.0. The optional field (`height` in this case) must be a field known in the analysis. This field may be read in from restart, be a native field initialized upon startup, or be an initialized user-defined field. Since the entire set of nodes in block_1 will be in the rigid body, internal forces and critical time steps (for explicit) will not be computed for elements in block_1.

Sierra/SM automatically outputs quantities such as displacement for the reference location of the rigid body. The name assigned to a rigid body will be used to construct variable names that give the quantities. This lets you identify the output associated with a rigid body based on the name you assigned for the rigid body.

Immediately before the results file is written, the accelerations for nodes associated with a rigid body are updated to reflect the accelerations due to the rigid-body constraints. This ensures that the accelerations sent to the results output are correct for a given time.

In summary, if you use a rigid body in an analysis, you will do one or more of the following steps:

- Include a `RIGID BODY` command block in the SIERRA scope. If desired, set reference location, mass, point mass, etc.
- Create a rigid body using one or more element blocks (except `PARTICLE` or `PERIDYNAMICS` element blocks). A `RIGID BODY` command line must appear in the `SECTION` command block used in the `PARAMETERS FOR BLOCK` command block for any element block associated with a rigid body.
- Include point mass element blocks with the rigid body if appropriate. To include point mass element blocks in a rigid body, a `RIGID BODY` command line must appear in the `SECTION` command block used in the `PARAMETERS FOR BLOCK` command block for those point mass element blocks.
- Associate an initial velocity or initial rotation about an axis with the rigid body, if appropriate. If any of the blocks associated with a rigid body have been given an initial velocity or initial rotation, the rigid body must have the same specification for the initial velocity or initial rotation.

The above steps involve a number of different command blocks. To demonstrate how to fully implement a rigid body, we will provide a specific example that exercises the various options available to a user.

Let us assume that we want to create a rigid body named `part_a` consisting of three element blocks. Two of the element blocks, element block 100 and element block 535, are eight-node hexahedra; one of the element blocks, element block 600, consists of only point masses. The `RIGID BODY` command block, `SECTION` command block, and the element blocks we want to associate with the rigid body will be as follows:

```
begin solid section hex_section
  rigid body = part_a
```

```
end
begin point mass section pm_section
  rigid body = part_a
  volume = 0.1
end

begin parameters for block block_100
  material steel
  solid mechanics model use elastic
  section = hex_section
end
begin parameters for block block_535
  material = aluminum
  solid mechanics model use elastic
  section = hex_section
end
begin parameters for block block_600
  material = mass_for_pointmass
  solid mechanics model use elastic
  section = pm_section
end
```

Suppose we want to have the rigid body initially rotating at 600 radians/sec about an axis parallel to the *x*-axis and passing through a point at (0, 10, 20). We would define this axis using the following set of `DEFINE` command lines:

```
define direction parallel_to_x with vector 2.0 0.0 0.0
define point off_axis with coordinates 0.0 10.0 20.0
define axis body_axis with point off_axis direction parallel_to_x
```

The initial angular velocity specification in the `RIGID BODY` command block is as follows:

```
begin rigid body part_a
  cylindrical axis = body_axis
  angular velocity = 600
end rigid body part_a
```

Sierra/SM automatically generates and outputs global data associated with the rigid body (e.g. displacement and quaternion). See Section 9.8 for details on global variable output. Individual field components may be output as discussed in Section 9.1.4 and Table 9.2. In general the field component `fieldi` of the rigid body named `part_a` will be written to the output file(s) with the name `FIELDI_PART_A`.

6.3.1.1 Multiple Rigid Bodies from a Single Block

Typically, all of the elements in a block are assigned to a single rigid body. However, it is sometimes necessary to define a very large number of rigid bodies. It would be unwieldy to create a

separate block for each rigid body. To avoid this, it is possible to create an element attribute in the input mesh file that contains an integer ID used to denote the rigid body in which each element should belong. The `RIGID BODIES FROM ATTRIBUTES` command is used to associate the attribute with the rigid body IDs. This command must be used in conjunction with the `RIGID BODY` command. These commands are used in the context of the section block. They are shown here in the `SOLID SECTION` block, but they can be used with any section type that supports rigid bodies.

```
BEGIN SOLID SECTION <string>sec_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id to
    <integer>last_id
  RIGID BODY = <string>rigid_body_name
END
```

If the `RIGID BODIES FROM ATTRIBUTES` command is used, a series of rigid bodies will be created. These rigid bodies are named using the specified `rigid_body_name`, followed by an underscore (_), and then by an integer ID, which ranges between the specified value of `first_id` and `last_id`. Elements having an attribute value equal to a given ID are added to the corresponding rigid body.

When referencing the rigid body name for to apply boundary conditions, it is important to specify the entire name of the rigid body, including the underscore and ID. For example, a name such as `part_a_1` would be used in a boundary condition.

6.3.2 Torsional Spring Mechanism

```
BEGIN TORSIONAL SPRING MECHANISM <string>spring_name
  NODE SETS = <string>nodelist_int1 <string>nodelist_int2
    <string>nodelist_int3 <string>nodelist_int4
  TORSIONAL STIFFNESS = <real>stiffness
  INITIAL TORQUE = <real>init_load
  PERIOD = <real>time_period
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [TORSIONAL SPRING MECHANISM <string>spring_name]
```

This feature was originally implemented to model a torsional spring wrapped around a fixed pin. One end of the pin is fixed to a base, and one end of the spring is attached to this base. There is an arm on the other end of the pin, and this arm can rotate around the pin. The second end of the spring is attached to this arm. The spring resists motion of the arm. Any similar mechanism can be modeled with the torsional spring. Although the torsional spring is element-like in its overall behavior, its implementation within the code structure is different from the other elements described in Chapter 6. The torsional spring does not make use of a section, and its command block (`TORSIONAL SPRING MECHANISM`) should appear in the region scope. A schematic for the torsional spring mechanism is shown in Figure 6.8.

The mechanism consists of two nodes that represent the axis of a torsional spring. Node 0 is at the base of the torsional spring, and node 1 is at the top of the torsional spring. A third node, reference

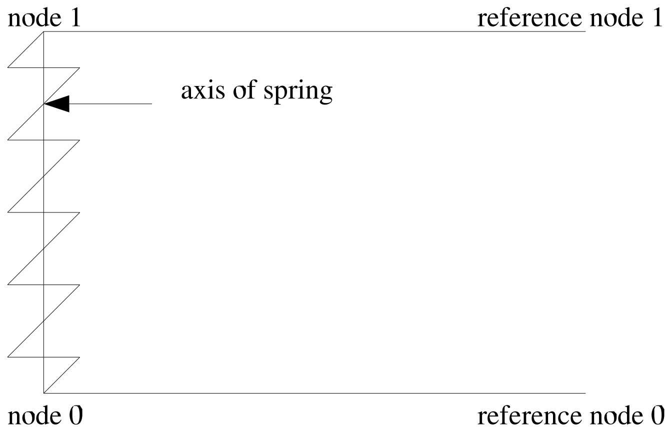


Figure 6.8: Schematic for torsional spring.

node 0, defines an arm extending from the axis of the torsional spring to some attachment point near the base of the spring. A fourth node, reference node 1, defines an arm extending from the axis of the torsional spring to some attachment point near the top of the spring. The rotation of the two arms relative to each other as measured along the axis of the torsional spring represents the angular deformation of the spring and determines the moment in the spring. The moment in the spring is translated into external forces at the two attachment points, reference node 0 and reference node 1.

In the `TORSIONAL SPRING MECHANISM` command block, the string `spring_name` is defined by the user. Via the `NODE SETS` command line, the mechanism is defined with four node sets, and each node set has a single node. The first set (`nodelist_int1`) defines node 0 in Figure 6.8; node 0 is the origin of a local coordinate system for the torsional spring mechanism. The second node set (`nodelist_int2`) defines node 1 in Figure 6.8; node 0 and node 1 define the axis of the torsional spring mechanism. The third node set (`nodelist_int3`) defines reference node 0; reference node 0 is an attachment point for the spring associated with node 0. The fourth node set (`node_int4`) defines reference node 1; reference node 1 is an attachment point for the spring associated with node 1.

The nodes defining the spring mechanism are used to set up a local coordinate system (x', y', z'). The (z'-axis runs along the axis of the spring from node 0 to node 1. The x'-axis extends from the axis of the spring and passes through reference node 0. If we are looking down the axis of the spring in the negative z'-direction, a positive rotation of the arm defined by node 1 and reference node 1 is in the counterclockwise direction. This is shown in Figure 6.9.

The torque, T, in the spring is simply

$$T = K\theta, \tag{6.5}$$

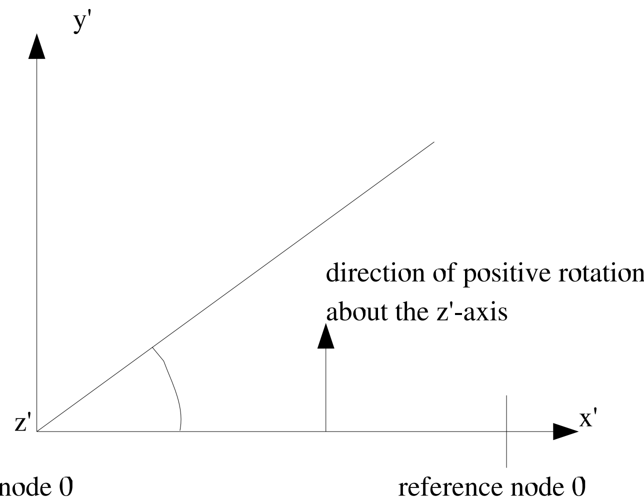


Figure 6.9: Positive direction of rotation for torsional spring.

where K is the torsional stiffness and θ is the rotation of the top arm relative to the bottom arm as measured along the axis of the spring. In the `TORSIONAL STIFFNESS` command line, K is specified by the real value `stiffness`.

The torque in the spring is converted to external forces with components in the global coordinate system XYZ. These external forces depend on the torque and the length of the spring arms. The length of the spring arms is automatically calculated.

You can apply an initial torque with the real value `init_load` in the `INITIAL TORQUE` command line. The maximum value of the torque is reached in the time specified by the real value `time_period` in the `PERIOD` command line.

The initial torque is applied over some period by specifying the angular rate of deformation in the torsional spring, $\dot{\theta}$, over some period p. At some given time t, the angular rate of deformation is

$$\dot{\theta} = \frac{ap}{2} \left[1 - \cos\left(\pi t / p \right) \right], \tag{6.6}$$

where

$$a = \frac{2T_i}{Kp}. \tag{6.7}$$

In Equation (6.7), T_i is the initial torque. Over the period p, the total strain increment generates the desired initial load in the truss.

During the initial load period, the time increments should be reasonably small so that the integration of $\dot{\theta}$ over the period is accurate. The period should be set long enough so that if the model was held in a steady state after time p, there would be only a small amount of oscillation in the load in the torsional spring.

When doing an analysis, you may not want to activate certain boundary conditions until after the prestressing is done. During the prestressing, time-independent boundary conditions such as fixed displacement will most likely be turned on. Time-dependent boundary conditions such as prescribed acceleration or prescribed force will most likely be activated after the prestressing is complete.

You can output the torque in the spring, the total rotation, and the last angle between the arms. The name specified on the command block is used to construct parameters for the mechanism. Suppose the input line is:

```
begin torsional spring mechanism lower_spring
```

where `lower_spring` is a user-specified name. The code will automatically generate the parameters `TS_lower_spring_MOMENT`, `TS_lower_spring_ROTATION`, and `TS_lower_spring_LAST_ANGLE`. These variables can then be output in a results file. For example, one could use

```
global TS_lower_spring_MOMENT as ts_lspring_m
global TS_lower_spring_ROTATION as ts_lspring_r
global TS_lower_spring_LAST_ANGLE as ts_lspring_la
```

in the `RESULTS OUTPUT` command block. If several torsional spring mechanisms appear in one model, you can generate unique names to keep track of the parameters associated with each spring. See Section 9.2 for further information about results output.

The `ACTIVE PERIODS` and `INACTIVE PERIODS` command lines determine when the torsional spring is active. See Section 2.5 for more information about these command lines. Although the active periods option is available in the `TORSIONAL SPRING` command block, use of this option to turn the torsional spring off and on repeatedly is not recommended. Turning the torsional spring off and on repeatedly may lead to erroneous behavior in the spring model.

6.4 Mass Property Calculations

```
BEGIN MASS PROPERTIES
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  # structure command
  STRUCTURE NAME = <string>structure_name
END [MASS PROPERTIES]
```

Sierra/SM automatically gives mass property information for the total model, which consists of all the element blocks. (The mass for the total model, for example, is the total mass of all the element blocks.) Sierra/SM also automatically gives mass property information for each element block.

In addition to the mass property information that is generated, Sierra/SM gives you the option of defining a structure that represents some combination of element blocks and then of calculating the mass properties for this particular structure. If you wish to define a structure that is a combination of some group of element blocks, you must use the `MASS PROPERTIES` command block. This command block appears in the region scope.

For the total model, each element block, and any user-defined structure, Sierra/SM reports the mass and the center of mass in the global coordinate system. It also reports the moments and products of inertia, as computed in the global coordinate system about the center of mass.

The `MASS PROPERTIES` command block contains two groups of commands—block set and structure. Each of these groups is basically independent of the other. Following are descriptions of the two command groups.

6.4.1 Block Set Commands

The `block set commands` portion of the `MASS PROPERTIES` command block defines a set of blocks for which mass properties are being requested, and can include some combination of the following command lines:

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of blocks. See Section 7.1.1 for more information about the use of these command lines for creating a set of blocks used by the command block. There must be at least one `BLOCK` or `INCLUDE ALL BLOCKS` command line in the command block.

The REMOVE BLOCK command line allows you to delete blocks from the set specified in the BLOCK and/or INCLUDE ALL BLOCKS command line(s) through the string list block_names. Typically, you would use the REMOVE BLOCK command line with the INCLUDE ALL BLOCKS command line. If you want to include all but a few of the element blocks, a combination of the REMOVE BLOCK command line and INCLUDE ALL BLOCKS should minimize input information.

Suppose that only one element block, block_300, is specified on the BLOCK command line. Then only the mass properties for that block will be calculated. If several element blocks are specified on the BLOCK command line, then that collection of blocks will be treated as one entity, and the mass properties for that single entity will be calculated. For example, if two element blocks, block_150 and block_210, are specified on the BLOCK command line, the total mass for the two element blocks will be reported as the total mass property.

6.4.2 Structure Command

The output for the mass properties will be identified by the command line:

```
STRUCTURE NAME = <string>structure_name
```

where the string structure_name is a user-defined name for the structure.

6.5 Element Death

```
              BEGIN ELEMENT DEATH <string>death_name
                #
                # block set commands
                BLOCK = <string list>block_names
                INCLUDE ALL BLOCKS
                REMOVE BLOCK = <string list>block_names
                #
                # criterion commands
Explicit        CRITERION IS AVG|MAX|MIN NODAL VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance
Explicit        CRITERION IS ELEMENT VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance [KILL WHEN
                  <integer>num_intg INTEGRATION POINTS REMAIN]
Explicit        CRITERION IS GLOBAL VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance
Explicit        CRITERION IS ALWAYS TRUE
                MATERIAL CRITERION
                  = <string list>material_model_names [KILL WHEN
                  <integer>num_intg INTEGRATION POINTS REMAIN]
                #
                # subroutine commands
Explicit        ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
Explicit        SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
Explicit        SUBROUTINE REAL PARAMETER: <string>param_name
                  = <real>param_value
Explicit        SUBROUTINE INTEGER PARAMETER: <string>param_name
                  = <integer>param_value
Explicit        SUBROUTINE STRING PARAMETER: <string>param_name
                  = <string>param_value
                #
                # evaluation commands
Explicit        CHECK STEP INTERVAL = <integer>num_steps
Explicit        CHECK TIME INTERVAL = <real>delta_t
Explicit        DEATH START TIME = <real>time
                #
                # miscellaneous option commands
                SUMMARY OUTPUT STEP INTERVAL = <integer>output_step_interval
                SUMMARY OUTPUT TIME INTERVAL = <real>output_time_interval
Explicit        DEATH ON INVERSION = OFF|ON(OFF)
Explicit        DEATH ON ILL DEFINED CONTACT = OFF|ON(OFF)
Explicit        DEATH STEPS = <integer>death_steps(1)
Explicit        FORCE VALID ACME CONNECTIVITY
Explicit        AGGRESSIVE CONTACT CLEANUP = <string>OFF|ON(OFF)
Explicit        DEATH METHOD = <string>DEACTIVATE ELEMENT|DEACTIVATE NODAL MPCS|
                DISCONNECT ELEMENT|INSERT COHESIVE ZONES(DEACTIVATE ELEMENT)
```

```
    #
    # death on proximity commands
    BEGIN DEATH PROXIMITY
      BLOCK = <string list>block_names
      NODE SET = <string list>nodelist_names
      SURFACE = <string list>surface_names
      REMOVE BLOCK = <string list>block_names
      REMOVE NODE SET = <string list>nodelist_names
      REMOVE SURFACE = <string list>surface_names
      TOLERANCE = <real>search_tolerance
    END [DEATH PROXIMITY]
    #
    # particle conversion commands
    BEGIN PARTICLE CONVERSION
      PARTICLE SECTION = <string>section_name
      NEW MATERIAL = <string>material_name
        MODEL NAME = <string>model_name
      TRANSFER = <string>NONE|ALL(ALL)
      MAX NUM PARTICLES = <integer>num(1)
      MIN PARTICLE CONVERSION VOLUME = <real>vol
    END [PARTICLE CONVERSION]
    ACTIVE PERIODS = <string list>period_names
    INACTIVE PERIODS = <string list>period_names
    #
    # cohesive zone setup commands
    COHESIVE SECTION = <string>sect_name
    COHESIVE MATERIAL = <string>mat_name
    COHESIVE MODEL = <string>model_name
    COHESIVE ZONE INITIALIZATION METHOD = <string>NONE|
      ELEMENT STRESS AVG(NONE)
  END [ELEMENT DEATH <string>death_name]
```

Explicit (BEGIN DEATH PROXIMITY)

Explicit (BEGIN PARTICLE CONVERSION)

The ELEMENT DEATH command block is used to remove elements from an analysis. For example, the command block can be used to remove elements that have fractured, that are no longer important to the analysis results, or that are nearing inversion (explicit only). This command block is located within the region scope. The name of the command block, death_name, is user-defined and can be referenced in other commands to update boundary or contact conditions based on the death of elements creating new exposed surfaces.

Any element in an element block or element blocks selected in the ELEMENT DEATH command block is removed (killed) when one of the criteria specified in the ELEMENT DEATH command block is satisfied by that element. When an element dies, it is removed permanently. Any number of ELEMENT DEATH command blocks may exist within a region.

When an element is killed, the contribution of that element's mass to the attached nodal mass is removed from the attached nodes. If all of the elements attached to a node are killed, the mass for the node and all associated nodal quantities will be set to zero. If all of the elements in a region are killed, the analysis will terminate.

In implicit calculations, elements may be killed based off a global variable or a material criterion, and this capability must be used in conjunction with control failure in a multilevel solver block. Control failure is described in Section 4.7. Element death will not be activated unless control failure is specified.

In explicit calculations, elements may be killed based off of any internal variable, derived variable, material state variable, or user defined variable. Note that user-defined variables for triggering element death should be used with care. There are timing and parallel issues regarding the use of a user-defined variable with element death. User subroutines may work for element death in some situations where one might want to reference a user variable.

The quantity used in an element death criterion statement should have only a single value at each integration point. For example, the following command:

Explicit
```
criterion is element value of stress(xx) > 10.0
```

is unambiguous and valid. However this command:

Explicit
```
criterion is element value of stress > 10.0
```

is ambiguous, and will result in an error. Stress is a 3x3 tensor and there is no rational way to compare that tensor to the scalar value 10. If a variable is defined on multiple integration points, the criterion will be triggered if the value is exceeded at any of the integration points. If the command:

Explicit
```
criterion is element value of stress(xx,:) > 10.0
```

were used with shells with five integration points, the shells would be killed if the value of `stress(xx)` at any of the five integration points exceeded 10.

The `ELEMENT DEATH` command block contains five groups of commands—block set, criteria, evaluation, miscellaneous, and cohesive zone setup. The command block must contain commands from the block set and criteria groups. Command lines from the evaluation and miscellaneous groups are optional, as are the cohesive zone commands.

Following are descriptions of the different command groups, an example of using the `ELEMENT DEATH` command block, and some concluding remarks related to element death visualization.

6.5.1 Block Set Commands

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK = <string list>block_names
```

The `block set commands` portion of the `ELEMENT DEATH` command block defines a set of blocks for selecting the elements to be referenced. These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of blocks, as described in Section 7.1.1.

Element death must apply to a group of elements. There are two commands for selecting the elements to be referenced: BLOCK and INCLUDE ALL BLOCKS. In the BLOCK command line, you can list a series of blocks through the string list block_names. This command line may also be repeated multiple times. The INCLUDE ALL BLOCKS command line adds all the element blocks present in the region to the current element death definition. There must be at least one BLOCK or INCLUDE ALL BLOCKS command line in the ELEMENT DEATH command block.

The REMOVE BLOCK command line allows you to delete blocks from the set specified in the BLOCK and/or INCLUDE ALL BLOCKS command line(s) through the string list block_names. Typically, you will use the REMOVE BLOCK command line with the INCLUDE ALL BLOCKS command line. If you want to include all but a few of the element blocks, a combination of the REMOVE BLOCK command line and INCLUDE ALL BLOCKS command line should minimize input information.

6.5.2 Criterion Commands

Any combination of death criteria (CRITERION IS NODAL, CRITERION IS ELEMENT, CRITERION IS GLOBAL, CRITERION IS ALWAYS TRUE, ELEMENT BLOCK SUBROUTINE, MATERIAL CRITERION as appropriate in explicit or implicit) can be specified within a single ELEMENT DEATH command block. However, only one user subroutine criterion (available for explicit only) may appear in a set of criteria command lines. If multiple death criteria are specified for a given element, it will be killed when the first of those criteria is met. In other words, element death with multiple criteria is an OR condition rather than an AND condition.

An explicit dynamics simulation with a material that has both a tension cutoff and a compression cutoff would be an example for which you would want to use element death with multiple criteria. You would use one criterion to set failure in tension and another criterion to set failure in compression. If either the tension cutoff value, as set by the tension criterion, or the compression cutoff value, as set by the compression criterion, was exceeded for an element, the element would be killed.

As another explicit dynamics example, you might have a problem that uses a nodal criterion and a user subroutine criterion. For this second example, you are precluded from using another subroutine criterion because you already have one. You could add some combination of nodal, element, global, or material criteria to the existing nodal and subroutine criteria, but you could not add another subroutine criterion.

6.5.2.1 Nodal Variable Death Criterion

```
CRITERION IS AVG|MAX|MIN NODAL VALUE OF
  <string>var_name <|<=|=|>=|> <real>tolerance
```

Any nodal variable may be used by an element death criterion. The input parameters are described as follows:

- Nodal variables are present on the nodes of an element, and these nodal values must be reduced to a single element value for use by the criterion. The available types of reduction

are AVG, which takes the average of the nodal values; MAX, which takes the maximum of the nodal values; and MIN, which takes the minimum of the nodal values.

- The string var_name gives the name of the variable. See Section 9.9 for a listing of the variables. Parenthesis syntax may be used in the variable name for selection specific variable components or integration points, see Section 9.1.

- The specified variable, with an optional component specification if the variable has components, may be compared to a given tolerance with one of five operators. The operator is specified with the appropriate symbol for less than (<) less than or equal to (<=), equal to (=), greater than or equal to (>=), or greater than (>) The given tolerance is specified with the real value tolerance.

6.5.2.2 Element Variable Death Criterion

```
CRITERION IS ELEMENT VALUE OF
  <string>var_name[(<integer>component_num)]
  <|<=|=|>=|> <real>tolerance |
  <string>derived_quantity[(<integer>int_num)]
  <|<=|=|>=|> <real>tolerance [KILL WHEN <integer>num_intg
  INTEGRATION POINTS REMAIN]
```

Any element variable may be used by an element death criterion. An element variable is present on the element itself, so no reduction is required, which is why the first line in the format of the above command line differs from that of the nodal criterion command line in Section 6.5.2.1.

For a criterion using an element variable, the variable name, component or integration point number, and tolerance can be specified in the same manner as defined for the nodal criterion command line.

Element state variables can also be used as criteria for element death. The syntax used to specify the variable name differs if a specific integration point is desired.

- For LAME solid material models, the variable name is called out directly by name.

- For shell material models, the variable name is called out directly by name. By default the element will be killed if the criteria is met at any integration point. Parenthesis syntax (See Section 9.1) may be used to kill based on the state of a specified integration point. For example:

  ```
  criterion = element value of crack_opening_strain(2) > 0.01
  ```

  would use the value of the state variable crack_opening_strain at the second shell integration point through the shell thickness as the criterion for element death.

Refer to Section 9.9.2 for tables with listings of state variables for the various material models.

Derived quantities can also be used for element death. For example the following command:

```
criterion = element value of von_mises > 1.0e+05
```

will cause elements to be killed any of the integration points reach a critical value of von_mises. Alternatively, the command:

```
criterion = element value of von_mises(3) > 1.0e+05
```

is used to kill an element if a specific integration point reaches the critical value of von_mises.

For elements with multiple integration points, if an element criterion is used with an integration point variable, the integration points are killed individually as the criterion is exceeded on each integration point. The integration points are killed by linearly tapering down all components of the stress over the specified number of death steps. See Section 6.5.4.4 for more information on death steps.

The optional KILL WHEN num_intg INTEGRATION POINTS REMAIN portion of the command line is used to kill the entire element when enough integration points have been killed. This occurs when the number of remaining points drops down to the specified num_intg. For example, for shells with five integration points, the following command:

```
criterion is element value of stress(xx,:) > 10.0 kill when 2 \#
  integration points remain
```

will check the xx component of stress at each integration point. When the first three integration points reach that criterion, their stresses are linearly tapered down to zero over the specified number of death steps. Once the contributions from each of those three integration points have completely decayed to zero, leaving two remaining integration points, the element is killed.

6.5.2.3 Global Death Criterion

```
CRITERION IS GLOBAL VALUE OF
<string>var_name[(<integer>component_num)]
<|<=|=|>=|> <real>tolerance
```

Any global variable may be used in an element death criterion. Once the global criterion is reached, all elements specified in the ELEMENT DEATH command block are killed. The variable name, component number, and tolerance can be specified in the same manner as defined for the nodal or element criterion command line.

The input parameters are described as follows:

- The string var_name gives the name of the global variable. See Section 9.9 for a listing of the global variables.

- Parenthesis syntax may be used in the variable name to specify specific components of the variable. See Section 9.1 for more information.

- The specified variable, with an optional component specification if the variable has components, may be compared to a given tolerance with one of five operators. The operator is

specified with the appropriate symbol for less than (<), less than or equal to (<=), equal to (=), greater than or equal to (>=), or greater than (>). The given tolerance is specified with the real value `tolerance`.

6.5.2.4 Always Death Criterion

```
CRITERION IS ALWAYS TRUE
```

This command forces the criterion that determines whether to kill elements specified in the current death block to always be true. In other words, it causes all of those elements to immediately be killed. This command can be used to instantaneously kill a set of elements or convert that set of elements to particles. Converting a set of elements to particles in this manner at the beginning of an analysis can be an alternative to generating a mesh with particle elements.

This command is often used together with the `DEATH START TIME` command (see Section 6.5.3) to cause a set of elements to be killed or converted to particles at a certain time. This can be used as an alternative to block deactivation using the `ACTIVE FOR PROCEDURE` command (see Section 6.1.5.9).

6.5.2.5 Material Death Criterion

```
MATERIAL CRITERION = <string list>material_model_names [KILL WHEN
  <integer>num_intg INTEGRATION POINTS REMAIN]
```

Some material models have a failure criterion. When this failure criterion is satisfied within an element, the element has fractured or disintegrated. The material models reduce the stress in these fractured or disintegrated elements to zero. The `MATERIAL CRITERION` command line can be used to remove these fractured or disintegrated elements from an analysis. Removal of the fractured elements can speed computations, enhance visualization, and prevent spurious inversion of these elements that may stop the analysis.

The material models currently supported for use with the `MATERIAL CRITERION` command line are:

- `ELASTIC_FRACTURE` (Solid only, see Section 5.2.4)
- `DUCTILE_FRACTURE` (Solid only, see Section 5.2.7)
- `ML_EP_FAIL` (Solid and Shell, see Section 5.2.9)

Element death will kill an element based on a material criterion when the material model indicates that the element is failed and can carry no more load. For a single integration point element, this occurs when the one integration point in the element fails. For elements with multiple integration points, the default behavior is for the element to be killed when all but one of the integration points has failed. This behavior is the default because for multi-integration point elements, particularly

shells, if there is only a single integration point left, the element is severely under-integrated. The final integration point will generally not attract more load and will never fail. This default behavior can be changed by using the optional KILL WHEN num_intg INTEGRATION POINTS REMAIN command. In this command, num_intg specifies the number of remaining integration points when the element is to be killed.

Suppose you have an element block named part1_ss304 that references a material named SS304. This material, SS304, uses the DUCTILE_FRACTURE material model (see Section 5.2.7). You also have an element block named ring5_al6061 that references a material named al6061. This material, al6061, uses the ML_EP_FAIL material model (see Section 5.2.9). If you have an ELEMENT DEATH command block with the command line:

```
BLOCK = part1_ss304 ring5_al6061
```

and the command line:

```
MATERIAL CRITERION = DUCTILE_FRACTURE ML_EP_FAIL
```

then any element in part1_ss304 that fails according to the material model DUCTILE_FRACTURE (in material SS304) and any element in ring5_al6061 that fails according to the material model ML_EP_FAIL (in material al6061) will be killed by element death.

6.5.2.6 Subroutine Death Criterion

```
ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

A death criterion can be specified via a user-defined subroutine (see Chapter 11), which is invoked by the ELEMENT BLOCK SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user. The user-defined subroutine for element death must be an element subroutine signature (see Section 11.2.2). The element subroutine will return an output values array and a flag array of one flag per element (see Table 11.2 in Chapter 11). The output values array is ignored. Death is determined by the flag return value. For user-defined subroutines, a flag return value of –1 indicates that the element should die. A flag return value greater than or equal to 0 indicates that the element should remain alive.

Following the ELEMENT BLOCK SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Section 11.

6.5.3 Evaluation Commands

```
CHECK STEP INTERVAL = <integer>num_steps
CHECK TIME INTERVAL = <real>delta_t
DEATH START TIME = <real>time
```

Evaluation of element death criteria may be time consuming. Additionally, reconstruction of contact or other boundary conditions after element death can be very time consuming. For these reasons, three command lines are available for determining the frequency at which element death is evaluated. The default is to evaluate element death at every time step. You can limit the number of times at which the element death evaluation is done by using the following commands.

- The `CHECK STEP INTERVAL` command line instructs element death to evaluate the element death criteria only every `num_steps` time steps.
- The `CHECK TIME INTERVAL` command line instructs element death to evaluate the element death criteria only every `delta_t` time units.
- The `DEATH START TIME` command line instructs element death not to evaluate death criteria before a user-specified time, as given by the real value `time`.

You may use both the `CHECK STEP INTERVAL` and `CHECK TIME INTERVAL` command lines in a command block. Evaluations for element death will be made at both the time and step intervals if both of these command lines are included.

All three of the above command lines—`CHECK STEP INTERVAL`, `CHECK TIME INTERVAL`, and `DEATH START TIME`—are optional command lines.

6.5.4 Miscellaneous Option Commands

The command lines listed below need not be present in the `ELEMENT DEATH` command block unless the conditions addressed by each call for their inclusion.

6.5.4.1 Summary Output Commands

At the end of a run, a summary of all element death blocks is output to the log file. The `SUMMARY OUTPUT STEP INTERVAL` or `SUMMARY OUTPUT STEP INTERVAL` commands can be used to request that the summary be output to the log file at regular intervals during the run. The `SUMMARY OUTPUT TIME INTERVAL` command results in the summary being printed every `output_step_interval` steps, while the `SUMMARY OUTPUT TIME INTERVAL` results in the summary being printed once every `output_time_interval` time units. These two commands can be supplied individually, together or not at all. If neither are used, the summary is printed only at the end of execution.

It should be noted that this command applies to all element death blocks. If these commands appear in multiple element death blocks, the values specified in the last instance of each of these commands prevails.

6.5.4.2 Death on Inversion

```
DEATH ON INVERSION = ON|OFF(OFF)
```

If the DEATH ON INVERSION command line is set to ON, any element that inverts will be killed. This command currently only works for uniform gradient 8-noded hex elements 4, 8, and 10 noded tet elements, and 4 noded shells.

6.5.4.3 Death on Ill-defined Contact

```
DEATH ON ILL DEFINED CONTACT = ON|OFF(OFF)
```

If the DEATH ON ILL DEFINED CONTACT command line is set to ON, any element that would create invalid geometry for contact will be killed. This currently only works for uniform gradient 8 node hex elements and 4 node tet elements. An example of an element that creates an invalid contact geometry is an element that is partially inverted in such a way that some regions of the element have negative volume and exterior faces of the element point in the wrong directions.

A highly deformed element can have a positive total volume, permitting internal force to still be calculated, but have inverted local regions that make it invalid for contact. For this reason, killing elements due to ill-defined contact can be important to obtain solutions for contact problems in which elements are experiencing large distortions. Using death on inversion alone is not sufficient for those cases.

6.5.4.4 Death Steps

```
DEATH STEPS = <integer>death_steps(1)
```

If the DEATH STEPS command line is used and the value for death_steps is set to a value greater than 1, all components of the stress and the hourglass forces in a killed element are linearly scaled down until they reach 0 over the specified number of steps. Gradually releasing the energy due to killing an element over a number of steps can avoid instabilities that might occur if that element were suddenly killed. If this is done over an excessively large number of steps, however, the killed element may participate in the analysis long after it is relevant, and may cause problems if it undergoes excessive deformation or inverts while its stresses are being tapered off. The default number of steps, as provided in the integer value death_steps, is 1.

The value you select for death_steps will depend on your analysis. A small number such as 3 or 5 may be sufficient to prevent instabilities for most cases. However, in some cases it may be necessary to use a value for death_steps of 10 or larger. The loads, material models, and model complexity in your analysis will impact the value of death_steps.

If an element death criterion is used in an element with multiple integration points, the stresses for the integration points that have exceeded the criterion are decayed individually over the specified number of death steps. When the required number of death steps for an integration point has been exceeded, that integration point no longer contributes to the element internal force, and is considered dead. When the required number of integration points to kill the entire element (based on the user-input `KILL WHEN num_intg INTEGRATION POINTS REMAIN` – See Section 6.5.2.2) have reached that state, the process of killing the element is initiated, and the stresses in the remaining integration points and the hourglass forces are all scaled down over the specified number of death steps.

6.5.4.5 Degenerate Mesh Repair

```
FORCE VALID ACME CONNECTIVITY
```

If the `FORCE VALID ACME CONNECTIVITY` command line is present, degenerate mesh occurrences will be repaired. Element death has the possibility of creating degenerate mesh occurrences that will not be accepted by the ACME (see Reference 12) contact algorithms used by Presto. For example, if two continuum elements are connected only by an edge, ACME will not accept the mesh as a valid mesh. For this degenerate mesh occurrence (continuum elements connected only at an edge), the degeneracy is repaired by deleting all elements attached to the offending edge if we have turned on this repair option.

The option to repair degenerate mesh occurrences is on by default if there is a `CONTACT DEFINITION` command block that includes the command line:

See Section 8.2.6 for more information on how contact interacts with element death.

If you do not have a `CONTACT DEFINITION` command block and want to repair degenerate mesh occurrences for whatever purposes, you should include the `FORCE VALID ACME CONNECTIVITY` command line.

6.5.4.6 Aggressive Contact Cleanup

```
AGGRESSIVE CONTACT CLEANUP = <string>OFF|ON(OFF)
```

The `AGGRESSIVE CONTACT CLEANUP` command line enables the use of element death in an attempt to guarantee a valid contact mesh. Certain complex meshes with extensive element death can be problematic for the contact algorithms on some mesh decompositions. This command allows element death to query the contact algorithm and then remove any elements that the contact algorithm flags as having the potential to cause contact to fail.

6.5.4.7 Death Method

```
DEATH METHOD = <string>DEACTIVATE ELEMENT|DEACTIVATE NODAL MPCS|
  DISCONNECT ELEMENT|INSERT COHESIVE ZONES(DEACTIVATE ELEMENT)
```

The DEATH METHOD command specifies what happens when an element meets the death criterion. The following strings can be used as arguments to this command: DEACTIVATE ELEMENT, DEACTIVATE NODAL MPCS, DISCONNECT ELEMENT, and INSERT COHESIVE ZONES. The behavior controlled by these options is described below.

- With the default option, DEACTIVATE ELEMENT, the element is deactivated, effectively removing it from the mesh.
- The DEACTIVATE NODAL MPCS option removes the nodes of a killed element from any multi-point constraints. This only has an effect if the element has nodes that are in multi-point constraints. The multi-point constraint deactivation option can be used to break an element away from the mesh, allowing it to move independently. It can also be used to activate cohesive zones.
- The DISCONNECT ELEMENT option disconnects the element from the mesh, allowing the element to move independently. The disconnected element will no longer share any nodes with neighboring elements, and will only interact with the remainder of the mesh through contact.
- The INSERT COHESIVE ZONE option disconnects the element from the mesh and places a cohesive zone between the element and each face-adjacent neighbor. If this option is used, the commands COHESIVE SECTION, COHESIVE MATERIAL, and COHESIVE MODEL must be used to define the type of cohesive zone to be inserted, and are documented in Section 6.5.5.

When using element disconnection, it is necessary to use at least some hourglass stiffness and/or viscosity to prevent large deformations of the disconnected elements in zero energy modes. In addition, it is recommended that a secondary shape-based criterion be used in a separate element death command block to fully remove the disconnected elements if they begin to deform too much. The following example of an additional element death block to be used together with element disconnection shows recommended shape criteria:

```
begin element death
  include all blocks
  #Kill an element if it has negative volume
  death on inversion = on
  #Kill an element if it has become locally inverted (i.e., concave)
  criterion is element value of nodal_jacobian_ratio <= 0.0
  #Kill an element that is becoming very large (shells only)
  criterion is element value of perimeter_ratio > 10.0
end
```

6.5.4.8 Death on Proximity

```
BEGIN DEATH PROXIMITY
  BLOCK = <string list>block_names
```

```
    NODE SET = <string list>nodelist_names
    SURFACE = <string list>surface_names
    REMOVE BLOCK = <string list>block_names
    REMOVE NODE SET = <string list>nodelist_names
    REMOVE SURFACE = <string list>surface_names
    TOLERANCE = <real>search_tolerance
  END [DEATH PROXIMITY]
```

Sierra/SM provides the option to kill elements that get within a search tolerance of nodes in a specified mesh entity set. Proximity is measured by the smallest distance from the nodes and element centroids of the elements specified in the `BEGIN ELEMENT DEATH` command block to the nodes in the mesh entity set defined in the command block `BEGIN DEATH PROXIMITY`.

The commands `BLOCK`, `NODE SET`, `SURFACE`, `REMOVE BLOCK`, `REMOVE NODE SET`, and `REMOVE SURFACE` inside the command block `BEGIN DEATH PROXIMITY` form the mesh entity set of nodes that will be used to search against for the proximity measure.

The command `TOLERANCE` specifies the distance tolerance. Once an element's proximity measure is less than or equal to the distance tolerance the element will be killed.

6.5.4.9 Particle Conversion

```
  BEGIN PARTICLE CONVERSION
    PARTICLE SECTION = <string>section_name
    NEW MATERIAL = <string>material_name
      MODEL NAME = <string>model_name
    TRANSFER = <string>NONE|ALL(ALL)
    MAX NUM PARTICLES = <integer>num(1)
    MIN PARTICLE CONVERSION VOLUME = <real>vol
  END [PARTICLE CONVERSION]
```

Sierra/SM provides the option to convert elements to particles when they are killed, rather than simply removing them from the mesh. This option is activated by using the `BEGIN PARTICLE CONVERSION` command block inside the `ELEMENT DEATH` command block. Particle conversion can be useful when elements become too highly deformed to be represented by solid elements. This allows for the highly deformed solid elements to be removed from the mesh, while retaining the mass associated with those elements in the form of particles.

The name of a particle section block (defined with a `PARTICLE SECTION` command block, see Section 6.2.11) must be provided with the `PARTICLE SECTION` command line to define the properties of the created particles. Particles may be created that use the SPH or mass particle formulation, as defined by the section.

The material models for the created particles may be optionally prescribed by providing the name and model for a defined material in the `NEW MATERIAL` command. By default, the particle uses the material model of the solid element from which it has been converted, and all element data that can be retained, including material state variables, are transferred during conversion. This behavior may be modified by specifying the transfer type through the `TRANSFER` command. If

all the applicable element data should be transferred, the ALL option should be specified (this is the default if no transfer type is specified). If the particles should be created with reinitialized state data, the NONE option should be specified. Moreover, if a new material type is specified for conversion, the TRANSFER option will apply, but likely much of the material state data will not be transferable and will be lost as the element is converted to a particle. The best option when using a new material is to also specify a transfer type of NONE.

Particle conversion works with most element types including hexahedra, tetrahedra, shells, beams, and other particles. The converted particle is placed at the centroid of the removed element. For shells, beams and particles the new particle is placed inside of the lofted volume that the structural element represents. The mass of the particle equals the mass of the converted element. The velocity and/or temperature of the converted particle is set to the average velocity/temperature of the source element's nodes. The radius of the converted particle is computed to allow the particle to fit inside the void space left by removal of the solid element. If no particle can fit in that space, no particle is created. This typically occurs when an element has inverted or the center of mass is not on the interior of the element, as seen in Figure 6.10.

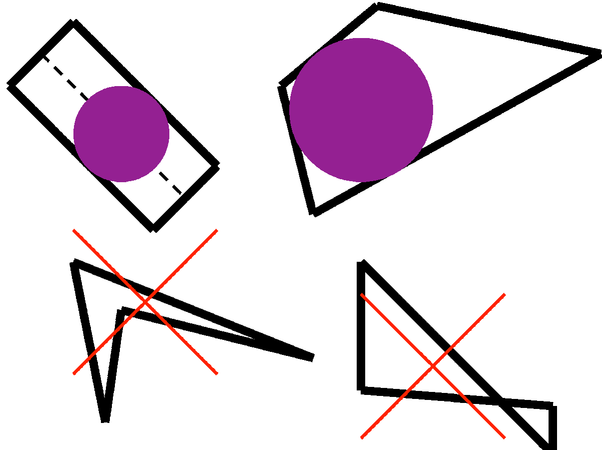


Figure 6.10: Examples of how an element is converted and when it is not converted.

A single element can optionally be converted into multiple particles. The number of particles to convert to is specified with the MAX NUM PARTICLES command and MIN PARTICLE CONVERSION VOLUME command. The usage of these commands is as follows:

If neither of these command lines is present, one particle is placed at the centroid of the element.

If only the MAX NUM PARTICLES command line is present, the element will be cut into the specified number of roughly equal volume sections. A particle will then be placed at the centroid of each of these sections.

If only the MIN PARTICLE CONVERSION VOLUME command line is present, the element will be subdivided until a number of volumes each of roughly the specified volume is reached. A new particle will then be placed at the centroid of each of these sub-volumes.

If both the MAX NUM PARTICLES and MIN PARTICLE CONVERSION VOLUME commands are specified, the element will be subdivided until either the minimum volume of the sub-parts is

reached, or the maximum number of sub-parts is reached. When one of these criteria is reached, a new particle is placed into each created sub-volume.

6.5.4.10 Active Periods

The following command lines can optionally be used in the ELEMENT DEATH command block:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

These command lines determine when element death is active. See Section 2.5 for more information about these optional command lines.

6.5.5 Cohesive Zone Setup Commands

The commands listed here are all related to adaptive insertion or activation of cohesive zones by element death.

```
COHESIVE SECTION = <string>sect_name
COHESIVE MATERIAL = <string>mat_name
COHESIVE MODEL = <string>model_name
COHESIVE ZONE INITIALIZATION METHOD = <string>NONE|
  ELEMENT STRESS AVG(NONE)
```

The first three of these commands are only applicable to adaptive insertion of cohesive elements by element death. They are used together to fully define the properties of the elements that are adaptively inserted.

The COHESIVE SECTION command is used to specify the name of a section used to define the section properties of the cohesive zone elements to be adaptively inserted. See Section 6.2.2 for a description of cohesive sections.

The COHESIVE MATERIAL command is used to specify the name of the material model to be used for the newly-created cohesive elements. This material model name is the user-provide name for the cohesive zone material provided by the parameter mat_name in the BEGIN PROPERTY SPECIFICATION FOR MATERIAL mat_name command block. The material models available for cohesive zones are documented in Section 5.3.

The COHESIVE MODEL command is used to select the material model to be used for the newly-created cohesive elements. This references the name of the material model model_name defined in a BEGIN PARAMETERS FOR MODEL model_name block. The material models available for cohesive zones are documented in Section 5.3.

The COHESIVE ZONE INITIALIZATION METHOD command controls the initialization of cohesive zones that are dynamically inserted or activated through element death. This command should

only be used if there are cohesive zones between elements and all nodes of those elements are initially attached together via multi-point constraints, or the INSERT COHESIVE ZONES death method is being used. When element death is used to deactivate the constraints or insert a cohesive element (with the DEATH METHOD = DEACTIVATE NODAL MPCS option or DEATH METHOD = INSERT COHESIVE ZONES option), the exposed cohesive zone can be given an initial state. The options are to either do nothing (NONE) or to initialize the tractions in the cohesive element based on the stresses in the two elements on either side of the cohesive zone (ELEMENT STRESS AVG). How the cohesive zone uses the initial traction will depend on the cohesive surface material model used.

6.5.6 Example

The following example provides instructions to kill elements in block_1 when they leave a bounding box. This type of element death can be useful in an analysis where some peripheral parts, because of fracture, separate and fly away from a central body, this central body being our part of interest. In this case, these peripheral parts no longer impact the solution. The instructions in this ELEMENT DEATH command block will cause the parts to be killed, thus speeding up the computation.

```
begin element death out_of_bounds
  block = block_1
  # check x coordinates
  criterion is avg nodal value of coordinates(1) >=  10
  criterion is avg nodal value of coordinates(1) <= -10
  # check y coordinates
  criterion is avg nodal value of coordinates(2) >=  10
  criterion is avg nodal value of coordinates(2) <= -10
  # check z coordinates
  criterion is avg nodal value of coordinates(3) >=  10
  criterion is avg nodal value of coordinates(3) <= -10
end element death out_of_bounds
```

6.5.7 Element Death Visualization

When an element dies, information about this element will still be sent, along with information for all other elements, to the Exodus II results file. (Chapter 9 describes the output of element variables to the results file.) The death status of the elements may be output to the results file by requesting element variable output for the element variable DEATH_STATUS. Including the command line

```
ELEMENT DEATH_STATUS as death_var
```

in a RESULTS OUTPUT command block (Chapter 9) will output this element variable with the name death_var in the results file.

The convention for DEATH_STATUS is as follows: An element with a value of 1.0 for DEATH_STATUS is a living element. An element with a value of 0.0 for DEATH_STATUS is a dead element. A value less than 0.0 indicates that the element was killed due to a code related issue (e.g. an unsupported geometry issue related to ACME). A value between 1.0 and 0.0 indicates an element

in the process of dying. A dying element has its material stress scaled down over a number of time steps. The current scaling factor for an element is given by DEATH_STATUS. Whether or not an element can have a value for DEATH_STATUS other than 0.0 or 1.0 will depend on whether or not you have used the DEATH STEPS option in the ELEMENT DEATH command block. If the number of steps over which death occurs is greater than 1, then DEATH_STATUS can be some value between 0.0 and 1.0.

If DEATH_STATUS is written to a results file, and the results file is used in a visualization program to examine the mesh for the model, it is possible to use DEATH_STATUS to exclude killed elements from any view of the model. A subset of the mesh showing just the living elements can be created by visualizing only those elements for which DEATH_STATUS = 1.0. The procedure for visualizing results in this way varies for different postprocessing tools.

When an analysis is using element death the log file contains a table of marker values that will be applied to dead elements. The marker values allow determining which elements where killed and by which criterion. The marker variables are stored in the element variable KILLED_BY_CRITERION and are available for output on the mesh results file. Additionally, a global count of how many elements were killed by each criterion is printed at the end of the run log file.

6.6 Particle Embedding

```
BEGIN PARTICLE EMBEDDING <string>particle_embedding_name
  BLOCK = <string>block_name
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string>block_name
  PARTICLE SECTION = <string>section_name
  NEW MATERIAL = <string>material_name
    MODEL NAME = <string>model_name
  TRANSFER = <string>NONE|ALL(ALL)
  MAX NUM PARTICLES = <integer>num(1)
  MIN PARTICLE CONVERSION VOLUME = <real>vol
END [PARTICLE EMBEDDING]
```

Particle embedding allows the user to specify element blocks in which each element is embedded with particles during initialization (i.e. before the first time step is taken). This capability can remove the need to create particles during element death and may be used for particle-solid interactions.

The embedded particles are created in the same way as during particle conversion for element death, except that the original solid element remains alive after the particles are created. Particles that are created are constrained to move with a material point in their controlling element (i.e. the element in which they were created). This constraint is released only if the controlling element dies. In addition, these embedded particles may interact with non-embedded particles to produce internal forces on the embedded particles. These internal forces are transferred to the controlling element using the embedded particle's location in the element. Note that the mass of the particles embedded in an element is assigned as in particle conversion for element death but that the particles' mass does not affect the solution until the controlling element dies. Once a particle's controlling element dies it is freed and behaves as a particle element.

The commands for this block are discussed in Section 6.5.4.9 and work for particle embedding in the same way, save `TRANSFER`. Particle embedding has two opportunities for transferring element data: at initialization and at controlling element death. For particle embedding the `TRANSFER` command applies to both of these transfers. If the `ALL` option is used, an embedded particle will be initialized with the initial element data of its controlling element and will have its element data reset to the evolved element data of its controlling element when its controlling element dies. If the `NONE` option is used, the embedded particle's element data will be initialized as a non-embedded particle's would be, and the particle's element data will not be modified at controlling element death.

6.7 Explicitly Computing Derived Quantities

```
BEGIN DERIVED OUTPUT
  COMPUTE AND STORE VARIABLE =
    <string>derived_quantity_name
END DERIVED OUTPUT
```

The above command block is used to explicitly compute and store a derived quantity into an internal field. This is useful if the field is needed by an outside capability such as a transfer or error estimation.

For example, to use a derived quantity in a transfer, you must use a `DERIVED OUTPUT` command block. To transfer the von Mises stress norm, you would use the following command block:

```
BEGIN DERIVED OUTPUT
  COMPUTE AND STORE VARIABLE = von_mises
END DERIVED OUTPUT
```

Tables 9.10 through 9.13 in Section 9 list element variables available for different types of elements. Variables that are only computed at user request are designated with a `yes` in the `Derived` column of these tables. These are the quantities that must be listed in a `BEGIN DERIVED OUTPUT` command block if they are to be transferred during a coupled analysis.

6.8 Mesh Rebalancing

Mesh rebalancing is a feature in Presto that may improve the efficiency of an analysis. Two command blocks can be used to control mesh rebalancing: REBALANCE and ZOLTAN PARAMETERS. The REBALANCE command block is required; the ZOLTAN PARAMETERS command block is optional. Sections 6.8.1 and 6.8.2 describe these command blocks.

6.8.1 Rebalance

```
BEGIN REBALANCE
  ELEMENT GROUPING TYPE = SPLIT SPH AND STANDARD ELEMENTS|
    UNIFORM UNIFIED(SPLIT SPH AND STANDARD ELEMENTS)
  INITIAL REBALANCE = ON|OFF(OFF)
  PERIODIC REBALANCE = ON|OFF|AUTO(OFF)
  REBALANCE STEP INTERVAL = <integer>step_interval
  LOAD RATIO THRESHOLD = <real>load_ratio
  COMMUNICATION RATIO THRESHOLD = <real>communication_ratio
  ZOLTAN PARAMETERS = <string>parameter_name
END [REBALANCE]
```

Initial decomposition of a mesh for parallel runs with Presto is done by a utility called loadbal. The initial decomposition provided by loadbal may not provide a decomposition for good-to-optimal parallel performance of Presto under certain circumstances. Therefore, Presto supports a simple mesh-rebalancing capability that can be used to improve the parallel performance of some problems. When mesh rebalancing is invoked, the parallel decomposition is changed, and elements are moved among the processors to balance the computational load and minimize the processor-to-processor communication. Mesh rebalancing may be useful in the following circumstances:

- The mesh decomposition produced by loadbal for sphere elements (i.e. for SPH or peridynamics) is nearly always poor. It is recommended that an initial mesh rebalance be done for all problems involving sphere elements.
- If a problem experiences very large deformations, periodic rebalancing may be helpful. In contact or SPH problems, communication is performed between physically nearby contact surfaces or SPH particles. To maintain optimum performance, it is helpful to have neighboring particles located on the same processors. Periodic mesh rebalancing can ensure that neighboring entities tend to remain on the same processor during large mesh deformations.

The REBALANCE command block is placed in the Presto region scope. The mesh rebalancing in Presto uses a mesh balancing library called Zoltan (Reference 13). Zoltan performs the actual rebalancing. By default, Presto creates a Zoltan object with a default set of parameters. However, a Zoltan object with a customized set of parameters can be created and referenced from the REBALANCE command block.

6.8.1.1 Rebalance Command Lines

```
ELEMENT GROUPING TYPE = SPLIT SPH AND STANDARD ELEMENTS|
  UNIFORM UNIFIED(SPLIT SPH AND STANDARD ELEMENTS)
INITIAL REBALANCE = on|off (off)
PERIODIC REBALANCE = on|off|auto (off)
REBALANCE STEP INTERVAL = <integer>step_interval
LOAD RATIO THRESHOLD = <real>load_ratio
COMMUNICATION RATIO THRESHOLD = <real>ratio
```

The above command lines control how and when the rebalancing is done.

The `ELEMENT GROUPING TYPE` command line specifies how the elements are grouped for rebalancing. If set to `SPLIT SPH AND STANDARD ELEMENTS`, which is the default behavior, this command causes SPH and standard elements to be split into separate groups. If it is set to the `UNIFORM UNIFIED` option, all elements can be grouped together.

The `INITIAL REBALANCE` command line is used to rebalance the mesh at time zero before any calculations occur. This option should be used if the initial mesh decomposition passed to Presto is poor.

If the `PERIODIC REBALANCE COMMAND` option is set to `on`, the mesh will be rebalanced every `step_interval` steps, where `step_interval` is the parameter specified by the `REBALANCE STEP INTERVAL` command line. If the option is `auto`, the mesh will be rebalanced every `step_interval` steps, when the communication ratio reaches a critical value, or when the load ratio reaches a critical value.

The communication ratio, currently defined only for SPH problems, is a measure of how much communication is required in the current mesh decomposition versus an estimate of the amount of communication with an optimal decomposition. Mesh rebalancing is expensive, so rebalancing should be done rarely. The `COMMUNICATION RATIO THRESHOLD` command line is used to specify the value of the communication ratio that triggers a rebalance. Setting this to a value between 1.25 and 1.5 is usually optimal.

The load ratio is the ratio of the current load (maximum number of elements on a processor divided by the average number of elements per processor) to that immediately after the previous rebalance. The `LOAD RATIO THRESHOLD` command specifies that a rebalance should occur when the load ratio exceeds the specified threshold. A specified value of 1.5 would trigger a rebalance when the load ratio is 50% higher than the previous rebalance (or initial decomposition, if no rebalances have yet occurred).

6.8.1.2 Zoltan Command Line

The command line

```
ZOLTAN PARAMETERS = <string>parameter_name
```

references a `ZOLTAN PARAMETERS` command block named `parameter_name`. Various parameters for Zoltan can be set in the `ZOLTAN PARAMETERS` command block. If you do not use the

ZOLTAN PARAMETERS command line, a default set of parameters is used. The default parameter command block is shown as follows:

```
BEGIN ZOLTAN PARAMETERS
  LOAD BALANCING METHOD = recursive coordinate bisection
    # string parameter
  OVER ALLOCATE MEMORY  = 1.5 # real parameter
  REUSE CUTS = true # string parameter
  ALGORITHM DEBUG LEVEL = 0 # integer parameter
  CHECK GEOMETRY = true # string parameter
  ZOLTAN DEBUG LEVEL = 0 # integer parameter
END ZOLTAN PARAMETERS
```

See Section 6.8.2 for a discussion of the ZOLTAN PARAMETERS command block. Section 6.8.2 lists the command lines that can be used to set Zoltan parameters.

6.8.2 Zoltan Parameters

```
BEGIN ZOLTAN PARAMETERS
  LOAD BALANCING METHOD  = <string>recursive coordinate bisection|
    recursive inertial bisection|hilbert space filling curve|
    octree
  DETERMINISTIC DECOMPOSITION = <string>false|true
  IMBALANCE TOLERANCE = <real>imb_tol
  OVER ALLOCATE MEMORY = <real>over_all_mem
  REUSE CUTS = <string>false|true
  ALGORITHM DEBUG LEVEL = <integer>alg_level
    #  0<=(alg_level)<=3
  CHECK GEOMETRY = <string>false|true
  KEEP CUTS = <string>false|true
  LOCK RCB DIRECTIONS = <string>false|true
  SET RCB DIRECTIONS = <string>do not order cuts|xyz|xzy|
    yzx|yxz|zxy|zyx
  RECTILINEAR RCB BLOCKS = <string>false|true
  RENUMBER PARTITIONS = <string>false|true
  OCTREE DIMENSION = <integer>oct_dimension
  OCTREE METHOD = <string>morton indexing|grey code|hilbert
  OCTREE MIN OBJECTS = <integer>min_obj  #  1<=(min_obj)
  OCTREE MAX OBJECTS = <integer>max_obj  #  1<=(max_obj)
  ZOLTAN DEBUG LEVEL = <integer>zoltan_level
    #  0<=(zoltan_level)<=10
  DEBUG PROCESSOR NUMBER = <integer>proc  #  1<=proc
  TIMER = <string> wall|cpu
  DEBUG MEMORY = <integer>dbg_mem  #  0<=(dbg_mem)<=3
END [ZOLTAN PARAMETERS]
```

The `ZOLTAN PARAMETERS` command block is used to set parameters for Zoltan (see Reference 13), a program that can be used for mesh rebalancing in Presto. The `ZOLTAN PARAMETERS` command block is used in association with the `REBALANCE` command block. A `REBALANCE` command block may reference a `ZOLTAN PARAMETERS` command block via the name, `parameter_name`, for the parameter command block. Reference Section 6.8.1 regarding the use of the `ZOLTAN PARAMETERS` command block for mesh rebalancing in Presto.

Setting the parameters for Zoltan involves some understanding of how Zoltan works. Consult Reference 13 for a discussion of the parameters that can be set by the various command lines in the `ZOLTAN PARAMETERS` command block. Note that some of the command lines in this command block have comments that provide additional information about the parameters. The "`#`" symbol precedes a comment.

In the above command block, = and `IS` are the allowed delimiters, which is different from the usual Presto convention of =, `IS`, and `ARE`. Note that the `ZOLTAN PARAMETERS` command block should be specified in the SIERRA scope when it is referenced from the `ZOLTAN PARAMETERS` command line in the `REBALANCE` command block. When the default set of parameters is used for a Zoltan object, the `ZOLTAN PARAMETERS` command block need not be included in the input file.

6.9 Remeshing

```
BEGIN REMESH
  MAX REMESH STEP INTERVAL = <integer>stepInterval (Infinity)
  MAX REMESH TIME INTERVAL = <real>timeInterval (Infinity)
  NEW MESH MAX EDGE LENGTH RATIO = <real>newMaxRatio (1.25)
  NEW MESH MIN EDGE LENGTH RATIO = <real>newMinRatio (0.25)
  NEW MESH MIN SHAPE = <real>newShape (0.125)
  REMESH AT MAX EDGE LENGTH RATIO = <real>maxCutoffRatio
    (newMaxEdgeLengthRatio*1.75)
  REMESH AT MIN EDGE LENGTH RATIO = <real>minCutoffRatio
    (newMinEdgeLengthRatio*0.25)
  REMESH AT SHAPE = <real>cutoffShape (0.025)
  CONTACT CLEANUP = AUTO|OFF|ON (AUTO)
  DEBUG OUTPUT LEVEL = <integer>level (0)
  MAX REMESH REBALANCE METRIC = <integer>rebalanceMetric (1.25)
  MAX NUMBER ELEMENTS = <integer>maxNumElem (500000)
  MIN SHAPE IMPROVEMENT = <real>minShapeImprovement(0.001)
  MAX NUM PATCH RECREATIONS = <integer>maxNumPatchRecreations(5)
  POISON CRITICAL VALUE = <integer>poisonCriticalValue(4)
  POISON SUBTRACTION INTERVAL =
    <integer>poisonSubtractionInterval(10)
  MAX ITERATIONS = <integer>maxIterations(5)

  BEGIN REMESH BLOCK SET
    #
    # Parameters for remesh block set
    #
  END [REMESH BLOCK SET]

  BEGIN ADAPTIVE REFINEMENT
    #
    # Parameters for adaptive refinement
    #
  END [ADAPTIVE REFINEMENT]
END [REMESH]
```

The `REMESH` command block, which is used within the region scope, sets parameters for remeshing a portion of the mesh. Remeshing involves removing badly shaped elements and inserting new elements of better quality that occupy the same volume. Depending on the degree to which the original elements are deformed, the new elements may occupy slightly more or slightly less volume than the original mesh. If regions of the mesh cannot be meshed with well-shaped elements having reasonable time steps, they may be removed entirely, potentially changing the topology. Examples of such regions include exterior slivers or very thin parts.

Remeshing is used exclusively with node-based tetrahedrons. Whenever possible, remeshing avoids moving, deleting or adding nodes. The node-based tetrahedron stores both material and

kinematic state information at the nodes and thus avoids many of the state advection problems associated with remeshing. However, some amount of state advection or non-conservation of energy, momentum, and mass may occur during remeshing.

In addition to remeshing locations with badly shaped elements, remeshing can be used to adaptively refine a mesh. Adaptive refinement may applied to an initially uniform mesh to produce one with a large number of elements in a region of interest and a coarse mesh elsewhere. Mesh refinement is fully conforming and works by altering the nodal characteristic edge lengths in an adapted mesh. The remeshing algorithm attempts to make the new mesh density consistent with the adapted characteristic edge lengths. Mesh adaptation happens only during remesh steps. Specifying a remesh step or a remesh time interval may be used to trigger adaptive refinement even if no large element deformation is taking place.

6.9.1 Remeshing Commands

The following commands control how and when remeshing is performed:

```
MAX REMESH STEP INTERVAL = <integer>stepInterval (Infinity)
MAX REMESH TIME INTERVAL = <real>timeInterval (Infinity)
NEW MESH MAX EDGE LENGTH RATIO = <real>newMaxRatio (1.25)
NEW MESH MIN EDGE LENGTH RATIO = <real>newMinRatio (0.25)
NEW MESH MIN SHAPE = <real>newShape (0.125)
REMESH AT MAX EDGE LENGTH RATIO = <real>maxCutoffRatio
  (newMeshMaxEdgeLengthRatio*1.75)
REMESH AT MIN EDGE LENGTH RATIO = <real>minCutoffRatio
  (newMeshMinEhgeLengthRatio*0.25)
REMESH AT SHAPE = <real>cutoffShape (0.025)
CONTACT CLEANUP = AUTO|OFF|ON (AUTO)
DEBUG OUTPUT LEVEL = <integer>level (0)
MAX REMESH REBALANCE METRIC = <integer>rebalanceMetric (1.25)
MAX NUMBER ELEMENTS = <integer> maxNumElems (500000)
MIN SHAPE IMPROVEMENT = <real>minShapeImprovement(0.001)
MAX NUM PATCH RECREATIONS = <integer>maxNumPatchRecreations(5)
POISON CRITICAL VALUE = <integer>poisonCriticalValue(4)
POISON SUBTRACTION INTERVAL =
  <integer>poisonSubtractionInterval(10)
MAX ITERATIONS = <integer>maxIterations(5)
```

The `MAX REMESH STEP INTERVAL` command defines the maximum number of simulation time steps that can pass between remeshing steps. Note that if any value of `MAX REMESH STEP INTERVAL` is specified, the code will perform a remesh step during initialization. This is useful for doing an initial mesh adaptation.

The `MAX REMESH TIME INTERVAL` command defines the maximum simulation time that can pass between remeshing steps.

The NEW MESH MAX EDGE LENGTH RATIO, NEW MESH MIN EDGE LENGTH RATIO, and NEW MESH MIN SHAPE commands specify target values for maximum edge length ratio, minimum edge length ratio, and minimum mesh shape in the new mesh. These metrics are described below in the definitions of the REMESH AT MAX EDGE LENGTH RATIO, REMESH AT MIN EDGE LENGTH RATIO, and REMESH AT SHAPE commands, respectively. It is recommended that the thresholds for triggering remeshing be at least a factor of four smaller than the new mesh values to prevent excessive remeshing calls.

The REMESH AT MAX EDGE LENGTH RATIO command specifies a threshold edge length ratio at which remeshing is triggered. The ratio is determined as the ratio of the longest edge connected to a given node to the characteristic initial mesh edge length associated with that node. Edge lengths that become large may no longer be able to represent required geometric details or accurately compute contact. The remeshing algorithm attempts to maintain edge lengths at roughly their original lengths, which may vary significantly across the mesh.

The REMESH AT MIN EDGE LENGTH RATIO command is similar to the REMESH AT MAX EDGE LENGTH RATIO command. This command triggers remeshing when the ratio of the shortest edge connected to a given node to the characteristic initial mesh edge length associated with that node falls below the specified threshold value. If edge lengths become too small, the mesh may have an excessively small time step or too many elements.

The REMESH AT SHAPE command defines the critical shape at which refinement is triggered. The element shape metric is based on a minimum tetrahedron face angle. The optimum tetrahedron has four equilateral faces and angles between those faces of approximately 50 degrees. A degenerate or inverted tetrahedron has a minimum face angle less than or equal to zero degrees. The element shape provides a measure of how close an element is to becoming degenerate or inverted.

The CONTACT CLEANUP command determines whether contact-related algorithms are run following remeshing. Remeshing near areas of contact may result in violations of contact constraints due to changes in element topology. Reapplying contact algorithms following remeshing ensures that contact requirements are maintained.

The DEBUG OUTPUT LEVEL command controls the amount of remeshing-related output printed to the the log file and terminal output. If this is set to zero, no output occurs, while larger numbers result in larger amounts of output. Output many include:

- Step-by-step values of metrics that trigger remeshing.
- The number of elements changed in a remesh step.
- Mass, energy, and momentum conservation balances during remeshing.

The MAX REMESH REBALANCE METRIC command line is used to control when the load rebalancing algorithm is called. During remeshing, the number of elements per processor can become out of balance, causing the parallel analyses to slow down. The specified rebalance_metric is a threshold for the ratio of the maximum number of elements on any processor to the average number of elements per processor. After each remeshing is performed, if that ratio exceeds the specified rebalance_metric, the rebalancing algorithm is called. The default value for this parameter is 1.25.

The MAX NUMBER ELEMENTS command line is used to set the maximum number of elements per processor that may be created during a remeshing analysis. If this value is exceeded, a warning is issued and no further remeshing is done. The default value for this parameter is 500000.

The MAX ITERATIONS command line is used to set the maximum number of remesh iterations that will be taken in a single time step. Lower values cause remesh to possibly run faster at the expense of worse elements or deleting difficult regions of the mesh. Valid values are 1 to infinity. The default value is 5.

The MAX NUM PATCH RECREATIONS command line is used to set the maximum number of patch recreations. This is an advanced remeshing algorithm option that relates to the iteration loop around remesh/parallel consolidation/overlap removal. Lower values cause remeshing to possibly run faster at the expense of quality elements (similar to max iterations described above). Valid values are 1 to infinity. The default value is 5.

The MIN SHAPE IMPROVEMENT command line is used to set what is considered a valid remesh operation. Generally larger values will cause remesh to function faster at expense of less overall mesh quality or more element deletion. Valid values are from 0.0 to approximately 0.1

The commands POISON CRITICAL VALUE and POISON SUBTRACTION INTERVAL are used to mark bad mesh regions. Each time an element is marked as bad all its nodes get a poison counter. Every subtraction interval steps all nodes remove one poison counter (never going below zero). If a node ever has more than the critical value of poison counters the node and all elements attached to it are removed from the mesh. This control is used to remove parts of the mesh that are frequently remeshing, likely due to a material model going haywire. Decreasing the critical value or increasing the subtraction interval cause the algorithm to more aggressively delete parts of the mesh. Increasing the critical value or decreasing the subtraction interval cause the code to avoid element deletion. Valid critical value 2 to Infinity. Valid subtraction interval 1 to Infinity. Note, setting subtraction interval to 1 in effect turns off the poison node element deletion.

6.9.2 Remesh Block Set

```
BEGIN REMESH BLOCK SET
  BLOCK = <string list>block_names
  REMOVE BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
END [REMESH BLOCK SET]
```

The BEGIN REMESH BLOCK SET command block, used in the scope of the REMESH command block, defines the element block or set of blocks that the given remesh command block applies to. The elements in the element block will be checked against the shape criteria and may be affected by remeshing. Note that it is currently assumed that all elements to be remeshed are part of a single material. Remeshing of equivalenced mesh blocks is not well supported.

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of blocks. See Section 7.1.1 for more information about the use of these command lines for creating a set of blocks used by the command block. There must be at least one BLOCK or

INCLUDE ALL BLOCKS command line in the command block.

The REMOVE BLOCK command line allows you to delete blocks from the set specified in the BLOCK and/or INCLUDE ALL BLOCKS command line(s) through the string list block_names. Typically, you would use the REMOVE BLOCK command line with the INCLUDE ALL BLOCKS command line. If you want to include all but a few of the element blocks, a combination of the REMOVE BLOCK command line and INCLUDE ALL BLOCKS should minimize input information.

6.9.3 Adaptive Refinement

```
BEGIN ADAPTIVE REFINEMENT
  #
  # adaptive refinement control commands
  ADAPT TYPE = NODE_PROXIMITY|POINT_PROXIMITY|
    SHARP_EDGE_PROXIMITY|FACE_PROXIMITY
  RADIUS = <real>radius
  ADAPT SHARP ANGLE = <real>angle (45)
  ADAPT SMOOTH ANGLE = <real>angle (10)
  GEOMETRIC POINT COORDINATES = <real>x <real>y <real>z
  ADAPT SIZE RATIO = <real>ratio
  TARGET MESH SIZE = <real>target_size
  #
  # tool mesh entity commands
  NODE SET = <string list>nodelist_names
  REMOVE NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  REMOVE BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  #
  # activation commands
  ACTIVE PERIODS = <string>period_names
  INACTIVE PERIODS = <string>period_names
END [ADAPTIVE REFINEMENT]
```

The ADAPTIVE REFINEMENT command block, used in the context of the REMESH command block, contains commands related to adaptive refinement. Adaptive refinement can be useful when the desired level of refinement in a region of the mesh is unknown when the mesh is created, or when a refinement region should move during the analysis. For example, complex physics may occur near moving contact boundaries. Adaptive refinement can be used to produce a fine mesh only near these boundaries and then have that refined region move with the contact boundaries as they move.

6.9.3.1 Adaptive Refinement Control Commands

The following commands used within the ADAPTIVE REFINEMENT command block control the adaptive refinement algorithm:

```
ADAPT TYPE = NODE_PROXIMITY|POINT_PROXIMITY|
  SHARP_EDGE_PROXIMITY|FACE_PROXIMITY
RADIUS = <real>radius
ADAPT SHARP ANGLE = <real>angle (45)
ADAPT SMOOTH ANGLE = <real>angle (10)
GEOMETRIC POINT COORDINATES = <real>x <real>y <real>z
ADAPT SIZE RATIO = <real>ratio
TARGET MESH SIZE = <real>target_size
```

The ADAPT TYPE command specifies the type of refinement. Currently four options exist:

- The NODE_PROXIMITY option refines areas of the remesh blocks that are near nodes of the tool blocks.
- The SHARP_EDGE_PROXIMITY option refines areas of the remesh blocks that are near exterior sharp edges of tool blocks. This command is specialized to refine regions for contact. To correctly represent contacting geometries, small elements are often required to allow material flow around sharp intruding edges.
- The POINT_PROXIMITY option refines all nodes near a defined point in space.
- The FACE_PROXIMITY option refines regions of the remesh blocks that are near faces of the tool blocks.

The RADIUS command defines the minimum distance from a node to a given tool object in the remesh block for that node to be considered for refinement. For example, if the refinement type is POINT_PROXIMITY, each node within a distance of RADIUS to the defined point will be refined. If refinement type is NODE_PROXIMITY, each node within a distance of RADIUS to any node in the tool block will be refined. Nodes that are closer to the refinement point have more refinement applied to them than nodes that are farther away. Thus, refinement grades the mesh smoothly from fine to coarse elements over the distance RADIUS.

The ADAPT SHARP ANGLE and ADAPT SMOOTH ANGLE commands are used only with the SHARP_EDGE_PROXIMITY refinement type and define edge sharpness thresholds for full, partial, and no mesh refinement. Edges with angles sharper (smaller) than SHARP ANGLE trigger full mesh refinement. Edges with angles smoother (larger) than ADAPT SMOOTH ANGLE do not trigger mesh refinement. Edges with angles that fall between these two angles trigger a reduced level of mesh refinement. These two commands define a transition zone over which the level of mesh refinement varies smoothly from full refinement to zero refinement.

The GEOMETRIC POINT COORDINATES command is used only with the POINT_PROXIMITY refinement type. This command defines the spatial location of the refinement tool point. This command may be used multiple times to define multiple refinement points.

The ADAPT SIZE RATIO command defines the degree of refinement applied. A value of 2.0 would specifies that the characteristic element size should be reduced by a factor of two in the refinement zone. A value less than one specifies that the mesh should be coarsened. Alternatively, the TARGET MESH SIZE may be used to directly specify the mesh size in the zone of adaptive refinement rather than specifying that size relative to the original mesh size.

6.9.3.2 Tool Mesh Entity Commands

The NODE_PROXIMITY and SHARP_EDGE_PROXIMITY types of adaptive refinement are based on proximity to tool mesh objects, which are collections of nodes or edges. For example, in a simulation of a sheet metal forming process, the punch tool may be defined as a tool mesh object, thus triggering adaptive refinement in the sheet metal that is in proximity to the punch tool.

These tool objects do not refine the body to be refined, but define objects that will trigger refinement if they are in the proximity of the body to be refined. The following commands are used within the ADAPTIVE REFINEMENT command block to define tool entities:

```
NODE SET = <string list>nodelist_names
REMOVE NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
REMOVE SURFACE = <string list>surface_names
BLOCK = <string list>block_names
REMOVE BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
```

These commands are a set of Boolean operators for constructing a set of mesh entities. See Section 7.1.1 for more information about the use of these command lines. Multiple types of these commands can be used together to form a set of nodes or edges.

It is important to note that for the NODE_PROXIMITY adaptivity type, any type of mesh entity can be used. A set of nodes is formed from those contained in all node sets, surfaces, and blocks listed. For the SHARP_EDGE_PROXIMITY adaptivity type, however, only blocks and surfaces may be used. That adaptivity type requires information about element edges that is not available from node sets.

6.9.3.3 Activation Commands

Adaptive refinement, if used, is by default active throughout an analysis. If desired, it can optionally be activated or deactivated during select time periods using the following commands in the ADAPTIVE REFINEMENT command block:

```
ACTIVE PERIODS = <string>period_names
INACTIVE PERIODS = <string>period_names
```

See Section 2.5 for more information regarding these commands.

6.10 References

1. Taylor, L. M., and D. P. Flanagan. *Pronto3D: A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912. Albuquerque, NM: Sandia National Laboratories, March 1989. pdf.

2. Rashid, M. M. "Incremental Kinematics for Finite Element Applications." *International Journal for Numerical Methods in Engineering* 36 (1993): 3937–3956. doi.

3. Dohrman, C. R., M. W. Heinstein, J. Jung, S. W. Key, and W. R. Witkowski. "Node-Based Uniform Strain Elements for Three-Node Triangular and Four-Node Tetrahedral Meshes." *International Journal for Numerical Methods in Engineering* 47 (2000): 1549–1568. doi.

4. Key, S. W., M. W. Heinstein, C. M. Stone, F. J. Mello, M. L. Blanford, and K. G. Budge. "A Suitable Low-Order, Tetrahedral Finite Element for Solids." *International Journal for Numerical Methods in Engineering* 44 (1999) 1785–1805. doi.

5. Key, S. W., and C. C. Hoff. "An Improved Constant Membrane and Bending Stress Shell Element for Explicit Transient Dynamics." *Computer Methods in Applied Mechanics and Engineering* 124, no. 1–2 (1995): 33–47. doi.

6. Laursen, T. A., S. W. Attaway, and R. I. Zadoks. *SEACAS Theory Manuals: Part III. Finite Element Analysis in Nonlinear Solid Mechanics*, SAND98-1760/3. Albuquerque, NM: Sandia National Laboratories, 1999. pdf.

7. Flanagan, D. P., and T. Belytschko. "A Uniform Strain Hexahedron and Quadrilateral with Orthogonal Hourglass Control." *International Journal for Numerical Methods in Engineering* 17 (1981): 679–706. doi.

8. Swegle, J. W. *SIERRA: PRESTO Theory Documentation: Energy Dependent Materials Version 1.0*. Albuquerque, NM: Sandia National Laboratories, October 2001.

9. Scherzinger, W. M., and D. C. Hammerand. *Constitutive Models in LAME*, SAND2007-5873. Albuquerque, NM: Sandia National Laboratories, September 2007. pdf.

10. Swegle, J. W., S. W. Attaway, M. W. Heinstein, F. J. Mello, and D. L. Hicks. *An Analysis of Smoothed Particle Hydrodynamics*, SAND93-2513. Albuquerque, NM: Sandia National Laboratories, March 1994. pdf.

11. Sjaardema, G. D. *Overview of the Sandia National Laboratories Engineering Analysis Code Access System*, SAND92-2292. Albuquerque, NM: Sandia National Laboratories, January 1993. pdf.

12. Brown, K. H., R. M. Summers, M. W. Glass, A. S. Gullerud, M. W. Heinstein, and R. E. Jones. *ACME: Algorithms for Contact in a Multiphysics Environment*, API Version, 2.2, SAND2004-5486. Albuquerque, NM: Sandia National Laboratories, October 2001. pdf.

13. Karen Devine, Erik Boman, Robert Heapby, Bruce Hendrickson, Courtenay Vaughan, "Zoltan Data Management Service for Parallel Dynamic Applications." *Computing in Science and Engineering* 4 (2002): 90–97. doi. See also ZOLTAN web site. link.

14. Monaghan, J. "SPH without a tensile instability." *Journal of Computational Physics* 12, (2000): 622. doi.

15. Q. Yang, A. Mota, M. Ortiz. "A class of variational strain-localization finite elements." *International Journal for Numerical Methods in Engineering*, (2005) 62:1013-1037. doi.

Chapter 7

Boundary Conditions and Initial Conditions

Sierra/SM offers a variety of options for defining boundary and initial conditions. Typically, boundary and initial conditions are defined on some subset of mesh entities (node, element face, element) defining a model. Sierra/SM offers a flexible means to define subsets of mesh entities. Section 7.1.1 describes commands that will let you define some subset of a mesh entity using a collection of commands that constitute a set of Boolean operators.

The remaining parts of this chapter discuss the following functionality:

- Section 7.3 presents methods for setting the initial values of variables in Sierra/SM. Sierra/SM has the flexibility to set a complex initial state for some variable such as nodal velocity or element stress.
- Kinematic boundary conditions typical of those you would expect in a solid mechanics code (fixed displacement, prescribed acceleration, etc.) are options in Sierra/SM and described in Section 7.4. Most of these boundary conditions let you specify a time history using a function, a user subroutine, or by reading values from a mesh file.
- Section 7.5 documents a number of initial velocity options available in Sierra/SM.
- Force boundary conditions typical of those you would expect in a solid mechanics code (prescribed force, traction, etc.) are options in Sierra/SM and described in Section 7.6. Most of these force boundary conditions let you specify a time history using a function or a user subroutine.
- Section 7.7 discusses the gravity load option. A gravity load is a body force boundary condition.
- Section 7.8 provides a description of centripetal force boundary conditions can be applied about a prescribed axis.
- Section 7.9 details a number of options available for describing a temperature field in Sierra/SM.
- Section 7.10 details the options available for describing a pore pressure field.

- Section 7.11 documents how fluid pressure boundary conditions can be applied as a hydrostatic pressure to each node pertaining to a set of prescribed surfaces.
- Section 7.12 describes a number of specialized boundary conditions.

7.1 General Boundary Condition Concepts

There are general principles that apply to all of the available types of boundary conditions. To apply a boundary condition, a set of mesh entities and the magnitude and/or direction in which it is to be applied must be specified. Sierra/SM provides several methods for both specifying the set of mesh entities and for prescribing how the boundary condition is to be applied. The general concepts on how this is done are applicable to all of the boundary condition types, and are described in the following sections.

7.1.1 Mesh-Entity Assignment Commands

A number of standard command lines exist to define a set of mesh entities (node, element face, element) associated with some type of boundary, initial, or load condition. All these command lines exist within the command blocks for the various prescribed conditions, which in turn exist within the region scope. These command lines, taken collectively, constitute a set of Boolean operators for constructing sets of mesh entities.

The first set of command lines we will consider is as follows:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
```

In the above command lines, the string list `nodelist_names` is used to represent one or more node sets as discussed in Section 1.5. A node set is referenced as `nodelist_id`, where id is some integer. For example, suppose you have three node lists in your model: 10, 23, and 105. If you want to combine all these node lists so that they form one set of nodes for, say, your boundary condition or initial condition, then you would use the command line:

```
NODE SET = nodelist_10 nodelist_23 nodelist_105
```

This convention applies as well to any surface-related command line that uses the string list `surface_names` or any block-related command line that uses the string list `block_names`.

The `NODE SET` command line associates a set of nodes with an initial, boundary, or load condition. A condition may be applied to multiple node sets by putting multiple node set names on the command line or by repeating the command line multiple times.

The SURFACE command line associates a set of element faces or their attached nodes with a boundary, initial, or load condition. A condition may be applied to multiple surfaces by putting multiple surface names on the command line or by repeating the command line multiple times. For example, suppose we wish to use the fixed displacement kinematic boundary condition. Although this is a nodal boundary condition (the condition is applied to individual nodes), a SURFACE command line can be used to establish the set of nodes. If the command line

```
SURFACE = surface_101
```

appears in a fixed displacement boundary condition, then all the nodes associated with surface 101 will be associated with the boundary condition.

The BLOCK command line associates a set of elements and its nodes and faces with a boundary condition. A boundary condition may be applied to multiple blocks by putting multiple block names on the command line or by repeating the command line multiple times.

The BLOCK must be used for kinematic boundary conditions on rigid bodies. If a block has been defined as a rigid body, the specified kinematic condition will be applied to the rigid body reference node.

For example, suppose we wish to use the fixed displacement kinematic boundary condition as in the previous example. If the command line

```
BLOCK = block_50
```

appears in a fixed displacement kinematic boundary condition, then all the nodes associated with block 50 will be associated with the boundary condition.

The INCLUDE ALL BLOCKS command line associates all blocks and hence all elements and nodes in the model with a boundary, initial, or load condition. The block command lines associated with boundary conditions, initial conditions, and gravity will NOT generate surfaces.

Any combination of the above command lines can be used to create a union of mesh entities. Suppose, for example, that the command lines

```
NODE SET = nodelist_2
SURFACE = surface_3
```

appear in a FIXED DISPLACEMENT command block for a kinematic boundary condition. The set of nodes associated with the boundary condition will be the union of the set of nodes associated with surface 3 and the set of nodes associated with node set 2.

When a union of mesh entities is created by using two or more of the above command lines, a node or element face may appear in more than one node set, surface or block. However, the prescribed condition is applied to each node or face only once. For example, node 67 may be a part of nodelist 2 and surface 3 but the boundary condition will only be applied to node 67 once.

The set of mesh entities associated with a boundary, initial, or load condition can be edited (mesh entities can be deleted from the set) by using the following command lines:

```
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

The REMOVE NODE SET command line deletes the nodes in the specified node set from the set of nodes used by the condition.

The REMOVE SURFACE command line deletes a set of element faces and their associated nodes from the set of element faces used by the prescribed condition.

The REMOVE BLOCK command line deletes a set of elements and their associated nodes from the set of elements used by the prescribed condition.

7.1.2 Methods for Specifying Boundary Conditions

There are three main methods which can be used to prescribe most types of boundary conditions available in Sierra/SM.

- The boundary condition can be prescribed using commands in the input file. These commands are categorized as “specification commands” in this document. Depending on the type of the boundary condition, it is be necessary to prescribe its direction and/or magnitude. Boundary conditions can be specified this way when a set of mesh entities is to experience a similar condition with a time variation that can be expressed by a function. One of the following commands is used to specify the direction of the boundary condition: COMPONENT, DIRECTION, CYLINDRICAL AXIS, or RADIAL AXIS. The magnitude is defined using one of MAGNITUDE, FUNCTION or ANGULAR VELOCITY. These commands are used in various combinations depending on the type of the boundary condition. The details of how to use them are provided in the descriptions of the various boundary condition types.
- If the nature of the boundary condition is such its variation in time and space can not be described easily by the combination of a function and a direction, it may be necessary to use a user-defined subroutine. User subroutines provide a very general capability to define how kinematic or force boundary conditions are applied. The use of user-defined subroutines does increase the complexity of defining the model, however. The user must write and debug the subroutine and compile and link it in with Sierra/SM. Because of the added complexity, user subroutines should only be used if the needed capability is not provided by the other methods of prescribing boundary conditions.
- For some types of boundary conditions, the values of the field to be prescribed can be read in from an existing output database. This is often used as a method to transfer results from one analysis code to another. One of the common uses for this capability is to compute temperatures using a thermal code, and then transfer the temperature fields to Sierra/SM to study combined mechanical and thermal effects. This capability can be used either to read in initial values or to read in a series of values that vary over time.

In the following sections describing specific types of boundary conditions, the commands are grouped according to these three categories.

7.2 Initial Mesh Modification

```
BEGIN INITIAL MESH MODIFICATION
  ROTATE BLOCK <string list>block_names
    ANGLE <real>angle
    ABOUT ORIGIN <real>Ox <real>Oy <real>Oz
    DIRECTION <real>Dx <real>Dy <real>Dz
  MOVE BLOCK <string list>block_names
    X <real>Mx  Y <real>My  Y <real>Mz
END
```

A handful of transformations can be applied to the initial mesh from within the input deck. These transformations happen prior to any other operations being performed such as contact overlap removal or evaluating initial conditions.

The `ROTATE BLOCK` command will rotate the named set of blocks, `block_names`. The rotation of these blocks is defined by a rotation axis and angle of rotation about that axis. The origin and direction of the rotation axis are defined by the `ABOUT ORIGIN` and `DIRECTION` commands, respectively. The `ANGLE` command sets the rotation angle, in degrees, by following the right-hand rule.

The `MOVE BLOCK` command will translate the named set of blocks with the specified `X`, `Y`, and `Z` translation vector.

The syntax and behavior of the mesh modification commands mimic those in Cubit. The defined transformations will be evaluated in order. Thus moving a block then rotating it about an axis will give a different result from rotating the block first then moving it.

7.3 Initial Variable Assignment

```
BEGIN INITIAL CONDITION
  #
  # mesh-entity set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # variable identification commands
  INITIALIZE VARIABLE NAME = <string>var_name
  VARIABLE TYPE = NODE|EDGE|FACE|ELEMENT|GLOBAL
  #
  # specification command
  MAGNITUDE = <real list>initial_values
  #
  # Weibull probability distribution commands
  WEIBULL SHAPE = <real>shapeVal
  WEIBULL SCALE = <real>scaleVal
  WEIBULL MEDIAN = <real>medianVal
  WEIBULL SEED = <int>seedVal (123456)
  WEIBULL SCALING FIELD TYPE = NODE|EDGE|FACE|ELEMENT|GLOBAL (ELEMENT)
  WEIBULL SCALING FIELD NAME = <string>fieldVal (VOLUME)
  WEIBULL SCALING REFERENCE VALUE = <real>value
  WEIBULL SCALING EXPONENT SCALE = <real>scaleVal (1.0)
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>subroutine_name |
    ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
```

```
    # additional command
    SCALE FACTOR = <real>scale_factor(1.0)
  END [INITIAL CONDITION]
```

Sierra/SM supports a general initialization procedure for setting the value of any variable. This procedure can be used to set material state variables, shell thickness, initial stress, etc. The initialization is performed both before and after the element and material model initialization. This allows the elements and material models to compute other initial variables based on variables specified by the user and also ensures that the variables specified by the user are not overwritten by the elements and material models. However, there is minimal checking in Sierra/SMto ensure that the changes made yield a consistent system. There is also no guarantee that the changes will not be overwritten or misinterpreted by some other internal routine depending on what variable is being changed. Thus, caution is advised when using this capability.

The `INITIAL CONDITION` command block, which appears in the region scope, is used to select a method and set values for initializing a variable. The command block specifies the initial value of a global variable or a variable associated with a set of mesh entities, i.e., nodes, edges, faces, or elements. The user has three options for setting initial values: with a constant magnitude, with an input mesh variable, or by a user subroutine. Only one of these three options can be specified in the command block.

The command block contains five groups of commands—mesh-entity set, variable identification, magnitude, input mesh variable, and user subroutine. In addition to the command lines in the five groups, there is one additional command line: `SCALE FACTOR`. Following are descriptions of the different command groups and the `SCALE FACTOR` command line.

7.3.1 Mesh-Entity Set Commands

The `mesh-entity set commands` portion of the `INITIAL CONDITION` command block specifies the nodes, element faces, or elements associated with the variable to be initialized. This portion of the command block can include some combination of the following command lines:

```
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes, element faces, or elements. See Section 7.1.1 for more information about the use of these command lines for mesh entities. There must be at least one `NODE SET`, `SURFACE`, `BLOCK`, or `INCLUDE ALL BLOCKS` command line in the command block.

7.3.2 Variable Identification Commands

Any variable used in the INITIAL CONDITION command block must exist in Sierra/SM. The variable can be any currently defined variable in Sierra/SM or any user-defined variable created with the USER VARIABLE command block (see Section 11.2.4).

There are two command lines that identify the variable:

```
INITIALIZE VARIABLE NAME = <string>var_name
VARIABLE TYPE = [NODE|EDGE|FACE|ELEMENT|GLOBAL]
```

The INITIALIZE VARIABLE NAME command line gives the name of the variable for which initial values are being assigned. As mentioned, the string var_name must be some variable known to Sierra/SM; it cannot be an arbitrary user-selected name.

The VARIABLE TYPE command line provides additional information about the variable being initialized. The options NODE, EDGE, FACE, ELEMENT, and GLOBAL on the command line indicate whether the variable is, respectively, a nodal, edge, face, element, or global quantity. One of these options must appear in the VARIABLE TYPE command line.

Both of these command lines are required regardless of the option selected to set values for the variable.

7.3.3 Specification Command

If the constant magnitude command is used, one or more initial values are specified directly in the command block. This is done using the following command line:

```
MAGNITUDE = <real list>initial_values
```

The initial_values specified on the MAGNITUDE command line will set the values for the variable given by var_name in the INITIALIZE VARIABLE NAME command line. The number of values is dependent on the type of the variable specified in the INITIALIZE VARIABLE NAME command line. For example, if the user wanted to initialize the velocity at a set of nodes, three quantities would have to be specified since the velocity at a node is a vector quantity. If the user wanted to initialize the stress tensor for a set of uniform-gradient, eight-node hexahedral elements, six quantities would have to be specified since the stress tensor for this element type is described with six values.

7.3.4 Weibull Probability Distribution Commands

The field to be initialized can optionally be initialized or scaled with random numbers generated to conform to a specified Weibull probability distribution function. This is accomplished by including some of the following commands:

```
WEIBULL SHAPE = <real>shapeVal
WEIBULL SCALE = <real>scaleVal
WEIBULL MEDIAN = <real>medianVal
WEIBULL SEED = <int>seedVal (123456)
```

The Weibull commands may be used in conjunction with the `MAGNITUDE` command or the `READ VARIABLE` command to set the initial value, m_i, that will be scaled by the distribution. Otherwise m_i starts off equal to 1.0.

Two and only two of the parameters `WEIBULL SHAPE`, `WEIBULL SCALE`, and `WEIBULL MEDIAN` may be specified. The relationship among `WEIBULL SHAPE`, k, (also known as the Weibull modulus) `WEIBULL SCALE`, λ, and `WEIBULL MEDIAN`, μ, is shown in the following equation:

$$\mu = \lambda(ln(2)^{\frac{1}{k}}) \tag{7.1}$$

The values at each point can be further scaled by another factor, α_i. If no scaling commands are used then $\alpha_i = 1.0$. Scaling is accomplished by using the Weibull scaling commands:

```
WEIBULL SCALING FIELD TYPE = NODE|EDGE|FACE|ELEMENT|GLOBAL (ELEMENT)
WEIBULL SCALING FIELD NAME = <string>fieldVal (VOLUME)
WEIBULL SCALING REFERENCE VALUE = <real>value
WEIBULL SCALING EXPONENT SCALE = <real>scaleVal (1.0)
```

The scaling commands are implemented to allow the use of any field variable for scaling; however, the field type must match the type of field being set. The `WEIBULL SCALING FIELD NAME` gives the name of a variable and the `WEIBULL SCALING FIELD TYPE` gives the variable field type which can be one of nodal, edge, face, element, or global type. The `WEIBULL SCALING FIELD TYPE` and `WEIBULL SCALING FIELD NAME` default to `ELEMENT` and `VOLUME` respectively. The `WEIBULL SCALING REFERENCE VALUE` gives the reference value v_{ref}. A `WEIBULL SCALING EXPONENT SCALE`, β, may also be defined and will default to 1.0 if not defined. The scaling factor α_i is formed with the following formula:

$$\alpha_i = \left(\frac{v_{ref}}{v_i}\right)^{\frac{\beta}{k}} \tag{7.2}$$

where v_i is the value of a field variable specified with `WEIBULL SCALING FIELD TYPE` and `WEIBULL SCALING FIELD NAME` at the current point i, and the rest of the variables are those defined previously.

The scaled initial value is determined by the equation:

$$InitialValue = m_i \alpha_i \mu \left(\frac{ln(R)}{ln(\frac{1}{2})}\right)^{\frac{1}{k}} \tag{7.3}$$

where R is a random number in an uniform distribution between 0 and 1.

7.3.5 External Mesh Database Commands

If the external database option is used, the initial values for a variable are read from an external mesh database. As an example, suppose the mesh file contains a set of nodal temperatures. These temperature values (which can vary for each node) can be used to initialize a temperature value associated with each node.

The values are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe initial conditions:

```
READ VARIABLE = <string>var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the variable from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database. The number of values associated with the variable in the mesh file must be the same number associated with the variable name specified in the INITIALIZE VARIABLE NAME command line. For example, if the variable specified by the INITIALIZE VARIABLE NAME has a single value, then the variable specified in the mesh file must also have a single value.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the initial conditions. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors. Also, the COPY VARIABLE command only handles nodal variables.

The variable name used on the mesh file can be arbitrary. The name can be identical to or different from the variable name specified on the INITIALIZE VARIABLE NAME command line.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is to use the value of the variable at the initial time in the analysis to prescribe the initial condition. The time history is interpolated as needed for an initial analysis time that does not correspond exactly to a time on the mesh file. The TIME command line can optionally be used to select a specific time to initialize a variable. If the specified time on the TIME command line does

not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.3.6 User Subroutine Commands

If the user subroutine option is used, the initial values will be calculated by a subroutine that is written by the user explicitly for this purpose. The subroutine will be called by Sierra/SM at the appropriate time to perform the calculations.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name |
  SURFACE SUBROUTINE = <string>subroutine_name |
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

Only one of the first three command lines listed above can be specified in the command block. The particular command line selected depends on the mesh-entity type of the variable being initialized. For example, variables associated with nodes would be initialized if you are using the NODE SET SUBROUTINE command line, variables associated with faces if you are using the SURFACE SUBROUTINE command line, and variables associated with elements if you are using the ELEMENT BLOCK SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the selected subroutine command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The application of user subroutines for variable initialization is essentially the same as the application of user subroutines in general. See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

When the user subroutine option is used for variable initialization, the user subroutine is called only once. Also, when a user subroutine is being used, the returned value is the new (initial) variable value at each mesh entity, and the flags array is ignored.

7.3.7 Additional Command

This command line provides an additional option for the INITIAL CONDITION command block:

```
SCALE FACTOR = <real>scale_factor(1.0)
```

Any initial value can be scaled by use of the SCALE FACTOR command line. An initial value generated by any one of the three initial-value-setting options in this command block (i.e., constant magnitude, input mesh, or user subroutine) will be scaled by the real value scale_factor.

7.4 Kinematic Boundary Conditions

The various kinematic boundary conditions available in Sierra/SM are described in this section. The kinematic boundary conditions are nested inside the region scope.

Kinematic constraints can potentially be in conflict with other constraints. Refer to Appendix E for information on how conflicting constraints are handled.

7.4.1 Fixed Displacement Components

```
BEGIN FIXED DISPLACEMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  COMPONENT = <string>X/Y/Z | COMPONENTS = <string>X/Y/Z
  #
  # additional commands
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [FIXED DISPLACEMENT]
```

The `FIXED DISPLACEMENT` command block fixes displacement components (X, Y, Z, or some combination thereof) for a set of nodes. This command block contains two groups of commands—node set and component. Each of these command groups is basically independent of the other. In addition to the command lines in the two command groups, there are two additional command lines: `ACTIVE PERIODS` and `INACTIVE PERIODS`. These are used to activate or deactivate this kinematic boundary condition for certain time periods.

For rigid bodies, any kinematic boundary condition is applied directly to the rigid body reference node. The kinematics for the nodes attached to the rigid body are applied via the rigid body constraint conditions. Since rigid bodies are defined using element blocks, only the `BLOCK` command may be used to specify fixed displacement conditions on rigid bodies.

Following are descriptions of the different command groups.

7.4.1.1 Node Set Commands

The node set commands portion of the FIXED DISPLACEMENT command block specifies the nodes associated with the boundary condition. This portion of the command block can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.1.2 Specification Commands

There are two component specification commands available in the FIXED DISPLACEMENT command block:

```
COMPONENT = X/Y/Z | COMPONENTS = X/Y/Z
```

The displacement components that are to be fixed can be specified with either the COMPONENT command line or the COMPONENTS command line. There can be only one COMPONENT command line or one COMPONENTS command line in the command block. The user can specify any combination of the components to be fixed, as in X, Z, X Z, Y X, etc.

7.4.1.3 Additional Commands

The ACTIVE PERIODS and INACTIVE PERIODS command lines can optionally appear in the FIXED DISPLACEMENT command block:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

These command lines determine when the boundary condition is active. See Section 2.5 for more information about these optional command lines.

Warning: If FIXED DISPLACEMENT becomes active the displacement is returned to zero. If the intention is to maintain the current displacement when activated PRESCRIBED VELOCITY with a value of zero should be used.

7.4.2 Prescribed Displacement

```
BEGIN PRESCRIBED DISPLACEMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED DISPLACEMENT]
```

The PRESCRIBED DISPLACEMENT command block prescribes a displacement field for a given set of nodes. The displacement field associates a vector giving the magnitude and direction of the displacement with each node in the set of nodes. The displacement field may vary over time and space. If the displacement field has only a time-varying magnitude and uses one of four methods for setting direction, the specification commands in the above command block can be used to specify the displacement field. If the displacement field is more complex, a user subroutine is

used to specify the displacement field. The displacement field can also be read from an external mesh database. In a given boundary condition command block, commands from only one of the command groups (specification commands, user subroutine commands, or external database commands) may be used.

For rigid bodies, any kinematic boundary condition is applied directly to the rigid body reference node. The kinematics for the mesh nodes attached to the rigid body are applied via the rigid body constraint conditions. Since rigid bodies are defined using element blocks, only the BLOCK command may be used to specify prescribed displacement conditions on rigid bodies.

The PRESCRIBED DISPLACEMENT command block contains four groups of commands—node set, function, user subroutine, and external database. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with the specification commands the user subroutine commands, or the external database command. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups and the SCALE FACTOR and ACTIVE PERIODS command lines.

7.4.2.1 Node Set Commands

The node set commands portion of the PRESCRIBED DISPLACEMENT command block defines a set of nodes associated with the prescribed displacement field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.2.2 Specification Commands

If the specification commands are used, the displacement vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED DISPLACEMENT command block.

Following are the command lines used to specify the prescribed displacement with a direction and a function:

```
DIRECTION = <string>defined_direction |
  COMPONENT = <string>X|Y|Z |
  CYLINDRICAL AXIS = <string>defined_axis |
  RADIAL AXIS = <string>defined_axis
FUNCTION = <string>function_name
```

The displacement can be specified along an arbitrary user-defined direction, along a component direction (X, Y, or Z), along the azimuthal direction in a cylindrical coordinate system (defined in reference to an axis), or along a radial direction (defined in reference to an axis). Only one of these options (i.e., command lines) is allowed. The displacement is prescribed only in the specified direction. A prescribed displacement boundary condition does not influence the motion in directions orthogonal to the prescribed direction.

- The `DIRECTION` command line is used to prescribe displacement in an arbitrary user-defined direction. The name in the string `defined_direction` is a reference to a direction, which is defined using the `DEFINE DIRECTION` command block within the SIERRA scope.
- The `COMPONENT` command line is used to specify that the prescribed displacement vector lies along one of the component directions. The `COMPONENT` command line is a shortcut to an internally defined direction vector; for example, component `x` corresponds to using direction vector (1, 0, 0).
- The `CYLINDRICAL AXIS` command line is used to specify that the prescribed displacement is to be applied in the azimuthal direction of a cylindrical coordinate system. The string `defined_axis` refers to the name of the axis of the cylindrical coordinate system, and which is defined via a `DEFINE AXIS` command block in the SIERRA scope. The displacement is prescribed as a rotation in radians about the axis. Nodes with this type of boundary condition are free to move in the radial and height directions in the cylindrical coordinate system. Restraints can be placed on the node set in those directions if desired by applying separate kinematic boundary conditions that contain `RADIAL AXIS` or `DIRECTION` commands that refer to the same axis. Note that this type of boundary condition is not a rotational boundary condition; it only affects translational degrees of freedom.

Known Issue: If a prescribed displacement with the `CYLINDRICAL AXIS` option is applied to nodes that fall on the axis, it will have no effect. Separate boundary conditions should be applied to those nodes to fix them in the plane normal to the axis.

- The `RADIAL AXIS` command line requires an axis definition that appears in the SIERRA scope. The string `defined_axis` uses an `axis_name` that is defined in the SIERRA scope (via a `DEFINE AXIS` command line). For this option, a radial line is drawn from a node to the radial axis. The prescribed displacement vector lies along this radial line from the node to the radial axis.

The magnitude of the displacement is specified by the FUNCTION command line. This references a function_name (defined in the SIERRA scope using a DEFINITION FOR FUNCTION command block) that specifies the magnitude of the displacement vector as a function of time. The magnitude can be scaled by use of the SCALE FACTOR command line described in Section 7.4.2.5.

7.4.2.3 User Subroutine Commands

If the user subroutine option is used, the displacement vector may vary spatially at any given time for each of the nodes in the node set associated with the particular PRESCRIBED DISPLACEMENT command block. The user subroutine option allows for a more complex description of the displacement field than do the specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a displacement direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the displacement field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.4.2.5.

See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

7.4.2.4 External Mesh Database Commands

If the external database option is used, the displacement vector (or specified components of the vector) is read from an external mesh database. The displacements are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The

finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the displacement:

```
READ VARIABLE = <string>var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the displacement vector from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the displacement. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.4.2.5 Additional Commands

These command lines in the PRESCRIBED DISPLACEMENT command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the displacement in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the displacement from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.4.3 Prescribed Velocity

```
BEGIN PRESCRIBED VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED VELOCITY]
```

The `PRESCRIBED VELOCITY` command block prescribes a velocity field for a given set of nodes. The velocity field associates a vector giving the magnitude and direction of the velocity with each node in the node set. The velocity field may vary over time and space. If the velocity field has only a time-varying magnitude and uses one of four methods for setting direction, the specification commands in the above command block can be used to specify the velocity field. If the velocity field is more complex, a user subroutine is used to specify the velocity field. The velocity field can also be

read from an external mesh database. In a given boundary condition command block, commands from only one of the command groups (specification commands, user subroutine commands, or external database commands) may be used.

For rigid bodies, any kinematic boundary condition is applied directly to the rigid body reference node. The kinematics for the mesh nodes attached to the rigid body are applied via the rigid body constraint conditions. Since rigid bodies are defined using element blocks, only the BLOCK command may be used to specify prescribed velocity conditions on rigid bodies.

The PRESCRIBED VELOCITY command block contains four groups of commands—node set, function, user subroutine, and external database. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.4.3.1 Node Set Commands

The node set commands portion of the PRESCRIBED VELOCITY command block defines a set of nodes associated with the prescribed velocity field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.3.2 Specification Commands

If the specification commands are used, the velocity vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED VELOCITY command block.

Following are the command lines used to specify the prescribed velocity with a direction and a function:

```
DIRECTION = <string>defined_direction |
```

```
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
```

The velocity can be specified along an arbitrary user-defined direction, along a component direction (X, Y, or Z), along the azimuthal direction in a cylindrical coordinate system (defined in reference to an axis), or along a radial direction (defined in reference to an axis). Only one of these options (i.e., command lines) is allowed. The velocity is prescribed only in the specified direction. A prescribed velocity boundary condition does not influence the motion in directions orthogonal to the prescribed direction.

- The `DIRECTION` command line is used to prescribe velocity in an arbitrary user-defined direction. The name in the string `defined_direction` is a reference to a direction, which is defined using the `DEFINE DIRECTION` command block within the SIERRA scope.

- The `COMPONENT` command line is used to specify that the prescribed velocity vector lies along one of the component directions. The `COMPONENT` command line is a shortcut to an internally defined direction vector; for example, component `x` corresponds to using direction vector (1, 0, 0).

- The `CYLINDRICAL AXIS` command line is used to specify that the prescribed velocity is to be applied in the azimuthal direction of a cylindrical coordinate system. The string `defined_axis` refers to the name of the axis of the cylindrical coordinate system, and which is defined via a `DEFINE AXIS` command block in the SIERRA scope. The velocity is prescribed as a rotation in radians about the axis. Nodes with this type of boundary condition are free to move in the radial and height directions in the cylindrical coordinate system. Restraints can be placed on the node set in those directions if desired by applying separate kinematic boundary conditions that contain `RADIAL AXIS` or `DIRECTION` commands that refer to the same axis. Note that this type of boundary condition is not a rotational boundary condition; it only affects translational degrees of freedom.

Known Issue: If a prescribed velocity with the `CYLINDRICAL AXIS` option is applied to nodes that fall on the axis, it will have no effect. Separate boundary conditions should be applied to those nodes to fix them in the plane normal to the axis.

- The `RADIAL AXIS` command line requires an axis definition that appears in the SIERRA scope. The string `defined_axis` uses an `axis_name` that is defined in the SIERRA scope (via a `DEFINE AXIS` command line). For this option, a radial line is drawn from a node to the radial axis. The velocity vector lies along this radial line from the node to the radial axis.

The magnitude of the velocity is specified by the `FUNCTION` command line. This references a `function_name` (defined in the SIERRA scope using a `DEFINITION FOR FUNCTION` command block) that specifies the magnitude of the velocity vector as a function of time. The magnitude can be scaled by use of the `SCALE FACTOR` command line described in Section 7.4.3.5.

7.4.3.3 User Subroutine Commands

If the user subroutine option is used, the velocity vector may vary spatially at any given time for each of the nodes in the node set associated with the particular PRESCRIBED VELOCITY command block. The user subroutine option allows for a more complex description of the velocity field than do the specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a velocity direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the velocity field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.4.3.5.

7.4.3.4 External Mesh Database Commands

If the external database option is used, the velocity vector (or specified components of the vector) is read from an external mesh database. The velocities are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the velocity:

```
READ VARIABLE = <string>var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the velocity vector from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the velocity. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.4.3.5 Additional Commands

These command lines in the PRESCRIBED VELOCITY command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the velocity in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the velocity from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.4.4 Prescribed Acceleration

```
BEGIN PRESCRIBED ACCELERATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED ACCELERATION]
```

The PRESCRIBED ACCELERATION command block prescribes an acceleration field for a given set of nodes. The acceleration field associates a vector giving the magnitude and direction of the acceleration with each node in the node set. The acceleration field may vary over time and space. If the acceleration field has only a time-varying component, the specification commands in the above command block can be used to specify the acceleration field. If the acceleration field has both time-varying and spatially varying components, a user subroutine is used to specify the acceleration field. The acceleration field can also be read from an external mesh database. In a given boundary condition command block, commands from only one of the command groups

(specification commands, user subroutine commands, or external database commands) may be used.

For rigid bodies, any kinematic boundary condition is applied directly to the rigid body reference node. The kinematics for the mesh nodes attached to the rigid body are applied via the rigid body constraint conditions. Since rigid bodies are defined using element blocks, only the BLOCK command may be used to specify prescribed acceleration conditions on rigid bodies.

The PRESCRIBED ACCELERATION command block contains four groups of commands—node set, function, user subroutine, and external database. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.4.4.1 Node Set Commands

The node set commands portion of the PRESCRIBED ACCELERATION command block defines a set of nodes associated with the prescribed acceleration field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.4.2 Specification Commands

If the specification commands are used, the acceleration vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED ACCELERATION command block. The direction of the acceleration vector is constant for all time; the magnitude of the acceleration vector may vary with time, however.

Following are the command lines used to specify the prescribed acceleration with a direction and a function:

```
DIRECTION = <string>defined_direction |
  COMPONENT = <string>X|Y|Z
FUNCTION = <string>function_name
```

The acceleration can be specified either along an arbitrary user-defined direction or along a component direction (X, Y, or Z), but not both. The acceleration is prescribed only in the specified direction. A prescribed acceleration boundary condition does not influence the motion in directions orthogonal to the prescribed direction.

- The `DIRECTION` command line is used to prescribe acceleration in an arbitrary user-defined direction. The name in the string `defined_direction` is a reference to a direction, which is defined using the `DEFINE DIRECTION` command block within the SIERRA scope.
- The `COMPONENT` command line is used to specify that the prescribed acceleration vector lies along one of the component directions. The `COMPONENT` command line is a shortcut to an internally defined direction vector; for example, component `x` corresponds to using direction vector (1, 0, 0).

The magnitude of the acceleration is specified by the `FUNCTION` command line. This references a `function_name` (defined in the SIERRA scope using a `DEFINITION FOR FUNCTION` command block) that specifies the magnitude of the acceleration vector as a function of time. The magnitude can be scaled by use of the `SCALE FACTOR` command line described in Section 7.4.4.5.

7.4.4.3 User Subroutine Commands

If the user subroutine option is used, the acceleration vector may vary spatially at any given time for each of the nodes in the node set associated with the particular `PRESCRIBED ACCELERATION` command block. The user subroutine option allows for a more complex description of the acceleration field than do the specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define an acceleration direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the acceleration field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.4.4.5.

See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

7.4.4.4 External Mesh Database Commands

If the external database option is used, the acceleration vector (or specified components of the vector) is read from an external mesh database. The accelerations are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the acceleration:

```
READ VARIABLE = <string>var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the acceleration vector from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the acceleration. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.4.4.5 Additional Commands

These command lines in the PRESCRIBED ACCELERATION command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the acceleration in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the acceleration from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.4.5 Periodic Boundary Condition

```
BEGIN PERIODIC
  SLAVE  = <string>slave_node_set
  MASTER = <string>master_node_or_face_set
  FACE CONSTRAINTS = FALSE|TRUE(FALSE)
  COMPONENT = <string>X/Y/Z
  PRESCRIBED QUANTITY = DISPLACEMENT|FORCE|VELOCITY
  SEARCH TOLERANCE = <real>tol
  FUNCTION = <string>func_name
  SCALE FACTOR = <real>func_scale
  POINT ON AXIS = <string>point_name
  REFERENCE AXIS = <string>direction_name
  THETA = <real>theta
  ACTIVE PERIODS = <string list>periods_names
  INACTIVE PERIODS = <string list>periods_names
END [PERIODIC]
```

The `PERIODIC` boundary condition command is used to define a set of periodic boundary constraints between two surfaces.

The `slave_node_set` is the name of the slave node set for the constraints. The `master_node_or_face_set` is the name of either the master node set or master surface for the constraint. If a master node set is used the master node set and slave node set must have the same number of nodes. Additionally the master node set and slave node set must be related by a simple translation. When two node sets are specified, a periodic constraint is defined between each pair of coincident (after applying the translation) nodes in the two node sets.

When a node set and a surface are specified the constraints will be defined between the the slave nodes and the faces of the master surface. When using node-face constraints `FACE CONSTRAINTS` must be set to `TRUE`. Furthermore, the slave node set and master face set must still be related to one another by a simple translation, but the number of nodes and mesh facets on the surfaces do not need to match. Better results are obtained when the more finely meshed surface is used as the slave node set and the coarser surface is used as the master surface.

`COMPONENT` specifies which directional component should be enforced as periodic. The quantity to be prescribed is defined by the `PRESCRIBED QUANTITY` command. This quantity can be `DISPLACEMENT`, `VELOCITY`, or `FORCE`.

The `SEARCH TOLERANCE` command specifies a search tolerance used for matching up nodes and faces between the master and slave sets. The search tolerance is necessary to account for slightly varying mesh topologies between the master and slave sets. The search tolerance should be something small, generally some small fraction of the characteristic face length of facets on the surfaces.

The `FUNCTION` and `SCALE FACTOR` commands are used to define a relative difference in the prescribed quantity on the slave and master sets. This function can be used to define an isotropic expansion or contraction of a periodic body. For the common use case the function should be zero meaning the prescribed quantities on the two sets are the same.

The commands POINT ON AXIS, REFERENCE AXIS, and THETA are used to define axis-symmetric periodic wedge conditions. point_name gives the name of a point defined by a domain level DEFINE POINT command. direction_name gives the name of a point defined by a domain level DEFINE DIRECTION command. Taken together these two commands define the axis of rotation for the wedge body. Theta defines the angle, in degrees, covered by the wedge.

Examples:

```
SLAVE                = nodeset_11
MASTER               = nodeset_12
COMPONENT            = xz
PRESCRIBED QUANTITY = displacement
SEARCH TOLERANCE     = 0.001
```

The above commands will create a periodic boundary condition between node sets 11 and 12. The translational offset between nodeset_11 and nodeset_12 will be automatically determined. The coincident (after the translation is applied) nodes in these node sets will be matched up. For each node pair a constraint will be created to ensure the x and z displacements of the two nodes in the pair are equal.

7.4.6 Fixed Rotation

```
BEGIN FIXED ROTATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  COMPONENT = <string>X/Y/Z | COMPONENTS = <string>X/Y/Z
  #
  # additional commands
  ACTIVE PERIODS = <string list>periods_names
  INACTIVE PERIODS = <string list>periods_names
END [FIXED ROTATION]
```

The `FIXED ROTATION` command is only applied to nodes that have rotational degrees of freedom such as shell element nodes and rigid body reference nodes. In the case of rigid bodies, the boundary condition must be specified on the block that defines the rigid body using the `BLOCK` line command.

The `FIXED ROTATION` command block fixes rotation about direction components (X, Y, Z, or some combination thereof) for a set of nodes. This command block contains two groups of commands—node set and component. Each of these command groups is basically independent of the other. In addition to the command lines in the two command groups, there are additional command lines: `ACTIVE PERIODS` and `INACTIVE PERIODS`. These command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.4.6.1 Node Set Commands

The `node set commands` portion of the command block specifies the nodes associated with the boundary condition. This portion of the command block can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.6.2 Specification Commands

There are two component specification commands available in the FIXED ROTATION command block:

```
COMPONENT = X/Y/Z | COMPONENTS = X/Y/Z
```

The rotation components that are to be fixed can be specified with either the COMPONENT command line or the COMPONENTS command line. There can be only one COMPONENT command line or one COMPONENTS command line in the command block. The user can specify any combination of the components to be fixed, as in X, Z, X Z, Y X, etc.

7.4.6.3 Additional Commands

The ACTIVE PERIODS and INACTIVE PERIODS command lines can optionally appear in the FIXED ROTATION command block:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

This command line determines when the boundary condition is active. See Section 2.5 for more information about this optional command line.

7.4.7 Prescribed Rotation

```
BEGIN PRESCRIBED ROTATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL
    <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED ROTATION]
```

Warning: The BEGIN PRESCRIBED ROTATION command is only applied to nodes that have rotational degrees of freedom such as shell element nodes and rigid body reference nodes. Displacements consistent with rotation about a fixed axis for nodes that do not have rotational degrees of freedom, are imposed using the CYLINDRICAL AXIS command line in the PRESCRIBED DISPLACEMENT command block described in Section 7.4.2.

Warning: The behavior of the BEGIN PRESCRIBED ROTATION command differs for implicit and explicit analyses. For implicit analyses, the BEGIN PRESCRIBED ROTATION command fully prescribes all rotational degrees of freedom. Consequently, a prescribed rotation in a given direction restricts rotation in the remaining two orthogonal directions. If multiple prescribed rotation boundary conditions are applied to a node, only the last one specified in the input is enforced.

In explicit analyses, the BEGIN PRESCRIBED ROTATION command block prescribes rotational degree of freedom in only the specified direction. The node is free to rotate in the directions orthogonal to that direction. Multiple prescribed rotation boundary conditions can be applied to constrain rotations in those other directions if desired.

For nodal rotational degrees of freedom, the rotations applied using the BEGIN PRESCRIBED ROTATION command are the three components of the rotational degree of freedom itself. In the case of rigid bodies, the rotations must be the nonlinear components that result in the desired quaternion on the rigid body reference node. For rigid bodies, the boundary condition is applied to the reference node and therefore must be specified on the block that defines the rigid body using the BLOCK line command.

The PRESCRIBED ROTATION command block prescribes the rotation about an axis for a given set of nodes. The rotation field associates a vector giving the magnitude and direction of the rotation with each node in the node set. The rotation field may vary over time and space. If the rotation field has only a time-varying component, the specification commands in the above command block can be used to specify the rotation field. If the rotation field has both time-varying and spatially varying components, a user subroutine is used to specify the rotation field. The rotation field can also be read from an external mesh database. In a given boundary condition command block, commands from only one of the command groups (specification commands, user subroutine commands, or external database commands) may be used.

The PRESCRIBED ROTATION command block contains four groups of commands—node set, function, user subroutine, and external database. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.4.7.1 Node Set Commands

The node set commands portion of the PRESCRIBED ROTATION command block defines a set of nodes associated with the prescribed rotation field and can include some combination of the

following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one `NODE SET`, `SURFACE`, `BLOCK`, or `INCLUDE ALL BLOCKS` command line in the command block.

7.4.7.2 Specification Commands

If the specification commands are used, the rotation vector at any given time is the same for all nodes in the node set associated with the particular `PRESCRIBED ROTATION` command block. The direction of the rotation vector is constant for all time; the magnitude of the rotation vector may vary with time, however.

Following are the command lines used to specify the prescribed rotation with a direction and a function:

```
DIRECTION = <string>defined_direction
FUNCTION = <string>function_name
```

The `DIRECTION` command line is used to prescribe rotation in an arbitrary user-defined direction. The name in the string `defined_direction` is a reference to a direction, which is defined using the `DEFINE DIRECTION` command block within the SIERRA scope. The rotation is prescribed only in the specified direction. A prescribed rotation boundary condition does not influence the rotation in directions orthogonal to the prescribed direction.

The magnitude of the rotation is specified by the `FUNCTION` command line. This references a `function_name` (defined in the SIERRA scope in a `DEFINITION FOR FUNCTION` command block) that specifies the magnitude of the rotation vector as a function of time. The magnitude can be scaled by use of the `SCALE FACTOR` command line described in Section 7.4.7.5.

The magnitude of the rotation, as specified by the product of the function and the scale factor, has units of radians per second.

7.4.7.3 User Subroutine Commands

If the user subroutine option is used, the rotation vector may vary spatially at any given time for each of the nodes in the node set associated with the particular `PRESCRIBED ROTATION` command

block. The user subroutine option allows for a more complex description of the rotation field than do the specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a rotation direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the rotation field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.4.7.5.

See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

7.4.7.4 External Mesh Database Commands

If the external database option is used, the rotation vector (or specified components of the vector) is read from an external mesh database. The rotations are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the rotation:

```
READ VARIABLE = <string>mesh_var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the rotation vector from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the rotation. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.4.7.5 Additional Commands

These command lines in the PRESCRIBED ROTATION command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the rotation in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the rotation from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.4.8 Prescribed Rotational Velocity

```
BEGIN PRESCRIBED ROTATIONAL VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
   COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>variable_name
  COPY VARIABLE = <string>variable_name [FROM MODEL
    <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED ROTATIONAL VELOCITY]
```

Warning: The PRESCRIBED ROTATIONAL VELOCITY command is only applied to nodes that have rotational degrees of freedom such as shell element nodes and rigid body reference nodes. Velocities consistent with rotation about a fixed axis for nodes that do not have rotational degrees of freedom, are imposed with the CYLINDRICAL AXIS command line in the PRESCRIBED VELOCITY command block described in Section 7.4.3.

Warning: The behavior of the BEGIN PRESCRIBED ROTATIONAL VELOCITY command differs for implicit and explicit analyses. For implicit analyses, the BEGIN PRESCRIBED ROTATIONAL VELOCITY command fully prescribes all rotational degrees of freedom. Consequently, a prescribed rotational velocity in a given direction restricts rotation in the remaining two orthogonal directions. If multiple prescribed rotational velocity boundary conditions are applied to a node, only the last one specified in the input is enforced.

In explicit analyses, the BEGIN PRESCRIBED ROTATIONAL VELOCITY command block prescribes rotational degree of freedom in only the specified direction. The node is free to rotate in the directions orthogonal to that direction. Multiple prescribed rotational velocity boundary conditions can be applied to constrain rotations in those other directions if desired.

The PRESCRIBED ROTATIONAL VELOCITY command block prescribes the rotational velocity about an axis for a given set of nodes. The rotational velocity field associates a vector giving the magnitude and direction of the rotational velocity with each node in the node set. The rotational velocity field may vary over time and space. If the rotational velocity field has only a time-varying component, the specification commands in the above command block can be used to specify the rotational velocity field. If the rotational velocity field has both time-varying and spatially varying components, a user subroutine is used to specify the rotational velocity field. The rotational velocity field can also be read from an external mesh database. In a given boundary condition command block, commands from only one of the command groups (specification commands, user subroutine commands, or external database commands) may be used.

The PRESCRIBED ROTATIONAL VELOCITY command block contains four groups of commands—node set, function, user subroutine, and external database. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.4.8.1 Node Set Commands

The node set commands portion of the PRESCRIBED ROTATIONAL VELOCITY command block defines a set of nodes associated with the prescribed rotational velocity field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
```

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.4.8.2 Specification Commands

If the specification commands are used, the rotational velocity vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED ROTATIONAL VELOCITY command block. The direction of the rotational velocity vector is constant for all time; the magnitude of the rotational velocity vector may vary with time, however.

Following are the command lines used to specify the prescribed rotational velocity with a direction and a function:

```
DIRECTION = <string>defined_direction |
  COMPONENT = <string>X|Y|Z
FUNCTION = <string>function_name
```

The rotational velocity can be specified either along an arbitrary user-defined direction or along a component direction (X, Y, or Z), but not both. The rotational velocity is prescribed only in the specified direction. A prescribed rotational velocity boundary condition does not influence the rotational velocity in directions orthogonal to the prescribed direction.

- The DIRECTION command line is used to prescribe rotational velocity in an arbitrary user-defined direction. The name in the string defined_direction is a reference to a direction, which is defined using the DEFINE DIRECTION command block within the SIERRA scope.

- The COMPONENT command line is used to specify that the prescribed rotational velocity vector lies along one of the component directions. The COMPONENT command line is a shortcut to an internally defined direction vector; for example, component x corresponds to using direction vector (1, 0, 0).

The magnitude of the rotational velocity is specified by the FUNCTION command line. This references a function_name (defined in the SIERRA scope in a DEFINITION FOR FUNCTION command block) that specifies the magnitude of the rotational velocity vector as a function of time. The magnitude can be scaled by use of the SCALE FACTOR command line described in Section 7.4.8.5.

The magnitude of the rotational velocity, as specified by the product of the function and the scale factor, has units of radians per second.

7.4.8.3 User Subroutine Commands

If the user subroutine option is used, the rotational velocity vector may vary spatially at any given time for each of the nodes in the node set associated with the particular PRESCRIBED ROTATIONAL VELOCITY command block. The user subroutine option allows for a more complex description of the rotational velocity field than do the specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a rotational velocity direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the rotational velocity field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.4.8.5.

See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

7.4.8.4 External Mesh Database Commands

If the external database option is used, the rotational velocity vector (or specified components of the vector) is read from an external mesh database. The rotational velocities are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the rotational velocity:

```
READ VARIABLE = <string>mesh_var_name
```

```
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The READ VARIABLE = <string>var_name command is used to read the rotational velocity vector from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the rotational velocity. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.4.8.5 Additional Commands

These command lines in the PRESCRIBED ROTATIONAL VELOCITY command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the rotational velocity in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the

magnitude of the rotational velocity from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The `ACTIVE PERIODS` and `INACTIVE PERIODS` command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.4.9 Reference Axis Rotation

```
BEGIN REFERENCE AXIS ROTATION
  #
  # block command
  BLOCK = <string list>block_names
  #
  # specification commands
  REFERENCE AXIS X FUNCTION = <string>function_name
  REFERENCE AXIS Y FUNCTION = <string>function_name
  REFERENCE AXIS Z FUNCTION = <string>function_name
  #
  # rotation commands
  ROTATION = <string>function_name
  ROTATIONAL VELOCITY = <string>function_name
  #
  # torque command
  TORQUE = <string>function_name
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [REFERENCE AXIS ROTATION]
```

The `REFERENCE AXIS ROTATION` command block is intended to control the rotation of a rigid body. The three `REFERENCE AXIS` line commands are required and together define a time-varying reference axis or vector. The rigid body will rotate to follow the vector. Depending on other line commands in the block, the rigid body will have either zero or one free rotational degree of freedom. If one degree of freedom is present, it is the rotation about the time-varying reference vector.

At most, one of the `ROTATION`, `ROTATIONAL VELOCITY`, or `TORQUE` line commands may be present. If the `ROTATION` or `ROTATIONAL VELOCITY` command line is present, the rigid body will be fully prescribed for rotation. The `ACTIVE PERIODS` and `INACTIVE PERIODS` command lines are used to activate or deactivate this kinematic boundary condition for certain time periods.

Use of the `REFERENCE AXIS ROTATION` command block will cause extra global variables to be written to the results file. These variables describe the reaction force, rotational reaction moment, rotational displacement, and rotational velocity associated with the boundary condition and in the local coordinate system described by the boundary condition. The names of the variables are:

```
REACTR_<NAME>
REACTS_<NAME>
REACTT_<NAME>

RREACTR_<NAME>
```

```
RREACTS_<NAME>
RREACTT_<NAME>

ROTDS_<NAME>

ROTVR_<NAME>
ROTVS_<NAME>
ROTVT_<NAME>
```

The variables beginning with REACT give the reaction forces in the boundary condition's local coordinate system. Those beginning with RREACT give the reaction moments.

Only the S component is given for the rotational displacement. This is due to the fact that the other coordinate directions are poorly defined, and therefore, rotational displacements in those directions provide no value.

The variables beginning with ROTV give the rotational velocity in the local coordinate system.

For each global variable, <NAME> is the name of the rigid body controlled by this boundary condition.

7.4.9.1 Block Command

The block command portion of the REFERENCE AXIS ROTATION command block defines a block associated with the prescribed rotation field and must include the following command line:

```
BLOCK = <string list>block_names
```

See Section 7.1.1 for more information about the use of this command line.

7.4.9.2 Specification Commands

The three specification commands are required and together define a time-varying vector that gives the orientation of a reference axis for rotation. The magnitude of the vector is ignored.

Following are the command lines used to specify the reference axis rotation with a direction:

```
REFERENCE AXIS X FUNCTION = <string>function_name
REFERENCE AXIS Y FUNCTION = <string>function_name
REFERENCE AXIS Z FUNCTION = <string>function_name
```

At a given time, the function referred to by the REFERENCE AXIS X FUNCTION line command gives the x component of a vector defining the reference axis. The pattern holds for the y and z components as well. Since the magnitude of the resulting axis or vector is not used, the three functions do not need to describe a unit vector in time.

7.4.9.3 Rotation Commands

It is possible to prescribe the complete rotation of a rigid body through the use of the ROTATION or ROTATIONAL VELOCITY line commands. Without these line commands, the rigid body is free to rotate about the reference axis.

Following are the command lines to prescribe the rotation about the reference axis:

```
ROTATION = <string>function_name
ROTATIONAL VELOCITY = <string>function_name
```

The function referred to by the ROTATION line command gives the magnitude of rotation in radians about the reference axis as a function of time. This function should not be changed across a restarted analysis.

The function referred to by the ROTATIONAL VELOCITY line command gives the rotational velocity in radians per second about the reference axis as a function of time. This function should not be changed across a restarted analysis.

7.4.9.4 Torque Command

A user may specify a moment about the reference axis through the use of the TORQUE line command.

```
TORQUE = <string>function_name
```

The function referred to by the TORQUE line command gives the torque or moment about the reference axis as a function of time.

7.4.9.5 Additional Commands

The SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS command lines can optionally appear in the REFERENCE AXIS ROTATION command block:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line applies the specified value as a scale factor to the rotation, rotational velocity, or torque functions.

The final two command lines can activate or deactivate the reference axis rotation for certain time periods. See Section 2.5 for more information about these command lines.

7.4.10 Subroutine Usage for Kinematic Boundary Conditions

The prescribed kinematic boundary conditions may be defined by a user subroutine. All these conditions use a node set subroutine. See Chapter 11 for an in-depth discussion of user subroutines. The kinematic boundary conditions will be applied to nodes. The subroutine that you write will have to return six output values per node and one output flag per node. The usage of the output values depends on the returned flag value for a node, as follows:

- If the flag value is negative, no constraint will be applied to the node.
- If the flag value is equal to zero, the constraint will be absolute. All components of the boundary condition will be specified. For example, suppose you have written a user subroutine to be called from a prescribed displacement subroutine. The prescribed displacements are to be passed through an array `output_values`. For a given node `inode`, the `output_values` array would have the following values:

```
output_values(1,inode) = displacement in x at inode
output_values(2,inode) = displacement in y at inode
output_values(3,inode) = displacement in z at inode
output_values(4,inode) = not used
output_values(5,inode) = not used
output_values(6,inode) = not used
```

- If the flag value is equal to one, the constraint will be a specified amount in a given direction. For example, suppose you have written a user subroutine to be called from a prescribed displacement subroutine. The prescribed displacements are to be passed through an array `output_values`. For a given node `inode`, the `output_values` array would have the following values:

```
output_values(1,inode) = magnitude of displacement
output(values(2,inode) = not used
output_values(3,inode) = not used
output_values(4,inode) = x component of direction vector
output_values(5,inode) = y component of direction vector
output_values(6,inode) = z component of direction vector
```

The direction in which the constraint will act is given by `output_values` 4 through 6 for `inode`. The magnitude of the displacement in the specified direction is given by `output_values` 1 at `inode`. To compute the constraint, Sierra/SM first normalizes the direction vector. Next, Sierra/SM multiplies the normalized direction vector by the magnitude of the displacement and applies the resultant constraint vector.

Displacements or velocities orthogonal to the prescribed direction will not be constrained. (This is true regardless of whether or not one uses a user subroutine for the prescribed kinematic boundary conditions.) Take the case of a prescribed displacement condition. The displacement orthogonal to a prescribed direction of motion depends on the internal and external forces orthogonal to the prescribed direction. Displacement orthogonal to the prescribed direction may or may not be zero.

7.5 Initial Velocity Conditions

```
BEGIN INITIAL VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # direction specification commands
  COMPONENT = <string>X|Y|Z |
    DIRECTION = <string>defined_direction
  MAGNITUDE = <real>magnitude_of_velocity
  #
  # angular velocity specification commands
  CYLINDRICAL AXIS = <string>defined_axis
  ANGULAR VELOCITY = <real>angular_velocity
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
END [INITIAL VELOCITY]
```

The `INITIAL VELOCITY` command block specifies an initial velocity field for a set of nodes. There are two simple options for specifying the initial velocity field: by direction and by angular velocity. The user subroutine option available is also available to specify an initial velocity. You may use only one of the available options—direction specification, angular velocity specification, or user subroutine.

The `INITIAL VELOCITY` command block contains four groups of commands—node set, direction specification, angular velocity specification, and user subroutine. Command lines associated with the node set commands must appear. As mentioned, command lines associated with one of the options must also appear. Following are descriptions of the different command groups.

7.5.1 Node Set Commands

The `node set commands` portion of the `INITIAL VELOCITY` command block defines a set of nodes associated with the initial velocity field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one `NODE SET`, `SURFACE`, `BLOCK`, or `INCLUDE ALL BLOCKS` command line in the command block.

7.5.2 Direction Specification Commands

If the direction specification commands are used, the initial velocity is applied along a defined direction with a specific magnitude. Following are the command lines for the direction option:

```
COMPONENT = <string>X|Y|Z |
  DIRECTION = <string>defined_direction
MAGNITUDE = <real>magnitude_of_velocity
```

The initial velocity can be specified either along an arbitrary user-defined direction or along a component direction (X, Y, or Z), but not both. The velocity is prescribed only in the specified direction. A prescribed velocity boundary condition does not influence the movement in directions orthogonal to the prescribed direction.

- The `DIRECTION` command line is used to prescribe initial velocity in an arbitrary user-defined direction. The name in the string `defined_direction` is a reference to a direction, which is defined using the `DEFINE DIRECTION` command block within the SIERRA scope.
- The `COMPONENT` command line is used to specify that the initial velocity vector lies along one of the component directions. The `COMPONENT` command line is a shortcut to an internally defined direction vector; for example, component `x` corresponds to using direction vector (1, 0, 0).

The magnitude of the initial velocity is given by the `MAGNITUDE` command line with the real value `magnitude_of_velocity`.

Either the `COMPONENT` command line or the `DIRECTION` command line must be specified with the `MAGNITUDE` command line if you use the direction specification commands.

7.5.3 Angular Velocity Specification Commands

If the angular velocity specification commands are used, the initial velocity is applied as an initial angular velocity about some axis. Following are the command lines for angular velocity specification:

```
CYLINDRICAL AXIS = <string>defined_axis
ANGULAR VELOCITY = <real>angular_velocity
```

The axis about which the body is initially rotating is given by the `CYLINDRICAL AXIS` command line. The string `defined_axis` uses an `axis_name` that is defined in the SIERRA scope (via a `DEFINE AXIS` command line).

The magnitude of the angular velocity about this axis is specified by the `ANGULAR VELOCITY` command line with the real value `angular_velocity`. This value is specified in units of radians per unit of time. Typically, the value for the angular velocity will be radians per second.

Both the `CYLINDRICAL AXIS` command line and the `ANGULAR VELOCITY` command line are required if you use the angular velocity specification commands.

7.5.4 User Subroutine Commands

If the user subroutine option is used, the initial velocity vector may vary spatially at any given time for each of the nodes in the node set associated with the particular `INITIAL CONDITION` command block. The user subroutine option allows for a more complex description of the initial velocity field than do the direction and angular-velocity options, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a velocity direction and a magnitude for every node to which the initial velocity field will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the initial velocity field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the `NODE SET SUBROUTINE` command line. The string `subroutine_name` is the name of a FORTRAN subroutine that is written by the user.

Following the `NODE SET SUBROUTINE` command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and

consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

See Section 7.4.10 and Chapter 11 for more details on implementing the user subroutine option.

7.6 Force Boundary Conditions

A variety of force boundary conditions are available in Sierra/SM. This section describes these boundary conditions.

7.6.1 Pressure

```
BEGIN PRESSURE
  #
  # surface set commands
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  #
  # specification commands
  FUNCTION = <string>function_name
  VELOCITY DAMPING COEFFICIENT = <real>damping_coefficient
  # user subroutine commands
  SURFACE SUBROUTINE = <string>subroutine_name |
    NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external pressure sources
  READ VARIABLE = <string>variable_name
  OBJECT TYPE = <string>NODE|FACE(NODE)
  TIME = <real>time
  FIELD VARIABLE = <string>field_variable
  #
  # output external forces from pressure
  EXTERNAL FORCE CONTRIBUTION OUTPUT NAME =
    <string>variable_name
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESSURE]
```

The PRESSURE command block applies a pressure to each face in the associated surfaces. The pressure field can be constant over the faces and vary in time, dependent on the average face velocity based on a damping coefficient, or determined by a user subroutine. If the pressure field

is constant over the faces and has only a time-varying component, the function command in the above command block can be used to specify the pressure field. For a velocity-dependent pressure, a damping coefficient is specified. If the pressure field has both time-varying and spatially varying components, user subroutine commands are used to specify the pressure field. The pressure field may also be obtained from a mesh file or from another SIERRA code through a transfer operator. You can use only one of these options—function, velocity damping coefficient, user subroutine, mesh file, transfer from another code—to specify the pressure field in a particular PRESSURE command block.

The PRESSURE command block can be used for surfaces that have faces derived from solid elements (eight-node hexahedra, four-node tetrahedra, eight-node tetrahedra, etc.), membranes, and shells. The PRESSURE command block can also be used for particle like elements if the elements are created through the use of element death particle conversion, Section 6.5.4.9. The surfaces must be defined on the original solid elements.

A pressure boundary condition generates nodal forces that are summed into the external force vector that is used to calculate the motion of a body. The external force vector contains the contribution from all forces acting on the body. There is an option in the PRESSURE command block to save information about the contribution to the external force vector due only to pressure loads. This option does not change the magnitude or time history of the pressure load (regardless of how they are defined), but merely stores information in a user-accessible variable.

The PRESSURE command block contains five groups of commands—surface set, function, user subroutine, external pressure, and output external forces. Each of these command groups is basically independent of the others. In addition to the command lines in the five command groups, there are two additional command lines: SCALE FACTOR and ACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with either the function command or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this force boundary condition for certain time periods. Following are descriptions of the different command groups.

7.6.1.1 Surface Set Commands

The surface set commands portion of the PRESSURE command block defines a set of surfaces associated with the pressure field and can include some combination of the following command lines:

```
SURFACE = <string list>surface_names
REMOVE SURFACE = <string list>surface_names
```

In the SURFACE command line, you can list a series of surfaces through the string list surface_names. There must be at least one SURFACE command line in the command block. The REMOVE SURFACE command line allows you to delete surfaces from the set specified in the SURFACE command line(s) through the string list surface_names. See Section 7.1.1 for more information about the use of these command lines for creating a set of surfaces used by the boundary condition.

7.6.1.2 Specification Commands

If the function command is used, the pressure vector at any given time is the same for all surfaces associated with the particular PRESSURE command block. The direction of the pressure vector is constant for all time; the magnitude of the pressure vector may vary with time.

Following is the command line used to specify the pressure with a function:

```
FUNCTION = <string>function_name
```

The pressure is applied in the opposite direction to the outward normals of the faces that define the surfaces. The magnitude of the pressure is specified by the FUNCTION command line. This references a function_name (defined in the SIERRA scope in a DEFINITION FOR FUNCTION command block) that specifies the magnitude of the pressure vector as a function of time. The magnitude can be scaled by use of the SCALE FACTOR command line described in Section 7.6.1.6.

A face force based on a velocity damping coefficient can be applied using the command line:

```
VELOCITY DAMPING COEFFICIENT = <real>damping_coefficient
```

In this case, nodal forces will be applied to the nodes of any faces included in the surfaces associated with the PRESSURE command block. A force vector acting normal to the face is computed by multiplying the average nodal velocity normal to the face by the damping coefficient. This face force is then evenly distributed among the nodes of the face.

7.6.1.3 User Subroutine Commands

If the user subroutine option is used, the pressure may vary spatially at any given time for each of the surfaces associated with the particular PRESSURE command block. The user subroutine option allows for a more complex description of the pressure field than does the function command, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a pressure for every face to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the pressure field.

Following are the command lines related to the user subroutine option:

```
SURFACE SUBROUTINE = <string>subroutine_name |
  NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the SURFACE SUBROUTINE command line or the NODE SET subroutine command line. The string subroutine_name in both command lines is the name of a FORTRAN subroutine that is written by the user. The particular command line selected depends on the mesh-entity type for which the pressure field is being calculated. Associating pressure values with faces would require the use of a SURFACE SUBROUTINE command line. Associating pressure values with nodes would require the use of a NODE SET SUBROUTINE command line.

Following the selected subroutine command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.6.1.6.

Usage requirements. Following are the usage requirements for the two types of subroutines:

- The surface subroutine operates on a group of faces. The subroutine that you write will return one output value per face. Suppose you write a user subroutine that returns the pressure information through an array output_value. The value output_value(1,iface) corresponds to the average pressure on face iface. The values of the flags array are not used.

- The node set subroutine that you write will return one value per node. Suppose you write a user subroutine that returns the pressure information through an array output_value. The return value output_value(1,inode) is the pressure at the node inode. The total pressure on the each face is found by integrating the pressures at the nodes. The values of the flags array are not used.

See Chapter 11 for more details on implementing the user subroutine option.

7.6.1.4 External Pressure Sources

Pressure may be obtained from two different external sources. The first option for obtaining pressure from an external source uses a mesh file. The commands for obtaining pressure information from a mesh file are as follows:

```
READ VARIABLE = <string>variable_name
OBJECT TYPE = <string>NODE|FACE(NODE)
TIME = <real>time
```

The READ VARIABLE command line specifies the name of the variable on the mesh file, variable_name, that is used to prescribe the pressure field. The OBJECT TYPE command line

specifies whether the pressure field on the mesh file is specified for nodes (the mesh object type is NODE) or for faces (the mesh object type is FACE). If the OBJECT TYPE command line is not present, it is assumed that the variable is for nodes. If the TIME command line is present, only the pressure field information at a given time, as set by the time parameter, is read from the mesh file. If the TIME command line is not present, the pressure field information for all times is read. Pressure field information will then be interpolated as necessary during an analysis.

The second option for obtaining pressure from and external sources relies on the transfer of information from another SIERRA code. The command for obtaining pressure information by transfer from another code is:

```
FIELD VARIABLE = <string>variable_name
```

Here variable_name is the name of the variable where pressure information is to be stored. The pressure information will be transferred into this variable from another SIERRA code via a transfer operator.

7.6.1.5 Output Command

This command line lets the user create a variable that stores information about the contribution to the external force vector at a node arising solely from a pressure:

```
EXTERNAL FORCE CONTRIBUTION OUTPUT NAME =
  <string>variable_name
```

If the above command line appears in a PRESSURE command block, then there will be a variable created with whatever name the user specifies for variable_name. The variable defines a three-dimensional vector at each node associated with this particular command block. The three-dimensional vector at each node represents the external force due solely to the pressure on the elements attached to that node. For example, if one of the nodes associated with this particular command block has four elements attached to it and each element has a pressure load, then the external force contribution at the node would be summed from the pressure load for all four elements.

Once this variable for the external force contribution from a pressure load is specified, it may be used like any other nodal variable. The user can, for example, specify the variable as a nodal variable to be output in a RESULTS OUTPUT command block. Or the user can reference the variable in a user subroutine.

7.6.1.6 Additional Commands

These command lines in the PRESSURE command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the function command or the user subroutine. For example, if the magnitude of the pressure in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the pressure from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.6.2 Traction

```
BEGIN TRACTION
  #
  # surface set commands
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  #
  # specification commands
  DIRECTION = <string>direction_name
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [TRACTION]
```

The TRACTION command block applies a traction to each face in the associated surfaces. The traction has units of force per unit area. (A traction, unlike a pressure, may not necessarily be in the direction of the normal to the face to which it is applied.) The given traction is integrated over the surface area of a face.

The traction field can be determined by a SIERRA function or a user subroutine. If the traction field is constant over the faces and has only a time-varying component, the specification commands in the above command block can be used to specify the traction field. If the traction field has both time-varying and spatially varying components, a user subroutine is used to specify the traction field.

The traction field can only be controlled by one method. Accordingly, a TRACTION command block can only contain one of the options: function or user subroutine.

Currently, the TRACTION command block can be used for surfaces that have faces derived from solid elements (eight-node hexahedra, four-node tetrahedra, eight-node tetrahedra, etc.), membranes, and shells.

The TRACTION command block contains three groups of commands—surface set and user subroutine. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines:

SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with the specification commands or the user subroutine option. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this force boundary condition for certain time periods. Following are descriptions of the different command groups.

7.6.2.1 Surface Set Commands

The surface set commands portion of the TRACTION command block defines a set of surfaces associated with the traction field and can include some combination of the following command lines:

```
SURFACE = <string list>surface_names
REMOVE SURFACE = <string list>surface_names
```

In the SURFACE command line, you can list a series of surfaces through the string list surface_names. There must be at least one SURFACE command line in the command block. The REMOVE SURFACE command line allows you to delete surfaces from the set specified in the SURFACE command line(s) through the string list surface_names. See Section 7.1.1 for more information about the use of these command lines for creating a set of surfaces used by the boundary condition.

7.6.2.2 Specification Commands

If the specification commands are used, the traction vector at any given time is the same for all surfaces associated with the particular TRACTION command block. The direction of the traction vector is constant for all time; the magnitude of the traction vector may vary with time, however.

Following are the command lines used to specify the traction with a direction and a function:

```
DIRECTION = <string>defined_direction
FUNCTION = <string>function_name
```

The traction is specified in an arbitrary user-defined direction, and is defined using the DIRECTION command line. The name in the string defined_direction is a reference to a direction, which is defined using the DEFINE DIRECTION command block within the SIERRA scope.

The magnitude of the traction is specified by the FUNCTION command line. This references a function_name (defined in the SIERRA scope in a DEFINITION FOR FUNCTION command block) that specifies the magnitude of the traction vector as a function of time. The magnitude can be scaled by use of the SCALE FACTOR command line described in Section 7.6.2.4.

7.6.2.3 User Subroutine Commands

If the user subroutine option is used, the traction vector may vary spatially at any given time for each of the surfaces associated with the particular `TRACTION` command block. The user subroutine option allows for a more complex description of the traction field than does the function option, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a traction for every face to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the traction field.

Following is the command line related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the `NODE SET` subroutine command line. The string `subroutine_name` is the name of a FORTRAN subroutine that is written by the user. Associating traction values with nodes requires the use of a `NODE SET SUBROUTINE` command line.

Following the `NODE SET SUBROUTINE` command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of `SUBROUTINE DEBUGGING OFF`, `SUBROUTINE DEBUGGING ON`, `SUBROUTINE REAL PARAMETER`, `SUBROUTINE INTEGER PARAMETER`, and `SUBROUTINE STRING PARAMETER`. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the `SCALE FACTOR` command line, as described in Section 7.6.2.4.

Usage requirements for the node set subroutine. The node set subroutine that you write will return six values per node. Suppose you have written a user subroutine that passes the output values through an array `output_values`. For a given node inode, the `output_values` array would have the following values:

```
output_values(1,inode) = magnitude of traction
output(values(2,inode) = not used
output_values(3,inode) = not used
output_values(4,inode) = x component of direction vector
output_values(5,inode) = y component of direction vector
output_values(6,inode) = z component of direction vector
```

The direction in which the traction will act is given by components 4 through 6 of output_values for inode. The magnitude of the traction in the specified direction is given by component 1 of output_values at inode. The total force on each node is found by integrating the local nodal tractions using the associated directions, which are normalized by Sierra/SMover the face areas. The values of the flags array are not used.

See Chapter 11 for more details on implementing the user subroutine option.

7.6.2.4 Additional Commands

These command lines in the TRACTION command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the traction in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the traction from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.6.3 Prescribed Force

```
BEGIN PRESCRIBED FORCE
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED FORCE]
```

The `PRESCRIBED FORCE` command block prescribes a force field for a given set of nodes. The force field associates a vector giving the magnitude and direction of the force with each node in the node set. The force field may vary over time and space. If the force field has only a time-varying component, the specification commands in the above command block can be used to specify the force field. If the force field has both time-varying and spatially varying components, a user subroutine is used to specify the force field. In a given boundary condition command block, commands from only one of the command groups (specification commands or user subroutine commands) may be used.

The `PRESCRIBED FORCE` command block contains three groups of commands—node set, function, and user subroutine. Each of these command groups is basically independent of the others. In addition to the command lines in the three command groups, there are three additional command lines: `SCALE FACTOR`, `ACTIVE PERIODS`, and `INACTIVE PERIODS`. The `SCALE FACTOR` com-

mand line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this force boundary condition for certain time periods. Following are descriptions of the different command groups.

7.6.3.1 Node Set Commands

The node set commands portion of the PRESCRIBED FORCE command block defines a set of nodes associated with the prescribed force field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.6.3.2 Specification Commands

If the specification commands are used, the force vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED FORCE command block. The direction of the force vector is constant for all time; the magnitude of the force vector may vary with time, however.

Following are the command lines used to specify the prescribe force with a direction and a function:

```
DIRECTION = <string>defined_direction |
  COMPONENT = <string>X|Y|Z
FUNCTION = <string>function_name
```

The force can be specified either along an arbitrary user-defined direction or along a component direction (X, Y, or Z), but not both.

- The DIRECTION command line is used to prescribe force in an arbitrary user-defined direction. The name in the string defined_direction is a reference to a direction, which is defined using the DEFINE DIRECTION command block within the SIERRA scope.

- The `COMPONENT` command line is used to specify that the force vector lies along one of the component directions. The `COMPONENT` command line is a shortcut to an internally defined direction vector; for example, component `x` corresponds to using direction vector (1, 0, 0).

The magnitude of the force is specified by the `FUNCTION` command line. This references a `function_name` (defined in the SIERRA scope in a `DEFINITION FOR FUNCTION` command block) that specifies the magnitude of the force vector as a function of time. The magnitude can be scaled by use of the `SCALE FACTOR` command line described in Section 7.6.3.4.

The force is applied only in the prescribed direction, and is not applied in any direction orthogonal to that direction.

7.6.3.3 User Subroutine Commands

If the user subroutine option is used, the force vector may vary spatially at any given time for each of the nodes in the node set associated with the particular `PRESCRIBED FORCE` command block. The user subroutine option allows for a more complex description of the force field than does the function option, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a force direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the force field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the `NODE SET SUBROUTINE` command line. The string `subroutine_name` is the name of a FORTRAN subroutine that is written by the user.

Following the `NODE SET SUBROUTINE` command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of `SUBROUTINE DEBUGGING OFF`, `SUBROUTINE DEBUGGING ON`, `SUBROUTINE REAL PARAMETER`, `SUBROUTINE INTEGER PARAMETER`, and `SUBROUTINE STRING PARAMETER`. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the `SCALE FACTOR` command line, as described in Section 7.6.3.4.

Usage requirements for the node set subroutine. The subroutine that you write will return three output values per node. Suppose you write a user subroutine that passes the output values through

an array output_values. For a given node inode, the output_values array would have the following values:

```
output_values(1,inode) = x component of force at inode
output_values(2,inode) = y component of force at inode
output_values(3,inode) = z component of force at inode
```

The three components of the force vector are given in output_values 1 through 3. The values of the flags array are ignored.

See Chapter 11 for more details on implementing the user subroutine option.

7.6.3.4 Additional Commands

These command lines in the PRESCRIBED FORCE command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the force in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the force from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.6.4 Prescribed Moment

```
BEGIN PRESCRIBED MOMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # specification commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED MOMENT]
```

The `PRESCRIBED MOMENT` command block prescribes a moment field for a given set of nodes. Moments can only be defined for nodes attached to beam or shell elements. The moment field associates a vector giving the magnitude and direction of the moment with each node in the node set. If the moment field has only a time-varying component, the specification commands in the above command block can be used to specify the moment field. If the moment field has both time-varying and spatially varying components, a user subroutine option is used to specify the moment field. In a given boundary condition command block, commands from only one of the command groups (specification commands or user subroutine commands) may be used.

The `PRESCRIBED MOMENT` command block contains three groups of commands—node set, function, and user subroutine. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: `SCALE FACTOR`, `ACTIVE PERIODS`, and `INACTIVE PERIODS`. The `SCALE FACTOR` com-

mand line can be used in conjunction with either the specification commands or the user subroutine commands. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this force boundary condition for certain time periods. Following are descriptions of the different command groups.

7.6.4.1 Node Set Commands

The node set commands portion of the PRESCRIBED MOMENT command block defines a set of nodes associated with the prescribed moment field and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

7.6.4.2 Specification Commands

If the specification commands are used, the moment vector at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED MOMENT command block. The direction of the moment vector is constant for all time; the magnitude of the moment vector may vary with time, however.

Following are the command lines used to specify the prescribed moment with a function and a direction:

```
DIRECTION = <string>defined_direction |
  COMPONENT = <string>X|Y|Z
FUNCTION = <string>function_name
```

The moment can be specified either along an arbitrary user-defined direction or along a component direction (X, Y, or Z), but not both.

- The DIRECTION command line is used to prescribe the moment in an arbitrary user-defined direction. The name in the string defined_direction is a reference to a direction, which is defined using the DEFINE DIRECTION command block within the SIERRA scope.

- The COMPONENT command line is used to specify that the moment vector lies along one of the component directions. The COMPONENT command line is a shortcut to an internally defined direction vector; for example, component x corresponds to using direction vector (1, 0, 0).

The magnitude of the moment is specified by the FUNCTION command line. This references a function_name (defined in the SIERRA scope in a DEFINITION FOR FUNCTION command block) that specifies the magnitude of the moment vector as a function of time. The magnitude can be scaled by use of the SCALE FACTOR command line described in Section 7.6.4.4.

The moment is applied only in the prescribed direction, and is not applied in any direction orthogonal to that direction.

7.6.4.3 User Subroutine Commands

If the user subroutine option is used, the moment vector may vary spatially at any given time for each of the nodes in the node set associated with the particular PRESCRIBED MOMENT command block. The user subroutine option allows for a more complex description of the moment field than do specification commands, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a moment direction and a magnitude for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the moment field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the NODE SET SUBROUTINE command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

The magnitude set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.6.4.4.

Usage requirements for the node set subroutine. The subroutine that you write will return three output values per node. Suppose you write a user subroutine that passes the output values through

an array output_values. For a given node inode, the output_values array would have the following values:

```
output_values(1,inode) = moment about x-direction at inode
output_values(2,inode) = moment about y-direction at inode
output_values(3,inode) = moment about z-direction at inode
```

The three components of the moment vector are given in output_values 1 through 3. The values of the flags array are ignored.

See Chapter 11 for more details on implementing the user subroutine option.

7.6.4.4 Additional Commands

These command lines in the PRESCRIBED MOMENT command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all vector magnitude values of the field defined by the specification commands or the user subroutine. For example, if the magnitude of the moment in a time history function is given as 1.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the magnitude of the moment from time 1.0 to 2.0 is 0.75. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.7 Gravity

```
BEGIN GRAVITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  DIRECTION = <string>defined_direction
  FUNCTION = <string>function_name
  GRAVITATIONAL CONSTANT = <real>g_constant
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [GRAVITY]
```

A gravity load is generally referred to as a body force boundary condition. A gravity load generates a force at a node that is proportional to the mass of the node. This section describes how to apply a gravity load to a body.

The `GRAVITY` command block is used to specify a gravity load that is applied to all nodes selected within a command block. The gravity load boundary condition uses the function and scale (gravitational constant and scale factor) information to generate a body force at a node based on the mass of the node. Multiple `GRAVITY` command blocks can be defined on different sets of nodes. If two different `GRAVITY` command blocks reference the same node, the node will have gravity loads applied by both of the command blocks. Care must be taken to make sure you do not apply multiple gravity loads to one block if you only want one gravity load condition applied.

The `node set commands` portion of the `GRAVITY` command block defines a set of nodes associated with the gravity load and can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list>surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes. See Section 7.1.1 for more information about the use of these command lines for

creating a set of nodes used by the boundary condition. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

The gravity load is specified along an arbitrary user-defined direction, and is defined using the DIRECTION command line. The name in the string defined_direction is a reference to a direction, which is defined using the DEFINE DIRECTION command block within the SIERRA scope.

The strength of the gravitational field can be varied with time by using the FUNCTION command line. This command line references a function_name defined in the SIERRA scope in a DEFINITION FOR FUNCTION command block.

A gravitational constant is specified by the GRAVITATIONAL CONSTANT command line in the real value g_constant. For example, the gravitational constant in units of inches and seconds would be 386.4 inches per second squared. You must set this quantity based on the actual units for your model.

The dependent variables in the function can be scaled by the real value scale_factor in the SCALE FACTOR command line. At any given time, the strength of the gravitational field is a product of the gravitational constant, the value of the function at that time, and the scale factor.

The ACTIVE PERIODS and INACTIVE PERIODS command lines provides an additional option for the gravity load condition. These command lines can activate or deactivate the gravity load for certain time periods. See Section 2.5 for more information about these command lines.

7.8 Centripetal Force

```
BEGIN CENTRIPETAL FORCE
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  CYLINDRICAL AXIS = <string>cylindrical_axis
  ROTATIONAL VELOCITY FUNCTION = <string>rotational_vel_name
  ROTATIONAL VELOCITY SCALE FACTOR = <real>rvsf(1.0)
  FORCE SCALE FACTOR = <real>fsf(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [CENTRIPETAL FORCE]
```

The centripetal force boundary condition adds an external force to the body that is equivalent to the centripetal force on the body rotating at the defined rotational velocity. The centripetal force boundary condition can be used, for example, in a static analysis to calculate the deformation of a body spinning with constant rotational velocity around some axis.

The centripetal force boundary condition will be applied to all nodes selected within the command block. The equation for the centripetal force on a node is given by Equation 7.4, where F is the magnitude of the centripetal force, m is the mass of the node, R is the current distance from the node to the rotational axis, and ω is the rotational velocity of the node about the rotational axis.

$$F = mR\omega^2 \tag{7.4}$$

The `CYLINDRICAL AXIS` command defines the name of the axis of rotation. The name given is the name of a previously defined axis created with the `DEFINE AXIS` command. The rotational velocity as a function of time is defined by multiplying the function named by the `ROTATIONAL VELOCITY FUNCTION` by the `ROTATIONAL VELOCITY SCALE FACTOR`. Additionally a scale factor on the final return force may be specified with he `FORCE SCALE FACTOR` command.

7.9 Prescribed Temperature

```
BEGIN PRESCRIBED TEMPERATURE
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  # specification command
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  TEMPERATURE TYPE = SOLID_ELEMENT|SHELL_ELEMENT (SOLID_ELEMENT)
  #
  # coupled analysis commands
  RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED TEMPERATURE]
```

The `PRESCRIBED TEMPERATURE` command block prescribes a temperature field for a given set of nodes. The prescribed temperature is for each node in the node set. The temperature field may vary over time and space. If the temperature field has only a time-varying component, the function command in the above command block can be used to specify the temperature field. If the temperature field has both time-varying and spatially varying components, a user subroutine option can be used to specify the temperature field. Finally, you may also read the temperature as a variable from the mesh file. You can select only one of these options—function, user subroutine, or read variable—in a command block.

Temperature is applied to nodes, but it is frequently used at the element level, such as in the case for thermal strains. If the temperatures are used at the element level, the nodal values are averaged (depending on element) connectivity to produce an element temperature. The temperatures must

be defined for all the nodes defining the connectivity for any given element. For this reason, we use block commands to derive a set of nodes at which to define temperatures. If the temperatures are used on an element basis, then the temperature at all the necessary nodes will be defined.

The PRESCRIBED TEMPERATURE command block contains four groups of commands—block set, function, user subroutine, and read variable. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: SCALE FACTOR, ACTIVE PERIODS, and INACTIVE PERIODS. The SCALE FACTOR command line can be used in conjunction with the function command, the user subroutine option, or the read variable option. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

7.9.1 Block Set Commands

The block set commands portion of the PRESCRIBED TEMPERATURE command block defines a set of nodes associated with the prescribed temperature field and can include some combination of the following command lines:

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes derived from some combination of element blocks. See Section 7.1.1 for more information about the use of these command lines for creating a set of nodes used by the boundary condition. There must be at least one BLOCK or INCLUDE ALL BLOCKS command line in the command block.

7.9.2 Specification Command

If the function command is used, the temperature at any given time is the same for all nodes in the node set associated with the particular PRESCRIBED TEMPERATURE command block. The command line

```
FUNCTION = <string>function_name
```

references a function_name (defined in the SIERRA scope using a DEFINITION FOR FUNCTION command block) that specifies the temperature as a function of time. The temperature can be scaled by use of the SCALE FACTOR command line described in Section 7.9.6.

7.9.3 User Subroutine Commands

If the user subroutine option is used, the temperature field may vary spatially at any given time for each of the nodes in the node set associated with the particular PRESCRIBED TEMPERATURE command block. The user subroutine option allows for a more complex description of the temperature field than does the function command, but the user subroutine option also requires that you write a user subroutine to implement this capability. The user subroutine will be used to define a temperature for every node to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the temperature field.

Following are the command lines related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Several other commands control the behavior of user subroutines: SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. These are described in Section 11.2.2. Examples of using these command lines are provided throughout Chapter 11.

The temperature set in the user subroutine can be scaled by use of the SCALE FACTOR command line, as described in Section 7.9.6.

See Chapter 11 for more details on implementing the user subroutine option.

7.9.4 External Mesh Database Commands

If the external database option is used, the temperature field is read from an external mesh database. The temperatures are read from a finite element model defined via the FINITE ELEMENT MODEL command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the USE FINITE ELEMENT MODEL command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the temperature:

```
READ VARIABLE = <string>mesh_var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
TEMPERATURE TYPE = SOLID_ELEMENT|SHELL_ELEMENT (SOLID_ELEMENT)
```

The READ VARIABLE command is used to read the temperature from the region's finite element mesh database. The var_name string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, then the COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the temperature. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

If temperature is to be prescribed for shell elements, a linear or quadratic thermal gradient can optionally be specified through the thickness of the shells using the TEMPERATURE TYPE command line. The SOLID_ELEMENT option to this command is the default behavior, and results in a single temperature being read for each node into the temperature variable. With the SHELL_ELEMENT option, temperatures are read into the shell_temperature variable. Shell elements may potentially define a temperature gradient though the thickness, in which case there will be multiple temperatures at a node to describe the temperature gradient through the shell. The TEMPERATURE_TYPE command is only valid in conjunction with the READ VARIABLE command.

Though-thickness shell temperatures follow the Aria/Calore convention. If there is a single shell temperature defined at a node, the temperature is constant through the thickness.

If there are two shell temperatures defined at a node, the first is the temperature on the bottom of the shell and the second the temperature at the top. The temperature varies linearly between the top and bottom.

If there are three shell temperatures defined at node, the first is the temperature at the bottom of the shell, the second the temperature at the middle of the shell, and the third the temperature at the top of the shell. The temperature varies quadratically through the thickness.

SOLID_ELEMENT and SHELL_ELEMENT temperatures may be defined simultaneously in the same analysis through two different temperature command blocks. If both are defined, the shell element temperature results override any solid element temperature results on the shell elements.

7.9.5 Coupled Analysis Commands

The RECEIVE FROM TRANSFER command provides the ability to set the temperature in a coupled analysis by transferring results from another SIERRA code, such as Aria.

```
RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
```

If this command is used in its default form, it is expected that the temperature is transferred to a nodal field named temperature in the Sierra/SM region. Sierra/SM performs an interpolation from the nodal field to an element field named temperature. The temperature can also be transferred directly to the element temperature field by using the optional FIELD TYPE = ELEMENT argument to this command.

If the RECEIVE FROM TRANSFER command is used, but the appropriate commands to perform the transfer between the two regions are missing, the temperature will be zero during the entire simulation.

It is also possible to use this line command to cause the initial temperatures obtained from a restart file to remain constant in time.

7.9.6 Additional Commands

These command lines in the PRESCRIBED TEMPERATURE command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all temperature values of the field defined by the function command, the user subroutine, or the read variable option. For example, if the temperature in a time history function is given as 100.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the temperature from time 1.0 to 2.0 is 50.25. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.10 Pore Pressure

```
BEGIN PORE PRESSURE
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  # specification command
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # coupled analysis commands
  RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PORE PRESSURE]
```

The `PORE PRESSURE` command block prescribes a pore pressure field for a given set of elements. The pore pressure is prescribed for each element in the block. The pore pressure field may vary over time and space. If the pore pressure field has only a time-varying component, the function command in the above command block can be used to specify the pore pressure field. If the pore pressure field has both time-varying and spatially varying components, a user subroutine option can be used to specify the pore pressure field. Finally, the pore pressure can be read as a variable from the mesh file. You can select only one of these options—function, user subroutine, or read variable—in a command block.

The `PORE PRESSURE` command block contains four groups of commands—block set, function, user subroutine, and read variable. Each of these command groups is basically independent of the others. In addition to the command lines in the four command groups, there are three additional command lines: `SCALE FACTOR`, `ACTIVE PERIODS`, and `INACTIVE PERIODS`. The

SCALE FACTOR command line can be used in conjunction with the function command, the user subroutine option, or the read variable option. The ACTIVE PERIODS and INACTIVE PERIODS command lines are used to activate or deactivate this kinematic boundary condition for certain time periods. Following are descriptions of the different command groups.

Biot's coefficient can be defined when prescribing pore pressure. See Section 5.1.2 for more information on Biot's coefficient.

7.10.1 Block Set Commands

The block set commands portion of the PORE PRESSURE command block defines a set of elements associated with the pore pressure field and can include a combination of the following command lines:

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of elements derived from some combination of element blocks. See Section 7.1.1 for more information about the use of these command lines for creating a set of elements used by the boundary condition. There must be at least one BLOCK or INCLUDE ALL BLOCKS command line in the command block.

7.10.2 Specification Command

If the FUNCTION command is used, the pore pressure at any given time is the same for all elements in the element set associated with the particular PORE PRESSURE command block. The command line

```
FUNCTION = <string>function_name
```

references a function_name (defined in the SIERRA scope using a DEFINITION FOR FUNCTION command block) that specifies the pore pressure as a function of time. The pore pressure can be scaled using the SCALE FACTOR command line described in Section 7.10.6.

7.10.3 User Subroutine Commands

If the user subroutine option is used, the pore pressure field may vary spatially at any given time for each of the elements in the element set associated with the particular PORE PRESSURE command block. The user subroutine option allows for a more complex description of the pore pressure field than does the FUNCTION command, but the user subroutine option also requires that a user

subroutine be written to implement this capability. The user subroutine will be used to define a pore pressure for every element to which the boundary condition will be applied. The subroutine will be called by Sierra/SM at the appropriate time to generate the pore pressure field.

Following are the command lines related to the user subroutine option:

```
ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

The user subroutine option is invoked by using the `ELEMENT BLOCK SUBROUTINE` command line. The string `subroutine_name` is the name of a FORTRAN subroutine that is written by the user.

Several other commands control the behavior of user subroutines: `SUBROUTINE DEBUGGING OFF`, `SUBROUTINE DEBUGGING ON`, `SUBROUTINE REAL PARAMETER`, `SUBROUTINE INTEGER PARAMETER`, and `SUBROUTINE STRING PARAMETER`. These are described in Section 11.2.2. Examples of using these command lines are provided throughout Chapter 11.

The pore pressure set in the user subroutine can be scaled by use of the `SCALE FACTOR` command line, as described in Section 7.10.6.

See Chapter 11 for more details on implementing the user subroutine option.

7.10.4 External Mesh Database Commands

The pore pressure field can be read from an external mesh database. The finite element model from which pore pressures are read is defined via the `FINITE ELEMENT MODEL` command block described in Section 6.1. The finite element model can either be the model used by the region for its mesh definition as specified with the `USE FINITE ELEMENT MODEL` command (see Section 2.3), or it can be a different (but compatible) model. The following command lines control the use of an external mesh database to prescribe the pore pressure:

```
READ VARIABLE = <string>mesh_var_name
COPY VARIABLE = <string>var_name [FROM MODEL <string>model_name]
TIME = <real>time
```

The `READ VARIABLE` command is used to read the pore pressure from the region's finite element mesh database. The `var_name` string specifies the name of the variable as it appears on the mesh database.

If the analysis will cause topology modifications due to the use of load balancing, remeshing, or other techniques that can result in nodes and/or elements being moved among processors, the

COPY VARIABLE command should be used. This command specifies that the variable named var_name will be used to specify the pore pressure. The FROM MODEL <string>model_name portion of the command is optional. If it is specified, the results are read from the mesh database named model_name. Otherwise, the region's finite element mesh database will be used as the model.

The primary difference between the behavior of the COPY VARIABLE command and the READ VARIABLE command is that the copy command handles topology modifications caused by load balancing and remeshing, which can move nodes and elements to different processors. The COPY VARIABLE command will correctly match the nodes from the external database to the corresponding nodes in the analysis and communicate the data between processors if needed. Note that the nodes in the two finite element models must match spatially, but they can be distributed on different processors.

The field to be read may be specified at an arbitrary number of different times on the mesh file. The default behavior is for the time history of the variable read from the file to be used to prescribe the boundary condition. The time history is interpolated as needed for analysis times that do not correspond exactly to times on the mesh file. The TIME command line can optionally be used to specify that the boundary condition be constant over time, based on the value of the variable read from the file at the specified time. If the specified time on the TIME command line does not correspond exactly to a time on the mesh file, the data on the mesh file will be interpolated as needed.

7.10.5 Coupled Analysis Commands

The RECEIVE FROM TRANSFER command provides the ability to set the pore pressure in a coupled analysis by transferring results from another SIERRA code, such as Aria.

```
RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
```

If this command is used in its default form, it is expected that the pore pressure is transferred to a nodal field named pore_pressure in the Sierra/SM region. Sierra/SM performs an interpolation from the nodal field to an element field named pore_pressure. The pore pressure can also be transferred directly to the element pore_pressure field by using the optional FIELD TYPE = ELEMENT argument to this command.

If the RECEIVE FROM TRANSFER command is used, but the appropriate commands to perform the transfer between the two regions are missing, the pore pressure will be zero during the entire simulation.

7.10.6 Additional Commands

These command lines in the PORE PRESSURE command block provide additional options for the boundary condition:

```
SCALE FACTOR = <real>scale_factor(1.0)
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The SCALE FACTOR command line is used to apply an additional scaling factor, which is constant in both time and space, to all pore pressure values of the field defined by the function command, the user subroutine, or the read variable option. For example, if the pore pressure in a time history function is given as 100.5 from time 1.0 to time 2.0 and the scale factor is 0.5, then the pore pressure from time 1.0 to 2.0 is 50.25. The default value for the scale factor is 1.0.

The ACTIVE PERIODS and INACTIVE PERIODS command lines determine when the boundary condition is active. See Section 2.5 for more information about these command lines.

7.11 Fluid Pressure

```
BEGIN FLUID PRESSURE
  #
  # surface set commands
  SURFACE = <string list>surface_names
  #
  # specification commands
  DENSITY = <real>fluid_density
  DENSITY FUNCTION = <string>density_function_name
  GRAVITATIONAL CONSTANT = <real>gravitational_acceleration
  FLUID SURFACE NORMAL = <string>global_component_names
  DEPTH = <real>fluid_depth
  DEPTH FUNCTION = <string>depth_function_name
  REFERENCE POINT = <string>reference_point_name
  #
  # additional commands
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESSURE]
```

The `FLUID PRESSURE` command block applies a hydrostatic pressure to each node of each face in the associated surfaces. The pressure at any node is determined from

$$P = \rho g h \tag{7.5}$$

where P is the pressure, ρ is the fluid density at the current time, g is the gravitational constant, and h is the current depth of the fluid above the node. The depth of the fluid is computed as the distance from the current fluid surface to the node in the direction of the fluid surface normal. The normal must be specified as one of the three global coordinate directions, x, y, or z. The global location of the fluid surface is found by adding the current depth to the appropriate coordinate component (the direction defined in the `FLUID SURFACE NORMAL` command) of a datum point. The datum point is either the point specified in the `REFERENCE POINT` line command, or, in the absence of this command, the minimum coordinate on the applied pressure surface in the component direction defined in the `FLUID SURFACE NORMAL` command. Once the current location of the fluid surface is computed, the depth at each node on the pressure surface is computed as the distance from the node to the fluid surface in the direction of the fluid surface normal.

Currently, the `FLUID PRESSURE` command block can be used for surfaces that have faces derived from solid elements (eight-node hexahedra, four-node tetrahedra, eight-node tetrahedra, etc.), membranes, and shells.

The `FLUID PRESSURE` command block contains three groups of commands—surface set, specification, and additional optional commands.

7.11.1 Surface Set Commands

The `surface set commands` portion of the `FLUID PRESSURE` command block defines a set of surfaces associated with the pressure field and consists of the following line:

```
SURFACE = <string list>surface_names
```

In the `SURFACE` command line, a series of surfaces can be listed through the string list `surface_names`. There must be at least one `SURFACE` command line in the command block. See Section 7.1.1 for more information about the use of command lines for creating a set of surfaces used by the boundary condition. The force computed from the hydrostatic pressure will be in the opposite direction of the face normal. When using shells or membranes, the analyst must ensure that all face normals composing the pressure application surface are in the correct direction.

7.11.2 Specification Commands

The density and the gravitational acceleration must be input by the user in units consistent with other material properties and the lengths in the mesh. To facilitate convergence of the initial load step, the gradual application of a hydrostatic load may be specified through a time history function for the density. A combination of the `DENSITY` and the `DENSITY FUNCTION` sets the value for the fluid density at each time. Either the `DENSITY` or the `DENSITY FUNCTION` must be input and both can be used together. If the `DENSITY` command is input without the `DENSITY FUNCTION` command, the density will be constant in time at that value. If the `DENSITY FUNCTION` command is input without a `DENSITY` command, the function is used as a time history of the density. If both the `DENSITY` and the `DENSITY FUNCTION` commands are input, the density value is used as a scale factor on the time history function. Finally, the `GRAVITATIONAL CONSTANT` sets the value for the acceleration due to gravity, g.

```
DENSITY = <real>fluid_density
DENSITY FUNCTION = <string>fluid_density_function
GRAVITATIONAL CONSTANT = <real>G
```

The following command lines are used to define the location of the fluid surface at any time during the analysis:

```
FLUID SURFACE NORMAL = <string>normal_component
DEPTH = <real>initial_fluid_depth
DEPTH FUNCTION = <string>depth_function_name
REFERENCE POINT = <string>point_name
```

The `FLUID SURFACE NORMAL` command sets the outward normal of the fluid surface to be one of the global component directions, x, y, or z. The fluid depth is then assumed to be in the direction opposite this global direction. The `DEPTH` command is used with the `DEPTH FUNCTION` command

to determine the fluid depth at any time. At least one of these commands must be input. If the DEPTH command is input without the DEPTH FUNCTION command, the fluid depth will be constant in time with that value. If the DEPTH FUNCTION command is input without a DEPTH command, the function is used as a time history of the depth. If both the DEPTH and the DEPTH FUNCTION commands are input, the specified depth is used as a scale factor on the time history function.

The depth and/or depth function are used to determine the current depth, which is added to the appropriate position of a datum point to compute the current location of the fluid surface in the FLUID SURFACE NORMAL component direction. The datum point is assumed to be the minimum coordinate in the component direction on the pressure surface defined in the SURFACE command if the optional REFERENCE POINT command described below is not used.

The REFERENCE POINT command line is used to specify the fluid surface relative to an external datum. When applying an external fluid pressure in a quasistatic analysis, a corresponding stiffness due to the external fluid is added to the diagonal terms of the stiffness matrix for the full tangent preconditioner to enhance convergence of the solver.

7.11.3 Additional Commands

These command lines in the FLUID PRESSURE command block provide additional options for the boundary condition:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

By default, the FLUID PRESSURE boundary condition will be active throughout an analysis. However, use of the ACTIVE PERIODS and INACTIVE PERIODS commands can be used to limit the action of the boundary condition to specific time periods. The ACTIVE PERIODS command line specifies when the boundary condition is active implying that it is inactive during any periods not included on the command line. Alternatively, the INACTIVE PERIODS determines when the boundary condition will not be active. See Section 2.5 for more information about these commands.

7.12 Specialized Boundary Conditions

Specialized boundary conditions that are provided to enforce kinematic conditions or apply loads are described in this section.

7.12.1 Cavity Expansion

```
BEGIN CAVITY EXPANSION
  EXPANSION RADIUS = <string>SPHERICAL|CYLINDRICAL
    (spherical)
  SURFACE = <string list>surface_ids
  REMOVE SURFACE = <string list>surface_ids
  FREE SURFACE = <real>top_surface_zcoord
    <real>bottom_surface_zcoord
  NODE SETS TO DEFINE BODY AXIS =
    <string>nodelist_1 <string>nodelist_id2
  TIP RADIUS = <real>tip_radius
  BEGIN LAYER <string>layer_name
    LAYER SURFACE = <real>top_layer_zcoord
      <real>bottom_layer_zcoord
    PRESSURE COEFFICIENTS = <real>c0 <real>c1 <real>c2
    SURFACE EFFECT = <string>NONE|SIMPLE_ON_OFF(NONE)
    FREE SURFACE EFFECT COEFFICIENTS = <real>coeff1
      <real>coeff2
  END [LAYER <string>layer_name]
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [CAVITY EXPANSION]
```

The CAVITY EXPANSION command block is used to apply a cavity expansion boundary condition to a surface on a body. This boundary condition is typically used for earth penetration studies where some type of projectile (penetrator) strikes a target. For a more detailed explanation of the numerical implementation of the cavity expansion boundary condition and the parameters for this boundary condition, consult Reference 1. The cavity expansion boundary condition is a complex boundary condition with several options, and the detailed explanation of the implementation of the boundary condition in the above reference is required reading to fully understand the input parameters for this boundary condition.

There are two types of cavity expansion—cylindrical expansion and spherical expansion. You can select either the spherical or cylindrical option by using the EXPANSION RADIUS command line; the default is SPHERICAL. Reference 1 describes these two types of cavity expansion.

The boundary condition is applied to the surfaces (surface_ids) in the finite element model specified by the SURFACE command line. (Any surface specified on the SURFACE command line can be removed from the list of surfaces by using a REMOVE SURFACE command line.) This boundary condition generates a pressure at a node based on the velocity and surface geometry at the node. Since cavity expansion is essentially a pressure boundary condition, cavity expansion

must be specified for a surface.

The target has a top free surface with a normal in the global positive z-direction; the target has a bottom free surface with a normal in the global negative z-direction. The point on the global z-axis intersected by the top free surface is given by the parameter `top_surface_zcoord` on the `FREE SURFACE` command line. The point on the global z-axis intersected by the bottom free surface is given by the parameter `bottom_surface_zcoord` on the `FREE SURFACE` command line.

It is necessary to define two points that lie on the axis (usually the axis of revolution) of the penetrator. These two nodes are specified with the `NODE SETS TO DEFINE BODY AXIS` command line. The first node should be a node toward the tip of the penetrator (`nodelist_1`), and the second node should be a node toward the back of the penetrator (`nodelist_2`). Only one node is allowed in each node set.

It is necessary to compute either a spherical or cylindrical radius for nodes on the surface where the cavity expansion boundary condition is applied. This is done automatically for most nodes. The calculations for these radii break down if the node is close to or at the tip of the axis of revolution of the penetrator. For nodes where the radii calculations break down, a user-defined radius can be specified with the `TIP RADIUS` command line. For more information, consult Reference 1.

Embedded within the target can be any number of layers. Each layer is defined with a `LAYER` command block. The command block begins with

```
BEGIN LAYER <string>layer_name
```

and is terminated with:

```
END [LAYER <string>layer_name]
```

Here the string `layer_name` is a user-selected name for the layer. This name must be unique to all other layer names defined in the `CAVITY EXPANSION` command blocks. The layer properties are defined by several different command lines—`LAYER SURFACE`, `PRESSURE COEFFICIENTS`, `SURFACE EFFECT`, and `FREE SURFACE EFFECT COEFFICIENTS`. These command lines are described next.

- `LAYER SURFACE = <real>top_layer_zcoord <real>bottom_layer_zcoord`

  The layer has a top surface with a normal in the global positive z-direction; the layer has a bottom surface with a normal in the global negative z-direction. In the `LAYER SURFACE` command line, the point on the global z-axis intersected by the top layer surface is given by the parameter `top_layer_zcoord`, and the point on the global z-axis intersected by the bottom layer surface is given by the parameter `bottom_layer_zcoord`.

- `PRESSURE COEFFICIENTS = <real>c0 <real>c1 <real>c2`

  The value of the pressure at a node is derived from an equation that is quadratic based on some scalar value derived from the velocity vector at the node. The three coefficients for the

quadratic equation (c0, c1, c2) in the PRESSURE COEFFICIENTS command line define the impact properties of a layer.

- SURFACE EFFECT = <string>NONE|SIMPLE_ON\str (NONE)

  There can be no surface effects associated with a layer, or there can be a simple on/off surface effect model associated with a layer. The type of surface effect is determined by the SURFACE EFFECT command line. The default is no surface effects. If the SIMPLE_ON_OFF model is chosen, it is necessary to specify free surface effect coefficients with the FREE SURFACE EFFECT COEFFICIENTS command line.

- FREE SURFACE EFFECT COEFFICIENTS = <real>coeff1 <real>coeff2

  All the parameters defined in a LAYER command block apply to that layer. If a simple on/off surface effect is applied to a layer, the surface effect coefficients are associated with the layer values. The surface effect parameter associated with the top of the layer is coeff1; the surface effect parameter associated with the bottom of the layer is coeff2.

The ACTIVE PERIODS and INACTIVE PERIODS command lines provide an additional option for cavity expansion. These command lines can activate or deactivate cavity expansion for certain time periods. See Section 2.5 for more information about these command lines.

7.12.2 Silent Boundary

```
BEGIN SILENT BOUNDARY
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [SILENT BOUNDARY]
```

The `SILENT BOUNDARY` command block is also referred to as a non-reflecting surface boundary condition. A wave striking this surface is not reflected. This boundary condition is implemented with the techniques described in Reference 2. The method described in this reference is excellent at transmitting the low- and medium-frequency content through the boundary. While the method does reflect some of the high-frequency content, the amount of energy reflected is usually minimal. On the whole, the silent boundary condition implemented in Presto is highly effective.

In the `SURFACE` command line, you can list a series of surfaces through the string list `surface_names`. There must be at least one `SURFACE` command line in the command block. The `REMOVE SURFACE` command line allows you to delete surfaces from the set specified in the `SURFACE` command line(s) through the string list `surface_names`. See Section 7.1.1 for more information about the use of these command lines for creating a set of surfaces used by the boundary condition.

The `ACTIVE PERIODS` and `INACTIVE PERIODS` command lines provide an additional option for the boundary condition. These command lines are used to activate or deactivate the boundary condition for certain time periods. See Section 2.5 for more information about these command lines.

7.12.3 Spot Weld

```
BEGIN SPOT WELD
  NODE SET = <string list>nodelist_ids
  REMOVE NODE SET = <string list>nodelist_ids
  SURFACE = <string list>surface_ids
  REMOVE SURFACE = <string list>surface_ids
  SECOND SURFACE = <string>surface_id
  NORMAL DISPLACEMENT FUNCTION =
    <string>function_nor_disp
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_nor_disp(1.0)
  TANGENTIAL DISPLACEMENT FUNCTION =
    <string>function_tang_disp
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_tang_disp(1.0)
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE FUNCTION = <string>fail_func_name
  FAILURE DECAY CYCLES = <integer>number_decay_cycles(10)
  SEARCH TOLERANCE = <real>search_tolerance
  IGNORE INITIAL OFFSET = NO|YES(NO)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [SPOT WELD]
```

The spot weld option lets the user model an “attachment” between a node on one surface and a face on another surface. This option models a weld or a small screw or bolt with a normal force-displacement curve like that shown in Figure 7.1 and a tangential force-displacement curve like that shown in Figure 7.2. The displacement shown in the figures is the distance, either normal or tangential, that the node moves from the nearest point on the face as measured in the original configuration. The force shown in the figure is the force at the attachment as a function of the distance between the two attachment points. (The force-displacement curve assumes the two attachment points are originally at the same location and the initial distance is zero, thus zero force at time zero. However, in most situations there is an initial gap which leads to non-zero forces at time zero.) Two force-displacement curves are required for the spot weld model; one curve models normal behavior, and the other curve models tangential behavior. It is worth noting the difference in how the normal and tangential components behave, therefore explaining why two curves are required. It is possible for the node to interpenetrate the surface resulting in a negative normal displacement between the connection points. Therefore, the normal force displacement curve must have the ability to capture negative displacements and thus negative forces. Although this penalty stiffness approach will work to prevent interpenetration, it is better to model this behavior using contact. The tangential displacement, however, is always positive.

The sign of the normal displacement of the spot weld (generally tension at positive displacement or compression at negative displacement) is determined by the relative motion of the attached node and face as well as the normal direction of the face. See Figure 7.3. If the relative normal motion of

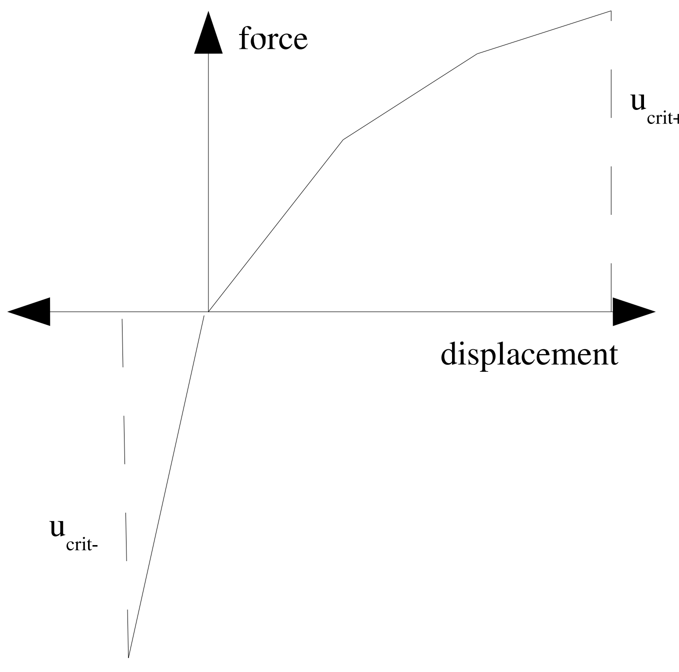


Figure 7.1: Force-displacement curve for spot weld normal force.

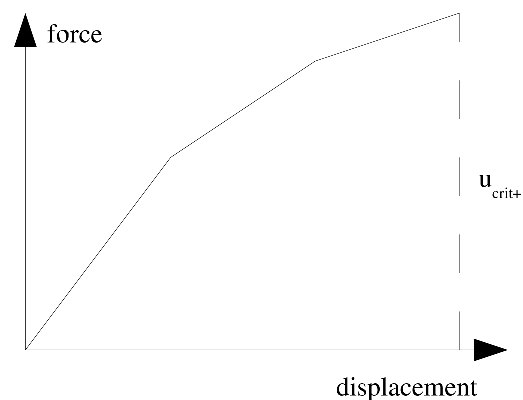


Figure 7.2: Force-displacement curve for spot weld tangential force.

the face is along the face normal the spot weld becomes more compressive. If the relative normal motion of the face is opposite to the face normal the spot weld becomes more tensile.

The attachment in Presto is defined between a node on one surface and the closest point on an element face on the other surface. Since a face is used to define one of the attachment points, it is possible to compute a normal vector and a tangent vector associated with the face. This

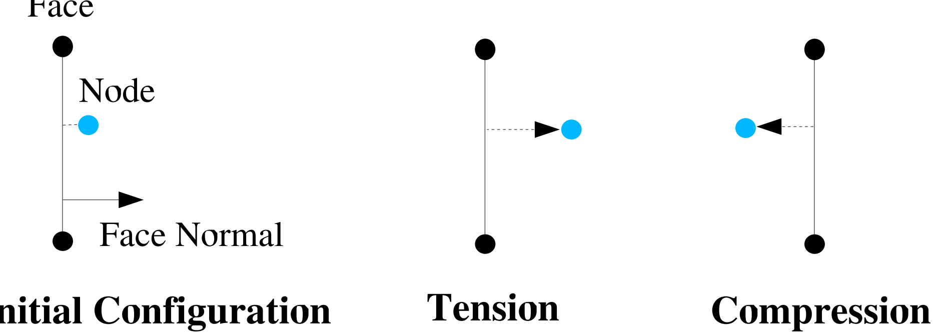


Figure 7.3: Sign convention for spot weld normal displacements.

allows us to resolve the displacement (distance, both positive and negative) and force (both tensile and compressive) into normal and tangential components. With normal and tangential vectors associated with the attachment, the attachment can be characterized for the case of pure tension and pure shear.

Presto includes two mechanisms for determining failure for cases that fall between pure tension and pure shear. In the first case, failure is governed by the equation

$$(u_n/u_{n_{crit}})^p + (u_t/u_{t_{crit}})^p < 1.0\,. \tag{7.6}$$

In Equation 7.6, the distance from the node to the original attachment point on the face as measured normal to the face is u_n, which is defined as the normal distance. The maximum value given for u_n in the normal force-displacement curve is $u_{n_{crit}}$, but is different for positive and negative displacements. In Figure 7.1, the value used for $u_{n_{crit}}$ is u_{crit+} in the positive direction and u_{crit-} in the negative direction. The distance from the node to the original attachment point on the face as measured along a tangent to the face is u_t, which is defined as the tangential distance. The maximum value given for u_t in the tangential force-displacement curve is $u_{t_{crit}}$. The value p is a user-specified exponent that controls the shape of the failure surface.

Alternatively, Presto permits a user-specified function to determine the failure surface. The function defines the ratio of $u_t/u_{t_{crit}}$ at which failure will occur as a function of $u_n/u_{n_{crit}}$. The function must range from 0.0 to 1.0, and have a value of 1.0 at 0.0 and a value of 0.0 at 1.0. These restrictions preserve proper failure for the cases of pure tension and pure shear.

To use the spot weld option in Presto, a `SPOT WELD` command block begins with the input line:

```
BEGIN SPOT WELD
```

and is terminated with the input line :

```
END [SPOT WELD]
```

Within the command block, it is necessary to specify the set of nodes on one side of the spot weld with the NODE SET command line. The NODE SET command line can list one or more node sets. Any node set listed on the NODE SET command line can be deleted from the list of node sets by using a REMOVE NODE SET command line. A set of element faces on an opposing side of the spot weld (which we will refer to as the *first* surface) is specified with the SURFACE command line. The SURFACE command line can list one or more surfaces. Any surface listed on the SURFACE command line can be deleted from the list of surface by using a REMOVE SURFACE command line. For any node in the node set, the closest point to this node on the opposing surface should lie within the element faces specified by the SURFACE command line.

The normal force-displacement curve is specified by a function named by the value function_nor_disp in the NORMAL DISPLACEMENT FUNCTION command line. This function can be scaled by the real value scale_nor_disp in the NORMAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0. The last points in the positive and negative directions are used as the displacements beyond which the spot weld fails. The tangential force-displacement curve is specified by a function named by the string function_tang_disp in the TANGENTIAL DISPLACEMENT FUNCTION command line. This function can be scaled by the real value scale_tang_disp given in the TANGENTIAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0. The last point in the positive direction in the tangential curve is used as the displacement beyond which the spot weld fails. The scaling factors for both the normal and tangential force-displacement curves apply only to the ordinates, or force values of the curves.

The failure surface between pure tension and pure shear is controlled by specifying either the failure envelope exponent, p in Equation 7.6, or a failure function. The failure exponent is specified by the real value exponent in the FAILURE ENVELOPE EXPONENT command line. The failure function is specified by the FAILURE FUNCTION command line. If both a failure function and a failure exponent are given, then the failure function is used.

For an explicit, transient dynamics code like Presto, it is better to remove the force for the spot weld over several load steps rather than over a single load step once the failure criterion is exceeded. The FAILURE DECAY CYCLES command line controls the number of load steps over which the final force is removed (default value is 10). To remove the final force at a spot weld over five load increments, the integer specified by number_decay_cycles would be set to 5. Once the force at the spot weld is reduced to zero, it remains zero for all subsequent time (despite the function definition).

The spot weld can take on area-based behavior by specifying a surface in place of a set of nodes. The identifier of this surface is specified by the string surface_id in the SECOND SURFACE command line. The area-based spot weld creates a weld between all nodes on the second surface and the faces of the first surface. The load-resistance curve at each node is derived from the tributary area of the node times the given force-displacement curves. Thus, for the area-based spot welds, the force-displacement curves give the force per unit area resisted by the weld.

The user must set a tolerance for the node-to-face search with the

```
SEARCH TOLERANCE = <real>search_tolerance
```

command line. The value you select for search_tolerance will depend upon the distance between the nodes and surfaces used to define the spot weld.

If the user sets IGNORE INITIAL OFFSET = YES, the initial distance between the node and the face will be taken as the zero normal displacement distance for the spot weld. When using the IGNORE INITIAL OFFSET = YES command the initial normal force in the will be the value of the force-displacement curve at displacement=zero. The sign convention used to determine if a spot weld is in compression or tension is the same as without using IGNORE INITIAL OFFSET = YES as shown Figure 7.4.

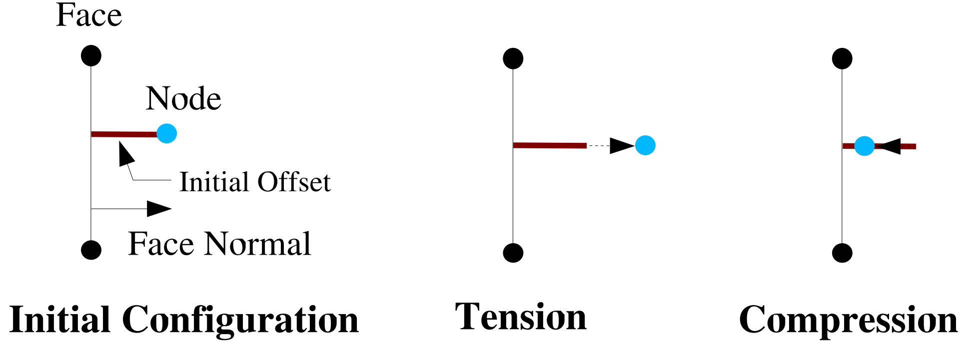


Figure 7.4: Sign convention for spot weld normal displacements with ignore initial offsets on.

The ACTIVE PERIODS and INACTIVE PERIODS command lines provide an additional option for the boundary condition. These command lines are used to activate or deactivate the boundary condition for certain time periods. See Section 2.5 for more information about these command lines.

In explicit dynamics, spot welds do not inherently have a critical time step due to the fact that they have zero mass. However, when a node-based time step estimate is used, spot welds contribute their stiffness to the nodes associated with the spot weld, thus influencing the time step calculated based on those nodes. The spot weld stiffness is calculated as the sum of the current derivative of the normal force function and the current derivative of the tangential force function, which is a conservative estimate. If the nodes attached to an element that governs the critical time step have spot weld stiffness contributions, a line is output to the log file stating the approximate percent reduction in critical time step due to the the spot weld stiffness contribution.

Output data can be obtained from spot welds. The list of available output variables is documented in Table 9.9.1.

Explicit Only

7.12.4 Line Weld

```
BEGIN LINE WELD
  SURFACE = <string list> surface_names
  REMOVE SURFACE = <string list> surface_names
  BLOCK = <string list> block_names
  REMOVE BLOCK = <string list>block_names
  SEARCH TOLERANCE = <real>search_tolerance
  R DISPLACEMENT FUNCTION = <string>r_disp_function_name
  R DISPLACEMENT SCALE FACTOR = <real>r_disp_scale
  S DISPLACEMENT FUNCTION = <string>s_disp_function_name
  S DISPLACEMENT SCALE FACTOR = <real>s_disp_scale
  T DISPLACEMENT FUNCTION = <string>t_disp_function_name
  T DISPLACEMENT SCALE FACTOR = <real>t_disp_scale
  R ROTATION FUNCTION = <string>r_rotation_function_name
  R ROTATION SCALE FACTOR = <real>r_rotation_scale
  S ROTATION FUNCTION = <string>s_rotation_function_name
  S ROTATION SCALE FACTOR = <real>s_rotation_scale
  T ROTATION FUNCTION = <string>t_rotation_function_name
  T ROTATION SCALE FACTOR = <real>t_rotation_scale
  FAILURE ENVELOPE EXPONENT = <real>k(2.0)
  FAILURE DECAY CYCLES = <integer>number_decay_cycles
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END LINE WELD
```

The line-weld capability is used to weld the edge of a shell to the face of another shell. The bond can transmit both translational and rotational forces. When failure of the line weld occurs, it breaks and no longer transmits any forces.

The edge of the shell that is tied to a surface is modeled with a block of one-dimensional elements (truss, beam, spring, etc.). The edge of the shell and the one-dimensional elements will share the same nodes. We will refer to the shell edge and the one-dimensional elements associated with it as the one-dimensional part of the line-weld model. The element blocks with the one-dimensional elements are specified by using the `BLOCK` command line. More than one element block can be listed on this command line. The element blocks referenced by the `BLOCK` command line must be one-dimensional elements—truss, beam, spring, etc.

The other part of the line weld is a set of faces defined by shell elements; this set of faces is the two-dimensional part of the line weld. The surface (the two-dimensional part of the model) to which the nodes (from the one-dimensional part of the model) are to be bonded is defined by any surface of element faces derived from shell elements. The line weld will bond each node in the element blocks listed in the `BLOCK` command line to the closest face (or faces) of element faces in the surfaces listed in the `SURFACE` command line. More than one surface can be listed on this command line.

The command line `SEARCH TOLERANCE` sets a tolerance on the search for node-to-face interac-

tions. For a given node, only those faces within the distance set by the search_tolerance parameter will be searched to determine whether the node should be welded to the face.

Each section of the line weld has its own local coordinate system (r, s, t). The r-direction lies along a one-dimensional element (and hence on the surface). The s-direction lies on the surface and is tangential to the one-dimensional element. The t-direction lies normal to the face and is orthogonal to the r- and s-directions. Force-displacement functions and moment-rotation functions may be specified for all axes in the local coordinate system. If one of the functions is left out, the resistance is zero for that axis. These functions are similar to the ones used for the spot weld (see Figure 7.2).

The force-displacement function in the r-direction represents shear resistance in the direction of the weld; this function is specified by a SIERRA function name on the R DISPLACEMENT FUNCTION command line. The force-displacement in the s-direction represents shear resistance tangential to the weld; this function is specified by a SIERRA function name on the S DISPLACEMENT FUNCTION command line. The force-displacement in the t-direction function represents tearing resistance normal to the surface; this function is specified by a SIERRA function name on the T DISPLACEMENT FUNCTION command line. The moment-rotation about the r-axis represents a rotational tearing resistance; this is specified by a SIERRA function name on the R ROTATION FUNCTION command line. The rotational resistances about the s-direction and the t-direction are likely not very meaningful, as rotations along these axes should be well constrained by the normal and tangential displacement relations. These two rotational resistances, if used, are defined with SIERRA function names on the S ROTATION FUNCTION and T ROTATION FUNCTION command lines. Note that each SIERRA function used in this command block is defined via a DEFINITION FOR FUNCTION command block in the SIERRA scope.

Any of the above functions can be scaled by using a corresponding scale factor. For example, the force-displacement function on the R DISPLACEMENT FUNCTION command line can be scaled by the parameter r_disp_scale on the R DISPLACEMENT SCALE FACTOR command line. Only the force values of the force-displacement curve will be scaled.

The failure function for the line weld is similar to that for the spot weld. Denote the displacement or rotation associated with a line weld as δ. Suppose that δ_i is a displacement in the r-direction. The force-displacement curve specified on the R DISPLACEMENT FUNCTION command line has a maximum value η. This is the maximum displacement the weld can endure in the r-direction before breaking. Associate this value of η with δ_i by designating it as η_i. Repeat this pairing process for all the displacements and rotations defining the line weld. Each displacement component in the line weld will be paired with one of the three maximum displacement values associated with the line weld. Each rotation component in the line weld will be paired with one of the three maximum rotation values associated with the line weld. Breaking of the weld under combined loading is calculated the same as the spot weld. The weld breaks under the following condition:

$$\sqrt[k]{\sum \left(\frac{\delta_i}{\eta_i}\right)^k} > 1 \,. \tag{7.7}$$

In the above equation, the parameter k is set by the user. A typical value for k is 2. The summation takes place over all the failure functions (force-displacement and moment-rotation) for all the

nodes. (The value for *k* is specified on the FAILURE ENVELOPE EXPONENT command line.)

For an explicit, transient dynamics code like Presto, it is better to remove the forces for the line weld over several time steps rather than over a single time step once the failure criterion is exceeded. The FAILURE DECAY CYCLES command line controls the number of time steps over which the final force is removed. To remove the final force at a line weld over five time steps, the integer specified by number_decay_cycles would be set to 5. Once the force in the line weld is reduced to zero, it remains zero for all subsequent time.

The ACTIVE PERIODS and INACTIVE PERIODS command lines provide an additional option for the boundary condition. These command lines are used to activate or deactivate the boundary condition for certain time periods. See Section 2.5 for more information about these command lines.

7.12.5 Viscous Damping

```
BEGIN VISCOUS DAMPING <string>damp_name
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  MASS DAMPING COEFFICIENT = <real>mass_damping
  STIFFNESS DAMPING COEFFICIENT = <real>stiff_damping
  #
  # additional command
  ACTIVE PERIODS = <string list>period names
  INACTIVE PERIODS = <string list>period_names
END [VISCOUS DAMPING <string>damp_name]
```

The `VISCOUS DAMPING` command block adds simple Rayleigh viscous damping to mesh nodes. At each node, Presto computes a damping coefficient, which is then multiplied by the node velocity to create a damping force. The damping coefficient is the sum of the mass times a mass damping coefficient and the nodal stiffness times a stiffness damping coefficient. In general, the mass damping portion damps out low-frequency modes in the mesh, while the stiffness damping portion damps out higher-frequency terms. Appropriate values for the damping coefficients depend on the frequencies of interest in the mesh. The general expression for the critical damping fraction, c_d, for a given frequency is

$$c_d = (k_d * \omega + m_d/\omega)/2\,, \tag{7.8}$$

where k_d is the stiffness damping coefficient, m_d is the mass damping coefficient, and ω is the frequency of interest. The stiffness damping portion must be used with caution. Because this term depends on the stiffness, it can affect the critical time step. Thus certain ranges of values for the stiffness damping coefficient can change the critical time step for the mesh. As Presto does not currently modify the critical time step based on the selected values for this coefficient, some choices for this parameter can cause solution instability.

7.12.5.1 Block Set Commands

The `block set commands` portion of the `VISCOUS DAMPING` command block defines a set of element blocks associated with the viscous damping and can include some combination of the following command lines:

```
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of element blocks. See Section 7.1.1 for more information about the use of these command

lines for creating a set of element blocks used by viscous damping. There must be at least one BLOCK or INCLUDE ALL BLOCKS command line in the command block.

All the nodes associated with the elements specified by the block set commands will have viscous damping forces applied.

7.12.5.2 Viscous Damping Coefficient

The mass damping coefficient, m_d, in Equation 7.8 is specified using the parameter mass_damping on the command line:

```
MASS DAMPING COEFFICIENT = <real>mass_damping
```

Mass damping most strongly damps the low-frequency modes, and in Presto, is applied as a boundary condition which does not affect the critical time step.

The stiffness damping coefficient, k_d, in Equation 7.8, is specified as the parameter stiff_damping on the command line:

```
STIFFNESS DAMPING COEFFICIENT = <real>stiff_damping
```

Stiffness damping most strongly damps high-frequency modes and is applied through a stress correction of the pressure term at the element level. Because this type of damping is done at the element level, it has a direct effect on the critical time step as seen in Equation 7.9, where $\epsilon = \frac{k_d \omega}{2}$ in the absence of mass damping and $\frac{2}{\omega} \approx \frac{m}{\kappa}$.

$$\Delta t \leq \frac{2}{\omega} \left(\sqrt{1 + \epsilon^2} - \epsilon \right) \tag{7.9}$$

7.12.5.3 Additional Command

The ACTIVE PERIODS and INACTIVE PERIODS command lines can optionally appear in the VISCOUS DAMPING command block:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

This command line can activate or deactivate the viscous damping for certain time periods. See Section 2.5 for more information about this command line.

7.12.6 Volume Repulsion Old

```
BEGIN VOLUME REPULSION OLD <string>repulsion
  FRICTION COEFFICIENT = <real>fric_coeff
  SCALE FACTOR = <real>scale_factor
  OVERLAP TYPE = [NODAL|VOLUMETRIC]

  BEGIN BLOCK SET <string>set
    BLOCK = <string list>block_names
    INCLUDE ALL BLOCKS
    REMOVE BLOCK = <string list>block_names
    #
    SURFACE = <string list>surface_names
    REMOVE SURFACE = <string list>surface_names
    #
    ACTIVE PERIODS = <string list>period_names
    INACTIVE PERIODS = <string list>period_names
    #
    LINE CYLINDER RADIUS = <real>cylinder_radius
    ELEMENT REPRESENTATION = [BEAM_ELEMENT_CYLINDERS|
      TRUE_SOLID_VOLUME|NODES]
  END [BLOCK SET <string>set]
END [VOLUME REPULSION OLD <string>repulsion]
```

The `VOLUME REPULSION OLD` command block is used to create a cylindrical volume around beam elements that is used as a frictional contact surface. Because beam elements are represented as line elements, there is no volume associated with the elements that can be used for contact. This command will generate nodal forces based on interpenetrations of a prescribed set of nodes with the cylindrical contact surfaces. The force is generated using a penalty stiffness method where the magnitude of the force depends on the mass of the node and the current time step. This command will be replaced by Dash contact in the future.

Definition of the coefficient of friction for the cylindrical surface is defined through the `FRICTION COEFFICIENT` command line.

The `SCALE FACTOR` command line specifies a scale factor that scales the force produced from node-surface interactions.

The `OVERLAP TYPE` command line must always be set to `VOLUMETRIC`.

7.12.6.1 Block Set

Complete definition of the contact surfaces and node sets are completely defined within multiple `BLOCK SET` command blocks. At least two `BLOCK SET` command blocks must be defined, one that defines the beam element blocks to wrap in cylinders by setting the `ELEMENT REPRESENTATION` command line to `BEAM_ELEMENT_CYLINDERS`. The second `BLOCK SET`

command block defines the node set used to contact the cylindrical surfaces by setting the ELEMENT REPRESENTATION to NODES.

The BLOCK, INCLUDE ALL BLOCKS and REMOVE BLOCK command lines must be used when defining the beam element blocks, while a combination of these block command lines and the following surface command lines, SURFACE and REMOVE SURFACE, must be used to define the node set used for contact.

ACTIVE PERIODScommand line defined the time periods in which this boundary condition is active, whereas the INACTIVE PERIODS command line defines when this boundary condition is inactive.

The LINE CYLINDER RADIUS command line specifies the radius of the cylindrical surface around the beam elements.

7.12.7 General Multi-Point Constraints

```
BEGIN MPC
  #
  # Master/Slave MPC commands
  MASTER NODE SET = <string list>master_nset
  MASTER NODES = <integer list>master_nodes
  MASTER SURFACE = <string list>master_surf
  MASTER BLOCK = <string list>master_block
  SLAVE NODE SET = <string list>slave_nset
  SLAVE NODES = <integer list>slave_nodes
  SLAVE SURFACE = <string list>slave_surf
  SLAVE BLOCK = <string list>slave_block
  #
  # Tied contact search commands
  SEARCH TOLERANCE = <real>tolerance
  VOLUMETRIC SEARCH TOLERANCE = <real>vtolerance
  #
  # Tied MPC commands
  TIED NODES = <integer list>tied_nodes
  TIED NODE SET = <string list>tied_nset
  #
  # DOF subset selection
  COMPONENTS = <enum>X|Y|Z|RX|RY|RZ
  REMOVE GAPS AND OVERLAPS = OFF|ON(OFF)
END [MPC]

# Control handling of multiple MPCs
RESOLVE MULTIPLE MPCS = ERROR|FIRST WINS|LAST WINS(ERROR)
```

Sierra/SM provides a general multi-point constraint (MPC) capability that allows a code user to specify arbitrary constraints between sets of nodes. The commands to define a MPC are all listed within a `MPC` command block. There are three types of MPCs: master/slave, tied contact, and tied. All of these types of MPCs are defined within the `MPC` command block, but different commands are used within that block for each case. The commands for each of these types of MPCs are described in detail below.

MPCs can potentially be in conflict with other constraints. Refer to Appendix E for information on how conflicting constraints are handled.

7.12.7.1 Master/Slave Multi-Point Constraints

The master/slave type of MPC imposes a constraint between a set of master nodes and a set of slave nodes. The motion of the three translational degrees of freedom of the slave nodes is constrained to be equal to the average motion of the master nodes. This type of MPC is typically most useful if there is either a single master node and one or more slave nodes, or multiple master nodes and a single slave node. If there are multiple slave nodes, they are constrained to move together as a set.

The sets of master and slave nodes used in the MPC can be defined by using a node set on the mesh file, a list of nodes provided in the input file, or a surface on the mesh file from which a list of nodes is extracted. This can be done for the set of master nodes using one or more of the following commands:

```
MASTER NODE SET = <string list>master_nset
MASTER NODES = <integer list>master_nodes
MASTER SURFACE = <string list>master_surf
MASTER BLOCK = <string list>master_block
SLAVE NODE SET = <string list>slave_nset
SLAVE NODES = <integer list>slave_nodes
SLAVE SURFACE = <string list>slave_surf
SLAVE BLOCK = <string list>slave_block
```

The `MASTER NODE SET` and `SLAVE NODE SET` command lines specify the names of node sets in the mesh file. The nodes in these node sets are included in the sets of master or slave nodes for the constraint.

The `MASTER NODES` and `SLAVE NODES` command lines specify lists of integer IDs of nodes to be included in the sets of master or slave nodes for the constraint.

The `MASTER SURFACE` and `SLAVE SURFACE` command lines specify the name of a surface in the mesh file. The nodes contained in this surface are included in the sets of master or slave nodes for the constraint.

The `MASTER BLOCK` and `SLAVE BLOCK` command lines specify the names of a block in the mesh file. The nodes contained in these blocks are included in the sets of master or slave nodes for the constraint.

7.12.7.2 Tied Contact

A proximity search can optionally be performed to create a set of MPCs that act as tied contact constraints. If the MPCs are created in this way, the search is performed at the time of initialization to find pairings of slave nodes to master faces. A separate constraint is created for each slave node. This is equivalent to using pure master/slave tied contact, and is significantly faster than standard tied contact for explicit dynamics analyses.

```
SEARCH TOLERANCE = <real> tolerance
```

The `SEARCH TOLERANCE` command line is used to request that a search be performed to create node/face constraints. This line must be present to use MPCs for tied contact. The tolerance value given on the line specifies the maximum distance between a node and a face to create an MPC. This has a similar meaning to the search tolerance used in standard tied contact.

Warning: The `SEARCH TOLERANCE` command line must be present to use MPCs for tied contact. If this command is not present in the `MPC` command block, a master/slave MPC as described in Section 7.12.7.1 will result. All slave nodes would be tied to all master nodes, which is very different from tied contact.

```
VOLUMETRIC SEARCH TOLERANCE = <real> vtolerance
```

In addition to creating node/face constraints, a search can be used to create volumetric constraints, in which a slave node is constrained to a volume bounded by master faces. The VOLUMETRIC SEARCH TOLERANCE command is used to enable volumetric constraints and sets the tolerance used for the search. The slave node is constrained to all nodes on the master surface that are within the volumetric search tolerance, vtolerance.

Currently the primary usage of volumetric constraints is to constrain a meshed void placed inside another mesh. The volumetric search is used in conjunction with the node/face search. A set of node/face constraints is first created for slave nodes within the standard search tolerance of any master face. Next, a volumetric constraint is created for any remaining unconstrained slave node within the volumetric search tolerance of the node/face constrained nodes. The volumetric constraint is formulated in a way that approximates an isoparametric map.

To use MPCs for tied contact, the master and slave surfaces must be defined. These may be defined using the MASTER SURFACE, MASTER BLOCK, SLAVE NODE SET, SLAVE SURFACE, and SLAVE BLOCK line commands. These are a subset of the commands available to define master/slave MPCs, as described in Section 7.12.7.1. It is important to note that the MASTER NODE SET can not be used to use MPCs for tied contact. The master surface must have information about faces, and this is not available with a node set.

The following example demonstrates how to use MPCs for tied contact between two surfaces:

```
BEGIN MPC
  MASTER SURFACE = surface_10
  SLAVE SURFACE = surface_11
  SEARCH TOLERANCE = 0.0001
END MPC
```

The following example demonstrates the usage of MPCs for both tied and volumetric constraints. Assuming that surface_10 surrounds the exterior surface of block_1, this block would result in tied contact between the nodes on the exterior of block_1 and surface_10, and volumetric constraints between the nodes on the interior of block_1 and surface_1.

```
BEGIN MPC
  MASTER SURFACE = surface_10
  SLAVE BLOCK = block_1
  SEARCH TOLERANCE = 0.0001
  VOLUMETRIC SEARCH TOLERANCE = 3.0
END MPC
```

7.12.7.3 Tied Multi-Point Constraints

The tied type of MPC imposes a constraint that ties together the motion of the three translational degrees of freedom for a set of nodes. Nodes are not specified as being masters or slaves for this type of constraint. The set of nodes to be tied together can be specified as either a list of node IDs or with a node set by using the TIED NODES or TIED NODE SET command.

```
TIED NODES = <integer list>tied_nodes
TIED NODE SET = <string list>tied_nset
```

The TIED NODES command line is used to specify an integer list of IDs of the nodes to be tied together. The TIED NODE SET can be used to specify the name of a node set that contains the nodes to be tied together. Only one of these commands can be used in a given MPC command block.

Warning: The tied MPC described here does not do a contact search. For the MPC to behave like tied contact, use the commands described in Section 7.12.7.2.

7.12.7.4 Resolve Multiple MPCs

The behavior of multi-point constraints is ill-defined when a master node is constrained to more than one set of slave nodes. Sierra/SM's MPC capability can handle chained MPCs, where a master node is a slave node in another constraint, but it cannot simultaneously enforce multiple MPCs that have the same master.

```
RESOLVE MULTIPLE MPCS = ERROR|FIRST WINS|LAST WINS(ERROR)
```

The RESOLVE MULTIPLE MPCS command line, used within the region scope, controls how to resolve cases where a slave node is constrained to more than one set of master nodes. Although multiple MPCs cannot be simultaneously enforced, this command provides ways to work around this problem that may be acceptable in many situations. The default option is ERROR, which results in an error message and terminates the code if this occurs. Alternatively, this command can be set to FIRST WINS to keep the first MPC found for a given slave node or LAST WINS to keep the last MPC found. This command line controls the behavior of all MPCs in the model.

7.12.7.5 Constraining a Subset of all DOFs

By default, multi-point constraints are applied to all degrees of freedom of the nodes involved. For nodes that only have translational degrees of freedom, all three components (X, Y and Z) are constrained. Likewise, for nodes that have both translational and rotational degrees of freedom, all six components (X, Y, Z, RX, RY and RZ) are constrained. The COMPONENTS command line can be used to enforce the constraint on a subset of the degrees of freedom. If the COMPONENTS command line is included in a MPC command block, only the components listed would be constrained.

Depending on how the model is constructed, there may be a gap between the master and slave nodes of an MPC in the initial configuration. MPCs are best posed when this gap is zero. The option `REMOVE GAPS AND OVERLAPS`, if set to `ON`, moves the MPC'd node to the constraint point during initialization, explicitly zeroing the gap. The default value for this command is `OFF`.

7.12.8 Submodel

```
BEGIN SUBMODEL
  #
  EMBEDDED BLOCKS = <string list>embedded_block
  ENCLOSING BLOCKS = <string list>enclosing_block
END [SUBMODEL]
```

Sierra/SM provides a method to embed a submodel in a larger finite element model. The element blocks for both the submodel and the larger system model should exist in the same mesh file. The space occupied by the embedded blocks should also be occupied by the enclosing blocks.

This capability ties each node of the submodel to an element in the larger finite element model. The code makes no correction for mass due to volume overlap. However, this correction in many cases can be done easily by hand simply by adjusting the density of the submodel block so that it is the difference between the density of the submodel block and the enclosing block.

The embedded blocks (the submodel blocks) and the enclosing blocks (the system model blocks) are specified using the following two line commands:

```
  EMBEDDED BLOCKS = <string list>embedded_block
  ENCLOSING BLOCKS = <string list>enclosing_block
```

For example, to embed `block_7` and `block_8` inside a system model where the embedded blocks are within `block_2`, `block_3`, and `block_5`, the following can be used:

```
BEGIN SUBMODEL
  EMBEDDED BLOCKS = block_7 block_8
  ENCLOSING BLOCKS = block2 block_3 block_5
END
```

7.13 References

1. Brown, K. H., J. R. Koteras, D. B. Longcope, and T. L. Warren. *CavityExpansion: A Library for Cavity Expansion Algorithms, Version 1.0*. SAND2003-1048. Albuquerque, NM: Sandia National Laboratories, April 2003. pdf.

2. Lysmer, J., and R. L. Kuhlmeyer. "Finite Dynamic Model for Infinite Media." *Journal of the Engineering Mechanics Division, Proceedings of the American Society of Civil Engineers* (August 1979): 859–877.

3. Cook, R. D., Malkus, D. S., and Plesha, M. E. *Concepts and Applications of Finite Element Analysis, Third Edition.* New York: John Wiley and Sons, 1989.

Chapter 8

Contact

This chapter describes the input syntax for defining interactions of contact surfaces in a Sierra/SM analysis. For more information on contact and its computational details, consult References 1 and 2.

Contact is the interaction of bodies when they physically touch. Contact algorithms ensure that surfaces do not interpenetrate, and that the interface behavior is computed based on specified surface physics models (for example, energy dissipation from a Coulomb friction model).

Contact between surfaces is modeled by first defining and then enforcing a set of contact constraints. A simple example of contact is shown in Figure 8.1. Block *a* has exterior surface *a*, and block *b* has exterior surface *b*. The surfaces are defined by a collection of finite element faces. For this two-dimensional drawing, the faces are straight lines between two nodes.

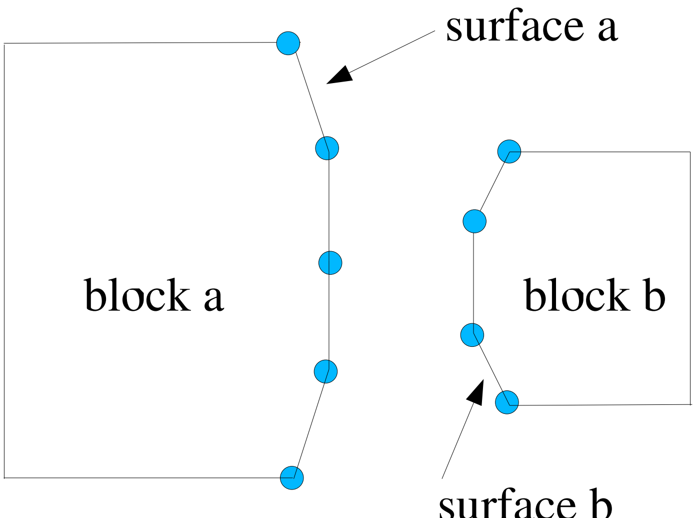


Figure 8.1: Two blocks at time step n before contact

Figure 8.1 shows the two blocks at time step n. Figure 8.2 shows the blocks at time step $n+1$ when the blocks have moved and deformed under the influence of external and internal forces. Before

contact is taken into account, the two blocks interpenetrate one another. This interpenetration is removed by applying the contact algorithm.

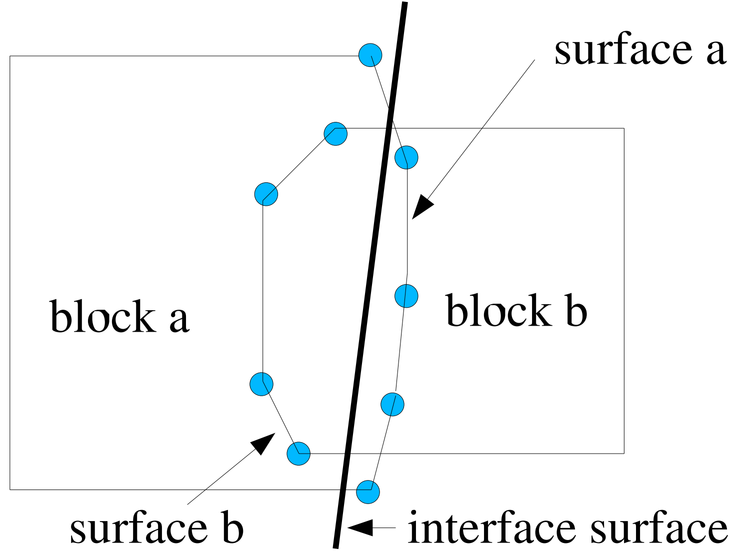


Figure 8.2: Two blocks at time step $n + 1$, after penetration

There are two major types of contact algorithms, both of which are available in Sierra/SM: node/face and face/face. Node/face contact enforcement is based on the concept of preventing nodes from penetrating faces on the opposing surface. Face/face contact prevents faces on opposing surfaces from penetrating each other.

For interpenetration to occur as shown in Figure 8.2, any node on surface *a* that penetrates surface *b* must pass through some face on surface *b*. Likewise, each node on surface *b* that penetrates surface *a* must pass through some face on surface *a*.

In a node/face contact algorithm, all the nodes on surface *a* could be forced to lie on surface *b*, where surface *b* has the configuration shown in Figure 8.2. In this case, surface *b* would be a master surface and surface *a* would be a slave surface. Or all nodes on surface *b* could be forced to lie on surface *a*, in which case surface *a* is a master surface and surface *b* is the slave.

One surface of an interaction may be designated as the master and the other as the slave. Alternatively, the code can automatically select the master and slave surfaces. More typically, nodes on both surfaces are moved to what can be described as an interface surface, which is shown by a thick black line in Figure 8.2. The interface surface is shown here as a straight line, but it would actually be a curved line in almost all cases.

For the typical case, for each node on surface *a* that has penetrated some face on surface *b*, forces based on the amount of node penetration are placed on both the node on *a* and the nodes associated with the face on *b* to remove the interpenetration. Likewise, forces are computed for each node on surface *b* that has penetrated some face on surface *a* and for the nodes associated with the penetrated face on surface *a*. The combination of all these forces will remove the interpenetration of the two blocks and produce the final interface configuration.

The preceding two-dimensional example is analogous to much of the contact that is encountered when contact is used in an analysis. Surfaces are generated that consist of a collection of faces. Contact interactions are defined on the current geometry and then contact enforcement is used to remove the interpenetration of the surfaces as well as satisfy surface physics.

Sierra/SM will also handle variations of contact described as follows:

- A special case of contact called "tied contact" allows you to tie two surfaces on different objects together. The two surfaces that are tied together share a coincident surface or are in very close proximity at the initial analysis. Tied contact keeps the surface together as the model deforms. Tied contact can be used to simplify meshing allowing creating attached parts without a fully contiguous finite element mesh.
- A mesh could have an initial overlap of two bodies due to the meshing process. We refer to this situation as "initial overlap." In explicit dynamics you have the option of removing this initial overlap in a stress free way. Remove of initial overlap prevents an immediate violent repulsion between the overlapping surfaces.

To specify contact enforcement in a model, the user must do the following:

- Identify all surfaces to be considered for contact.
- Define friction models to be used in the surface interactions.
- Identify which surfaces will interact with one another and which friction models will be used in those interactions.
- Specify various algorithmic options to help the contact algorithm run faster, more robustly, or more accurately for the particular problem being analyzed.

8.1 Contact Definition Block

```
BEGIN CONTACT DEFINITION <string>name
  #
  # contact surface definition commands
  SKIN ALL BLOCKS = <string>ON|OFF(OFF)
    [EXCLUDE <string list> block_names]
  CONTACT SURFACE <string>name CONTAINS <string list>surface_names
  #
  BEGIN CONTACT SURFACE <string>name
    BLOCK = <string list>block_names
    SURFACE = <string list>surface_names
    NODE SET = <string list>node_set_names
    REMOVE BLOCK = <string list>block_names
    REMOVE SURFACE = <string list>surface_names
    REMOVE NODE SET = <string list>nodelist_names
    SUBSET <string>subname WITH NORMAL <real> <real> <real>
  END [CONTACT SURFACE <string>name]
  #
  CONTACT NODE SET <string>surface_name
    CONTAINS <string>nodelist_names
  #
  BEGIN ANALYTIC PLANE <string>name                     (Explicit)
    NORMAL = <string>defined_direction
    POINT = <string>defined_point
    REFERENCE RIGID BODY = <string>rb_name
  END [ANALYTIC PLANE <string>name]
  #
  BEGIN ANALYTIC CYLINDER <string>name                  (Explicit)
    CENTER = <string>defined_point
    AXIAL DIRECTION = <string>defined_axis
    RADIUS = <real>cylinder_radius
    LENGTH = <real>cylinder_length
    CONTACT NORMAL = <string>OUTSIDE|INSIDE
  END [ANALYTIC CYLINDER <string>name]
  #
  BEGIN ANALYTIC SPHERE <string>name                    (Explicit)
    CENTER = <string>defined_point
    RADIUS = <real>sphere_radius
  END [ANALYTIC SPHERE <string>name]
  #
  # friction model definition commands
  BEGIN FRICTIONLESS MODEL <string>name
  END [FRICTIONLESS MODEL <string>name]
  #
  BEGIN CONSTANT FRICTION MODEL <string>name
    FRICTION COEFFICIENT = <real>coeff
```

```
END [CONSTANT FRICTION MODEL <string>name]
#
BEGIN TIED MODEL <string>name
END [TIED MODEL <string>name]
#
BEGIN GLUED MODEL <string>name
END [GLUED MODEL <string>name]
#
BEGIN SPRING WELD MODEL <string>name
  NORMAL DISPLACEMENT FUNCTION = <string>func_name
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_factor(1.0)
  TANGENTIAL DISPLACEMENT FUNCTION = <string>func_name
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_factor(1.0)
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [SPRING WELD MODEL <string>name]
#
BEGIN SURFACE WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [SURFACE WELD MODEL <string>name]
#
BEGIN AREA WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [AREA WELD MODEL <string>name]
#
BEGIN ADHESION MODEL <string>name
  ADHESION FUNCTION = <string>func_name
  ADHESION SCALE FACTOR = <real>scale_factor(1.0)
END [ADHESION MODEL <string>name]
#
BEGIN COHESIVE ZONE MODEL <string>name
  TRACTION DISPLACEMENT FUNCTION = <string>func_name
  TRACTION DISPLACEMENT SCALE FACTOR = <real>scale_factor(1.0)
  CRITICAL NORMAL GAP = <real>crit_norm_gap
  CRITICAL TANGENTIAL GAP = <real>crit_tangential_gap
```

Explicit (margin markers beside SPRING WELD, SURFACE WELD, AREA WELD, ADHESION, and COHESIVE ZONE MODEL blocks)

```
END [COHESIVE ZONE MODEL <string>name]
#
BEGIN JUNCTION MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CUTOFF DISTANCE FOR TANGENTIAL TRACTION =
    <real>distance
END [JUNCTION MODEL <string>name]
#
BEGIN THREADED MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR =
    <real>scale_factor(1.0)
  TANGENTIAL TRACTION GAP FUNCTION = <string>func_name
  TANGENTIAL TRACTION GAP SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [THREADED MODEL <string>name]
#
BEGIN PV_DEPENDENT MODEL <string>name
  STATIC COEFFICIENT = <real>stat_coeff
  DYNAMIC COEFFICIENT = <real>dyn_coeff
  VELOCITY DECAY = <real>vel_decay
  REFERENCE PRESSURE = <real>p_ref
  OFFSET PRESSURE = <real>p_off
  PRESSURE EXPONENT = <real>p_exp
END [PV_DEPENDENT MODEL <string>name]
#
BEGIN HYBRID MODEL <string>name
  INITIALLY CLOSE = <string>close_name
  INITIALLY FAR   = <string>far_name
END [HYBRID MODEL <string>name]
#
BEGIN TIME VARIANT MODEL <string>name
  MODEL = <string>model DURING PERIODS <string_list>time_periods
END [TIME VARIANT MODEL]
#
BEGIN USER SUBROUTINE MODEL <string>name
  INITIALIZE MODEL SUBROUTINE = <string>init_model_name
```

Margin labels: **Explicit** (beside JUNCTION MODEL, THREADED MODEL, PV_DEPENDENT MODEL, USER SUBROUTINE MODEL)

```
    INITIALIZE TIME STEP SUBROUTINE = <string>init_ts_name
    INITIALIZE NODE STATE DATA SUBROUTINE =
      <string>init_node_data_name
    LIMIT FORCE SUBROUTINE = <string>limit_force_name
    ACTIVE SUBROUTINE = <string>active_name
    INTERACTION TYPE SUBROUTINE = <string>interaction_name
  END [USER SUBROUTINE MODEL <string>name]
  #
  #  interaction definition commands
  BEGIN INTERACTION DEFAULTS [<string>name]
    CONTACT SURFACES = <string list>surface_names
    SELF CONTACT = <string>ON|OFF(OFF)
    GENERAL CONTACT = <string>ON|OFF(OFF)
    AUTOMATIC KINEMATIC PARTITION = <string>ON|OFF(OFF)
    INTERACTION BEHAVIOR = <string>SLIDING|
      INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
    FRICTION MODEL = <string>friction_model_name|
      FRICTIONLESS(FRICTIONLESS)
    CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
  END [INTERACTION DEFAULTS <string>name]
  #
  BEGIN INTERACTION [<string>name]
    SURFACES = <string>surface1 <string>surface2
    MASTER = <string>surface
    SLAVE = <string>surface
    CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
    #
    # tolerance commands
Implicit    CAPTURE TOLERANCE = <real>cap_tol
    NORMAL TOLERANCE = <real>norm_tol
    TANGENTIAL TOLERANCE = <real>tang_tol
    FRICTION MODEL = <string>model_name|TIED|FRICTIONLESS
      (FRICTIONLESS)
    FACE MULTIPLIER = <real>face_multiplier(0.1)
    OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
    INTERFACE MATERIAL = <string>int_matl_name
      MODEL = <string>int_model_name
    KINEMATIC PARTITION = <real>kin_part
    AUTOMATIC KINEMATIC PARTITION
    INTERACTION BEHAVIOR = <string>SLIDING|INFINITESIMAL_SLIDING|
      NO_INTERACTION(SLIDING)
    #
    # kinematic enforcement only
Implicit    FRICTION COEFFICIENT = <real>coeff
Implicit    FRICTION COEFFICIENT FUNCTION = <string>coeff_func
Implicit    PUSHBACK FACTOR = <real>pushback_factor(1.0)
```

```
Implicit      TENSION RELEASE = <real>ten_release
Implicit      TENSION RELEASE FUNCTION = <string>ten_release_func
            END [INTERACTION <string>name]
            #
            ACTIVE PERIODS = <string list>period_names
            #
            BEGIN USER SEARCH BOX <string>name
              CENTER = <string>center_point
              X DISPLACEMENT FUNCTION = <string>x_disp_function_name
              Y DISPLACEMENT FUNCTION = <string>y_disp_function_name
              Z DISPLACEMENT FUNCTION = <string>z_disp_function_name
              X DISPLACEMENT SCALE FACTOR = <real>x_disp_scale_factor
              Y DISPLACEMENT SCALE FACTOR = <real>y_disp_scale_factor
              Z DISPLACEMENT SCALE FACTOR = <real>z_disp_scale_factor
            END [SEARCH OPTIONS <string>name]
            #
            # contact algorithm option commands
Implicit    ENFORCEMENT = <string>AL|KINEMATIC(AL)
            SEARCH = <string>ACME|DASH(ACME)
Explicit    UPDATE ALL SURFACES FOR ELEMENT DEATH = <string>ON|OFF(ON)
            #
Explicit    BEGIN REMOVE INITIAL OVERLAP
              OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
              OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
              SHELL OVERLAP ITERATIONS = <integer>max_iter(10)
              SHELL OVERLAP TOLERANCE = <real>shell_over_tol(0.0)
            END [REMOVE INITIAL OVERLAP]
            #
Explicit    MULTIPLE INTERACTIONS = <string>ON|OFF(ON)
Explicit    MULTIPLE INTERACTIONS WITH ANGLE = <real>angle_in_deg(60.0)
            BEGIN SURFACE NORMAL SMOOTHING
              ANGLE = <real>angle_in_deg
              DISTANCE = <real>distance
              RESOLUTION = <string>NODE|EDGE
            END [SURFACE NORMAL SMOOTHING]
Explicit    ERODED FACE TREATMENT = <string>NONE|ALL(ALL)
            #
            BEGIN SURFACE OPTIONS
              SURFACE = <string_list>surface_names
              REMOVE SURFACE = <string_list>removed_surface_names
              BEAM RADIUS = <real> radius
              LOFTING ALGORITHM = <string>ON|OFF(ON)
              COINCIDENT SHELL TREATMENT = <string>DISALLOW|IGNORE|
                SIMPLE(DISALLOW)
              COINCIDENT SHELL HEX TREATMENT = <string>DISALLOW|
                IGNORE|TAPERED|EMBEDDED(DISALLOW)
              CONTACT SHELL THICKNESS =
```

```
        ACTUAL_THICKNESS|LET_CONTACT_CHOOSE(ACTUAL_THICKNESS)
      ALLOWABLE SHELL THICKNESS TO ELEMENT SIZE RATIOS =
        <real>lower_bound(0.1) TO <real>upper_bound(1.0)
    END [SURFACE OPTIONS]
    #
    COMPUTE CONTACT VARIABLES = ON|OFF(OFF)
    #
    BEGIN SEARCH OPTIONS [<string>name]
      GLOBAL SEARCH INCREMENT = <integer>num_steps(1)
      GLOBAL SEARCH ONCE = <string>ON|OFF(OFF)
      SEARCH TOLERANCE = <string>AUTOMATIC|USER_DEFINED(AUTOMATIC)
      NORMAL TOLERANCE = <real>norm_tol
      TANGENTIAL TOLERANCE = <real>tang_tol
Implicit      CAPTURE TOLERANCE = <real>cap_tol
Implicit      TENSION RELEASE = <real>ten_release
Implicit      SLIP PENALTY = <real>slip_pen
      FACE MULTIPLIER = <real>face_multiplier(0.1)
    END [SEARCH OPTIONS <string>name]
    #
Explicit    BEGIN ENFORCEMENT OPTIONS [<string>name]
      MOMENTUM BALANCE ITERATIONS = <integer>num_iter(5)
      NUM GEOMETRY UPDATE ITERATIONS = <integer>num_iter(5)
    END [ENFORCEMENT OPTIONS <string>name]
    #
    BEGIN DASH OPTIONS
      INTERACTION DEFINITION SCHEME = EXPLICIT|AUTOMATIC (AUTOMATIC)
      SEARCH LENGTH SCALING = <real>scale(0.15)
      ACCURACY LEVEL = <real>accuracy(1.0)
      SUBDIVISION LEVEL = <int>sublevel(0)
    END
  END [CONTACT DEFINITION <string>name]
```

The command block begins with the input line:

```
BEGIN CONTACT DEFINITION <string>name
```

and is terminated with the input line:

```
END [CONTACT DEFINITION <string>name]
```

The string `name` is a name for this contact definition. The name should be unique among all the contact definitions in an analysis and is used in diagnostics printed in the log file or error messages related to the contact definition.

A typical analysis will have only one `CONTACT DEFINITION` command block. However, more than one contact definition may be used. Each `CONTACT DEFINITION` command block creates

its own set of contact constraints that operate independently from one another. Use of fewer contact definition commands blocks will tend to provide more efficient processing and better conflict resolutions when different types of contact constraints come into conflict.

Contact behaves somewhat differently in explicit verses explicit analyses. Additionally there are multiple contact enforcement libraries available (ACME and Dash). The detailed descriptions of each of the commands list what options and command are available based on what type of analysis is being done and contact library is being used. Unless otherwise noted the behavior of a command is the same regardless of the type of analysis or the contact algorithm being used.

8.2 Defining the Contact Surfaces

The user must provide a description of which surfaces will be considered for contact. There are many options available to either automatically detect and create contact surfaces or fine tune specific contact surface definitions.

Many separate contact surfaces can be defined by the analysis. Each contact surface can contain one or more of the following:

- The contact surface can include the exterior faces or "skin" of a block of finite elements. An exterior face is an outward facing finite element face that is not sandwiched between two contiguous finite elements. For more information on blocks skinning algorithms see Section 8.2.7.
- The contact surface can include a set of faces defined as a surface on the input mesh. All faces included in the face set are considered for contact even if those faces are not exterior faces of the finite elements.
- The contact surface can include a set of nodes defined by a node set, surface, or block on the input mesh. These nodes need not be attached to finite element faces. The nodes could for example be the nodes of beam elements or particle elements.
- The contact surface can be a lofted set of faces around structural or lower dimensional elements. For example contact can define a lofted brick around a two dimensional shell element, a lofted prism around a one dimensional beam element, or lofted sphere around a zero dimensional particle element.
- The contact surface can be defined by an analytic surface. An analytic surface is defined by an algebraic expression, not by a collection of faces derived from elements. For example, an algebraic expression that defines the surface of a cylinder.

8.2.1 Contact Surface Command Line

```
CONTACT SURFACE <string>name CONTAINS <string list>surface_names
```

This command line identifies a set of surfaces (specified as side sets) and element blocks that will be considered as a single contact surface; the string `name` is the unique name given to this contact surface. The list denoted by `surfaces_names` is a list of strings identifying surfaces and blocks that are to be associated with this contact surface. The surfaces can be side sets, element blocks, or any combination of the two as defined in the mesh file.

Any specified element blocks in the `surface_names` list that are solid elements (bricks, tetrahedra, wedges, etc.) are "skinned." A surface is created from all exterior faces of that element block. See Section 8.2.7 for more information on block skinning.

For specified element blocks in the `surface_names` list that are lower dimensional (shells, beams, and particles), lofted geometry is created for use with contact enforcement.

Known Issue: The ACME contact algorithm is unable to loft beam and particle elements and will ignore any beam or particle element blocks. See Section 8.5.8 for more information on lofted geometry.

The `name` you create for a surface can be referenced in command blocks that specify how that surface will interact with other contact surfaces or interact when the surface contacts itself. See Section 8.4.1.1 and Section 8.4.2.1.

The surfaces can contain heterogeneous combinations of finite element faces and lofted geometry.

Warning: A given finite element face or lofted element can be placed in at most one contact surface. See Section 8.2.5 for more information on what happens if different contact surfaces try to include the same mesh objects.

8.2.2 Contact Node Set Command Line

```
CONTACT NODE SET <string>surface_name
  CONTAINS <string list>nodelist_names
```

The CONTACT NODE SET command line creates a contact surface named surface_name that contains a set of nodes. The list denoted by nodelist_names is a list of strings identifying node sets, surfaces, and blocks that are to be associated with the contact node set surface_name. All nodes present in the listed node sets, surfaces, or blocks will be added to the contact node set. The created node set contains only the mesh nodes of surfaces or blocks, not the faces, and not any lofted geometry. Thus, the contact node set can be used only in a node to face or node to analytic surface interaction.

8.2.3 Contact Surface Command Block

```
BEGIN CONTACT SURFACE <string>name
  BLOCK = <string list>block_names
  SURFACE = <string list>surface_names
  NODE SET = <string list>node_set_names
  REMOVE BLOCK = <string list>block_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE NODE SET = <string list>node_set_names
  SUBSET <string>subname WITH NORMAL <real> <real> <real>
END [CONTACT SURFACE <string>name]
```

The `CONTACT SURFACE` command block can be used to define a contact surface consisting of a more complex collection of finite element faces or nodes than can be defined using the `CONTACT SURFACE` command line.

If you want to define a surface named `name` that is a set of faces, you can use a combination of the command lines `BLOCK`, `SURFACE`, `REMOVE BLOCK`, and `REMOVE SURFACE`. Blocks listed in the `block_names` list will be skinned for exterior faces if the block contains solid elements (like hexes, tetrahedra, or wedges.) For blocks that contain shells, beams, or particles, the lofted geometry around those shells beams and particles will be included in the contact surface.

If a `BLOCK` command line and a `SURFACE` command line are both specified within the same `CONTACT SURFACE` command block, the contact surface created will be the union of the faces in the skin of the blocks and the faces in the specified surfaces in the mesh file.

A node set for use in contact can be created by using the `NODE SET` command. If `REMOVE NODE SET`, `REMOVE SURFACE`, or `REMOVE BLOCK` is specified, the nodes in the specified node sets, surfaces, and blocks are removed from the contact node node set.

The contact surface created by the `CONTACT SURFACE` command block must be either a set of faces and lofted geometry or a set of nodes. It cannot be a mix of the two. Thus, the `BLOCK`, `SURFACE`, `REMOVE BLOCK`, and `REMOVE SURFACE` commands can be used together to create a resultant set of faces or lofted geometry. Likewise, the `NODE SET`, `REMOVE NODE SET`, `REMOVE SURFACE`, and `REMOVE BLOCK` commands can also be used together to create a resultant set of nodes. The `NODE SET` and `BLOCK` or `SURFACE` commands, however, cannot be used together in the same `CONTACT SURFACE` command block.

At least one of the `BLOCK`, `SURFACE`, or `NODE SET` command lines must be present in a given `CONTACT SURFACE` command block.

8.2.3.1 Surface Subsetting

```
SUBSET <string>subname WITH NORMAL <real> <real> <real>
```

A contact surface can optionally be subdivided into multiple sub-surfaces based on the surface normal. This command will create a new surface that contains the faces mostly aligned with the specified normal direction. Faces for which the dot product of the unitized face normal vector and

the unitized specified normal is greater than 0.45 are placed in the new surface. All other faces are left in the original surface. For example, the following command:

```
BEGIN CONTACT SURFACE surf1
  BLOCK = block_1
  SUBSET xp WITH NORMAL  1  0  0
  SUBSET zm WITH NORMAL  0  0 -1
END
```

will create three contact surfaces. Surface `surf1_xp` will contain the faces most closely aligned with the positive x-axis. Surface `surf1_zm` will contain the faces mostly aligned with the negative z-axis. Surface `surf1` will contain all remaining faces in the skin of `block_1`.

8.2.4 Skin All Blocks Command

```
SKIN ALL BLOCKS = <string>ON|OFF(OFF)
  [EXCLUDE <string list>block_names]
```

The SKIN ALL BLOCKS command provides a convenient way to create a number of contact surfaces covering the exterior skin or lofted geometry of all or most of the element blocks in the analysis. This command may create many surfaces, the name of each surface is same as the element block it was generated from. The optional EXCLUDE keyword can be used to skin all element blocks except the listed blocks. For more information on how block skinning works see Section 8.2.7

The SKIN ALL BLOCKS is useful for large models in which individual specifying each contact surfaces would be unwieldy. The behavior of skin all blocks is best described by example. If a finite element analysis contains five element blocks:

- block_1 (hex8 elements)
- block_3 (tetrahedral elements)
- block_4 (shell elements)
- block_5 (beam elements)

The command:

```
SKIN ALL BLOCKS = ON EXCLUDE block_2
```

Would be equivalent to defining several independent surfaces with:

```
CONTACT SURFACE block_1 CONTAINS block_1
CONTACT SURFACE block_3 CONTAINS block_3
CONTACT SURFACE block_4 CONTAINS block_4
CONTACT SURFACE block_5 CONTAINS block_5
```

The generated contact surfaces would be named block_1, block_3, block_4, and block_5. The contact surfaces named block_1 and block_3 would contain the exterior faces of the solid element blocks block_1 and block_3, respectively. The contact surfaces named block_4 and block_5 would contain lofted geometry around the shell and beam elements in block_4 and block_5, respectively.

The SKIN ALL BLOCKS command can be combined with any of the other surface definition methods available in the CONTACT DEFINITION command block.

8.2.5 Overlapping Contact Surfaces

The contact enforcement algorithm only allows for a face to be associated with a single contact surface. If a face were allowed to belong to more than one surface involved in contact, ambiguities would arise in determining the interaction properties for that face. A situation where a face could potentially belong to more than one contact surface is when overlapping surfaces are used in contact. This could also occur if a combination of skinned block surfaces and side sets on the mesh file are used to define a contact surface. The side sets can potentially include the same finite element faces as the block skin.

If a given finite element face is potentially part of more than one contact surface, that face is included in the first contact surface to which it belongs that is listed in the `CONTACT DEFINITION` block. If the same face also potentially belongs to any other contact surface listed later in the `CONTACT DEFINITION` block, it is not included in the later contact surface. A warning message is generated in the log file when this occurs.

Overlapping contact surfaces can be used to implement more advanced behavior such as a pair of surfaces being tied only at a specific point.

8.2.6 Element Death, Remeshing, and Surface Updates

When elements are killed in an analysis, contact surfaces may need to be updated to account for the removal of faces attached to killed elements or the addition of faces exposed by element death. The updates happen automatically when elements in a skinned block are removed due to element death (See Section 6.5). The update of these surfaces requires a full re-initialization of contact. Thus, surface-physics models that involve state data may lose some information when the new contact surfaces are created. If contact surfaces are defined by side sets rather than by block skinning, element death will cause the facets attached to killed elements to be removed from the contact surface, but no new facets will be added.

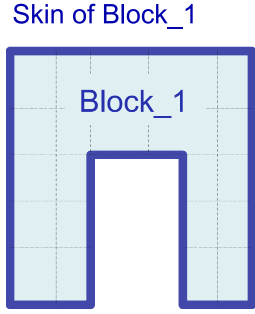


Figure 8.3: Simple block skinning example

8.2.7 Block Skinning and Surface Definition Examples

A surface is defined as a collection of finite element faces, nodes, and lofted geometry. For a continuum element, any face that is not shared with another element can be considered for contact. A shell element has both a top face and a bottom face. Top, bottom, and side surfaces of shells are automatically created by the contact algorithm and may be lofted by a user-specified thickness. Contact is enforced on shells by computing the contact forces on the top and bottom surfaces of the shells and then moving the resulting forces back to the original shell nodes.

The easiest method to create contact surfaces is by skinning blocks. Block skinning identifies the exterior surfaces of a block and puts them in a contact surface. The following examples demonstrate how block skinning works in various circumstances.

8.2.7.1 Simple Block Skinning

See Figure 8.3. This first example is a single block being skinned. This skinning could occur by one of the input lines

```
CONTACT SURFACE block_1 CONTAINS block_1
```

or if the model contains only `block_1`

```
SKIN ALL BLOCKS = ON
```

For either syntax all the exterior facets of `block_1` are automatically identified and are placed into a contact surface which is also named `block_1`.

8.2.7.2 Block Skinning With Multiple Element Blocks

Figure 8.4 shows an example of two blocks that are equivalenced on their interface. An equivalenced interface means that the two blocks share nodes at the interface so that there are no exposed contact faces between `block_1` and `block_2`. The syntax to create the contact surfaces shown in Figure 8.4 is:

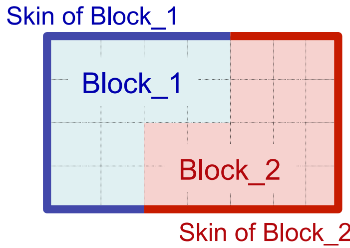


Figure 8.4: Block skinning with multiple blocks example

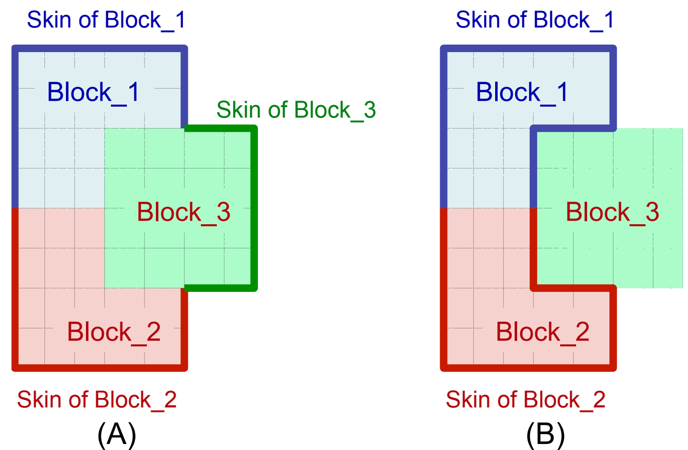


Figure 8.5: Block skinning advanced example

```
CONTACT SURFACE block_1 CONTAINS block_1
CONTACT SURFACE block_2 CONTAINS block_2
```

Alternatively, if the model contains only `block_1` and `block_2`, this could be accomplished by requesting that all blocks be skinned:

```
SKIN ALL BLOCKS = ON
```

For either syntax, all the exterior facets that are connected to elements that are part of `block_1` are placed in contact surface `block_1`. All the exterior facets that are connected to elements of `block_2` are placed in contact surface `block_2`. Any interior facet (one that is sandwiched between two elements) is not placed in any contact surface.

8.2.7.3 More Complex Block Skinning Cases

Block skinning considers only those elements actually present in the contact definition when deciding if a face is exposed. The surfaces shown by heavy lines in Figure 8.5(A) will result from the following syntax:

```
CONTACT SURFACE block_1 CONTAINS block_1
```

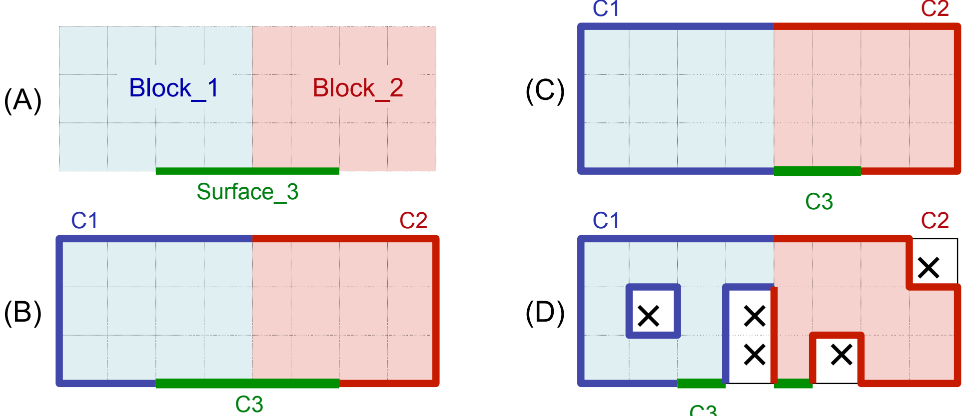


Figure 8.6: Contact surface definition examples

```
CONTACT SURFACE block_2 CONTAINS block_2
CONTACT SURFACE block_3 CONTAINS block_3
```

All the exterior facets of the model have been placed in the contact surfaces associated with elements of the named blocks. Any facet sandwiched between two contact surface blocks is not considered to be on the exterior, and is thus not placed in any contact surface. The surfaces shown in Figure 8.5(B) will result from the following syntax:

```
CONTACT SURFACE block_1 CONTAINS block_1
CONTACT SURFACE block_2 CONTAINS block_2
```

or

```
SKIN ALL BLOCKS = ON EXCLUDE block_3
```

In this case, `block_3` was not included in the contact definition. Thus the faces on the `block_1` `block_3` and `block_2` and `block_3` boundaries will be considered exterior for the purposes of contact. Since `block_3` is not part of the contact definition none of the remaining faces of `block_3` will be part of any contact surface.

8.2.7.4 Combining Block Skinning, Side Sets, and Element Death

Figure 8.6(A) shows a set of element blocks and surface sidesets as defined in a mesh file. Blocks `block_1` and `block_2` are equivalenced sharing nodes on their boundary. The surface `surface_3` is a sideset defined on part of the exterior of `block_1` and `block_2`. The contact surfaces shown in Figure 8.6(B) will be generated by the following syntax:

```
CONTACT SURFACE c3 CONTAINS surface_3
CONTACT SURFACE c1 CONTAINS block_1
CONTACT SURFACE c2 CONTAINS block_2
```

As described in Section 8.2.5, the first surface defined in the input file that includes a facet owns that facet. If any subsequently defined contact surfaces attempt to include a facet that already belongs to a contact surface, it will be left in the original contact surface that it belongs to, and will not be made part of the surface that is defined later in the input file. In other words, a given mesh facet can belong in at most one contact surface, and when a conflict arises, the facet will belong to the first contact surface in the input file that included it. Thus, c3 contains the facets associated with surface_3. Contact surfaces c1 and c2 do not contain any of the facets of side set surface_3.

The surfaces in Figure 8.6(C) are be generated by the following syntax:

```
CONTACT SURFACE c1 CONTAINS block_1
CONTACT SURFACE c3 CONTAINS surface_3
CONTACT SURFACE c2 CONTAINS block_2
```

Since contact surface c1 is listed before contact surface c3 in the input file, c1, the skin of block_1, contains the contact facets that could have been part of c1 or c3. Since contact surface c3 is listed before contact surface c2 in the input file, contact surface c3 includes the remaining portions of surface_3, while c2, the skin of block_2, contains only the remaining exterior facets of block_2.

When element death occurs, new contact facets may be exposed and old contact facets may be deleted. After element death, the ownership of contact facets in the updated contact surfaces will be evaluated as if the dead elements never existed. Figure 8.6(D) shows the effect of element death on the contact surfaces. For this case the contact surfaces are defined by the same syntax as the case shown in Figure 8.6(B):

```
CONTACT SURFACE c3 CONTAINS surface_3
CONTACT SURFACE c1 CONTAINS block_1
CONTACT SURFACE c2 CONTAINS block_2
```

Elements shown in the figure with an X are dead elements. Any newly exposed facets of block_1 and block_2 are added to the relevant contact surfaces. If all the elements attached to contact facet are deleted, the contact facet itself is also deleted. If a contact surface is defined via a side set, that surface can only lose faces. In other words, a contact surface defined by a contact side set cannot gain any faces that were not part of the original side set during the process of element death.

8.2.7.5 Equivalenced Solid and Shell Meshes

A mesh that includes both solid elements such as hexes and shell elements obeys some special rules when defining contact surfaces via block skinning. Figure 8.7(A) shows block_1, (solid hex elements) and block_2 (quadrilateral shell elements). The nodes of the shell element block are equivalenced with the nodes of the hex element block, or in other words, the two blocks share nodes on their interface. Figure 8.7(B) shows the contact surfaces that would be generated using the following commands:

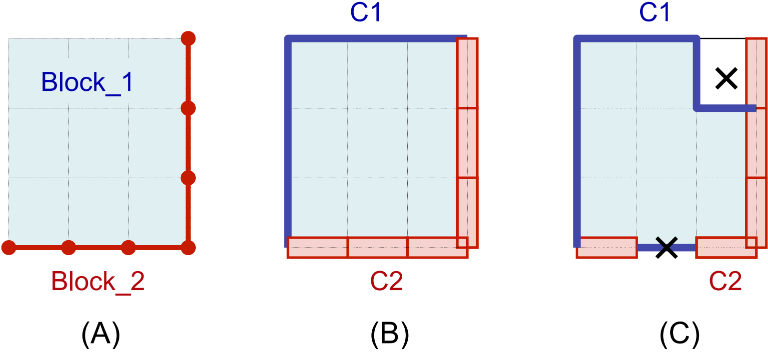


Figure 8.7: Hex and shell contact surface definitions

```
CONTACT SURFACE c1 CONTAINS block_1
CONTACT SURFACE c2 CONTAINS block_2
```

No matter what the order of the contact surfaces in the input file is, the shell block will always be considered "outside" the solid element block for this case. On portions of `block_1` without any cladding shells, the standard block skinning rules apply. On the interface between `block_1` and `block_2` the quad faces on the `block_1` hex elements are considered sandwiched between the `block_1` hex elements and the `block_2` shell elements. Thus, `c1` contains none of the interface facets. Contact surface `c2` contains the lofted shell geometry associated with the shell elements of `block_2`.

In Figure 8.7(C), some elements have been removed due to element death. As in the example in Section 8.2.7.4, when elements are killed via element death the contact surfaces are reevaluated as if those elements were never there. When a cladding shell is removed the exterior face of `block_1` is exposed and joins contact surface `c1`. The death of one of the lofted shell elements removes all faces associated with that element's lofted geometry.

8.2.8 Analytic Contact Surfaces

Sierra/SM permits the definition of rigid analytic surfaces for use in contact. Contact evaluation between a deformable body and a rigid analytic surface can be much faster than contact evaluation between two faceted bodies. Therefore, using a rigid analytic surface is more efficient than using a faceted body to try to approximate a geometric surface.

Several types of analytic surfaces definitions are available.

8.2.8.1 General Analytic Surfaces

Defining a general analytic body is somewhat involved. First the mathematical description of the body must be input at the sierra scope:

```
BEGIN ANALYTIC SURFACE <string>geomName
```

```
    ORIGIN = <real>Ox <real>Oy <real>Oz
    RAXIS  = <real>Rx <real>Ry <real>Rz
    SAXIS  = <real>Sx <real>Sy <real>Sz

    STARTING POINT = <real>start_r <real>start_s
    CIRCLE    = <real>p2r <real>p2s <real>p3r <real> p3s
    LINE      = <real>p2r <real>p1r
    REVOLVE   = <real>start_theta <real>end_theta
    TRANSLATE = <real>start_t <real> end_t
  END
```

The analytic surface is created by defining a two dimensional set of lines and then extruding or revolving those lines to create a three dimensional surface.

The ORIGIN command defines the XYZ coordinates of the origin of the local two dimensional RS coordinate system. The RAXIS and SAXIS commands define the R and S axes for the two dimensional coordinate system. If R and S are not fully orthogonal as input SAXIS will be orthogonalized against RAXIS. A T axis is also defined that is orthogonal to both R and S and obeys the right hand rule.

Exactly one STARTING POINT command must be placed in the command block. The starting point command gives the coordinates for the first RS line segment point. Subsequent lines and curves are added by providing LINE and CIRCLE commands. The LINE command creates a new linear line segment from the end of the last line segment or the starting point to the provided point. The CIRCLE command creates a new curved line segment. The CIRCLE command creates a circular arc going though the end of the last line segment or starting point and the two provided points. The circular arc will end at the last provided point p3.

Once all line segments are defined in RS coordinates, those line segments can be turned into a three dimensional body. The REVOLVE command revolves a set of lines about the R axis. The start angle and end angle for the revolve are specified in degrees with start_theta and end_theta. The zero degree line for theta is the S axis. The TRANSLATE commands extrudes the two dimensional line segments into a three dimensional surface. The extrusion is along the T axis starting from start_t and ending at end_t. Exactly REVOLVE or one TRANSLATE command must be specified in the ANALYTIC SURFACE command block.

The geometry for the rigid body is defined at the BEGIN SIERRA scope. the rigid surface itself is input into the contact scope via the ANALYTIC GENERAL SURFACE command block inside the BEGIN CONTACT DEFINITION scope:

```
  BEGIN ANALYTIC GENERAL SURFACE <string>surfName
    ANALYTIC SURFACE = <string>geomName
    REFERENCE RIGID BODY = <string>rbName
  END
```

The ANALYTIC GENERAL SURFACE command block creates an analytic surface named surfName. The name surfName can be referenced when defining interactions between contact

surfaces. The `geomName` string in the `ANALYTIC SURFACE` command references the name of an analytic geometry defined via the `ANALYTIC SURFACE` command block in the sierra scope.

Optionally the analytic surface can be associated with a rigid body, for information on rigid bodies see Section 6.3.1. If an analytic surface is associated with a rigid body the surface will translate and rotate along with the rigid body. Additionally any contact forces applied to the analytic surface will be assembled to the rigid body reference node causing the rigid body to move.

The `VISUALIZE CONTACT FACETS` option (See Section 8.7.1) can be used to help confirm that analytic rigid surfaces are being defined as expected. When using the visualize facet option general analytic surfaces will be represented by an approximate faceted surface in the output meshes.

Example: The example below defines an analytic cylinder. The cylinder is centered at `(0.0, 0.0, 0.0)`. The center axis of the cylinder lies along the z axis. The cylinder has a radius `1.0` and length of `10.0`. The cylindrical surface is defined by creating a line in RS space and then revolving that line about the T axis.

The cylinder is attached to a rigid body `block_1000`, the analytic cylinder around `block_1000` impacts the meshed exterior surface of finite element block `block_1`.

```
BEGIN SIERRA
  BEGIN ANALYTIC SURFACE axleGeom
    ORIGIN         = 0.0  0.0  0.0
    RAXIS          = 1.0  0.0  0.0
    SAXIS          = 0.0  1.0  0.0
    STARTING POINT = -5.0 1.0
    LINE           =  5.0 1.0
    REVOLVE        = 0.0 360.0
  END

  BEGIN RIGID BODY axleRB
  END

  BEGIN SOLID SECTION axleSect
    RIGID BODY = axleSect
  END

  BEGIN FINITE ELEMENT MODEL mesh1
    BEGIN PARAMETERS FOR BLOCK block_1
    END

    BEGIN PARAMETER FOR BLOCK block_1000
      SECTION = axleSect
    END
  END


  BEGIN PRESTO PROCEDURE p1
    BEGIN PRESTO REGION r1
```

```
        BEGIN CONTACT DEFINITION c1

          BEGIN ANALYTIC GENERAL SURFACE axleSurf
            ANALYTIC SURFACE = axleGeom
            REFERENCE RIGID BODY = block_1000
          END

          CONTACT SURFACE block_1_surf CONTAINS block_1

          BEGIN INTERACTION
            MASTER         = axleSurf
            SLAVE          = block_1_surf
            FRICTION MODEL = frictionless
          END
        END
      END
    END
  END
```

8.2.8.2 Plane

```
BEGIN ANALYTIC PLANE <string>name
  NORMAL = <string>defined_direction
  POINT = <string>defined_point
  REFERENCE RIGID BODY = <string>rb_name
END [ANALYTIC PLANE <string>name]
```

Analytic planes are not deformable and two analytic planes cannot interact with each other through contact. The `ANALYTIC PLANE` command block for defining an analytic plane begins with the input line:

The string `name` is a user-selected name for this particular analytic plane. This name is used to identify the surface in the interaction definitions. The string `defined_direction` in the `NORMAL` command line refers to a vector that has been defined with a `DEFINE DIRECTION` command line; this vector defines the outward normal to the plane. The string `defined_point` in the `POINT` command line refers to a point in a plane that has been defined with a `DEFINE POINT` command line. The plane is infinite in size. The body the plane is contacting should be initially be on the positive outward normal side of the plane. See Section 2.1.6 for more information on defining points and directions.

The `REFERENCE RIGID BODY` command can be used to connect the analytic plane to the rigid body block named by `rb_name`. If the rigid body block rotates or translates, the analytic contact plane will rotate and translate with it. The `REFERENCE RIGID BODY` option only works with the Dash search option.

8.2.8.3 Cylinder

```
BEGIN ANALYTIC CYLINDER <string>name
  CENTER = <string>defined_point
  AXIAL DIRECTION = <string>defined_axis
  RADIUS = <real>cylinder_radius
  LENGTH = <real>cylinder_length
  CONTACT NORMAL = <string>OUTSIDE|INSIDE
END [ANALYTIC CYLINDER <string>name]
```

Analytic cylindrical surfaces are not deformable, they cannot be moved, and two analytic cylindrical surfaces will not interact with each other.

The string `name` is a user-selected name for this particular analytic cylinder. This name is used to identify the surface in the interaction definitions. The cylindrical surface has a finite length. The center point of the cylinder and the direction of the radial axis of the cylinder are defined by the `CENTER` and `AXIAL DIRECTION` command lines, respectively. The string `defined_point` in the `CENTER` command line refers to a point that has been defined with a `DEFINE POINT` command line; the string `defined_axis` in the `AXIAL DIRECTION` command line refers to a direction that has been defined with a `DEFINE DIRECTION` command line. See Section 2.1.6 for more information on defining points and directions.

The radius of the cylinder is the real value `cylinder_radius` specified with the `RADIUS` command line, and the length of the cylinder is the real value `cylinder_length` specified by the `LENGTH` command line. The length of the cylinder (`cylinder_length`) extends a distance of `cylinder_length` divided by 2 along the cylinder axis in both directions from the center point.

The `CONTACT NORMAL` command defines whether the normal of the contact cylinder points outward or inward.

```
CONTACT NORMAL = OUTSIDE | INSIDE
```

8.2.8.4 Sphere

```
BEGIN ANALYTIC SPHERE <string>name
  CENTER = <string>defined_point
  RADIUS = <real>sphere_radius
END [ANALYTIC SPHERE <string>name]
```

Analytic spherical surfaces are not deformable, they cannot be moved, and two analytic spherical surfaces will not interact with each other. The string `name` is a user-selected name for this particular analytic sphere. This name is used to identify the surface in the interaction definitions. The center point of the sphere is defined by the `CENTER` command line, which references a point, `defined_point`, specified by a `DEFINE POINT` command line. The radius of the sphere is the real value `sphere_radius` specified with the `RADIUS` command line. See Section 2.1.6 for more information on defining points.

8.3 Friction Models

Friction models are used to describe the physics of interactions that occur between contact surfaces. The user then associates these friction models to pairs of interactions in the interaction-definition blocks (see Section 8.4.1 and Section 8.4.2). During the search phase of contact, node-face or face-face interactions are identified, and the designated friction model is used to determine how the resulting contact forces are resolved between these pairs.

The following friction models are currently available: frictionless contact, constant coulomb friction, tied contact, glued contact, spring weld, surface weld, area weld, adhesion, cohesive zone, junction, threaded joint, and pressure-velocity–dependent friction. Friction models can also be defined by user subroutines. Not all friction models are available with all analysis types and all contact libraries. If a model is not available for certain analysis types, that fact is noted in the documentation of those models below.

In addition to the friction models, any of the cohesive zone models available for use with interface elements (see Section 5.3) can be used as interface models within Dash contact for explicit dynamic analyses. See Section 8.4.2.5 for more details on the usage of those models within contact.

By default, interactions between contact surfaces with no friction model assigned are treated as frictionless. All friction models are command blocks, although some of the models do not have any command lines inside the command block. The commands for defining the available friction models are described next. Friction models are associated with specific pairings of contact surfaces through the interaction-definition blocks in Section 8.4.1 and Section 8.4.2.

8.3.0.5 Frictionless Model

```
BEGIN FRICTIONLESS MODEL <string>name
END [FRICTIONLESS MODEL <string>name]
```

The `FRICTIONLESS MODEL` command block defines frictionless contact between surfaces. In frictionless contact, contact forces are computed normal to the contact surfaces to prevent penetration, but no forces are computed tangential to the contact surfaces. The string `name` is a user-selected name for this friction model that is used when identifying this model in the interaction definitions. No command lines are needed inside the command block. A default named frictionless model named `frictionless` can be used without defining this command block.

8.3.0.6 Constant Friction Model

```
BEGIN CONSTANT FRICTION MODEL <string>name
  FRICTION COEFFICIENT = <real>coeff
END [CONSTANT FRICTION MODEL <string>name]
```

The `CONSTANT FRICTION MODEL` command block defines a constant coulomb friction coefficient between two surfaces as they slide past each other in contact. No resistance is provided to keep the surfaces together if they start to separate. The string `name` is a user-selected name for this

friction model that is used to identify this model in the interaction definitions, and `coeff` is the constant coulomb friction coefficient. There is no default value for the friction coefficient.

8.3.0.7 Velocity Dependent Coulomb Friction Model

```
BEGIN VELOCITY DEPENDENT COULOMB MODEL <string>name
  COULOMB COEFFICIENT = <string>coeff_func
END [VELOCITY DEPENDENT COULOMB MODEL <string>name]
```

The `VELOCITY DEPENDENT COULOMB MODEL` command block defines a Coulomb friction law between two surfaces as they slide past each other in contact. The value of the Coulomb coefficient at any given time is a function of the relative velocity at the interface. No resistance is provided to keep the surfaces together if they start to separate. The string `name` is a user-selected name for this friction model that is used to identify this model in the interaction definitions. The `COULOMB COEFFICIENT` command provides `coeff_func`, the name of a previously defined XY function, where X is the velocity magnitude and Y is the coulomb coefficient. The velocity dependent coulomb model can be used to define stick-slip friction behavior where the friction coefficient is the stick value when velocity is nominally small and the slip value when velocity is larger than some nominal value.

8.3.0.8 Tied Model

```
BEGIN TIED MODEL <string>name
END [TIED MODEL <string>name]
```

The `TIED MODEL` command block restricts nodes found in initial contact with faces to stay in the same relative location to the faces throughout the analysis. The string `name` is a user-selected name for this friction model that is used to identify this model in the interaction definitions. No command lines are needed inside the command block. A default named tied model named `tied` can be used without defining this command block.

8.3.0.9 Dynamic Tied Model

```
BEGIN DYNAMIC TIED MODEL <string>name
END [DYNAMIC TIED MODEL <string>name]
```

The `DYNAMIC TIED MODEL` command block defines a contact interaction that constrains opposing faces to stay in the same relative location to each other throughout the analysis. No command lines are needed within this command block. A default dynamic tied model named `dynamic_tied` can be used without defining this command block. The dynamic tied model is very similar to the tied model except that the dynamic tied model enforces zero relative velocity at the interface while the tied model enforces zero relative displacement. In the limit, both these are identical. If the analysis involves substantial topology changes, such as element death or remeshing, the dynamic tied model may be more stable, but slightly less accurate.

8.3.0.10 Glued Model

```
BEGIN GLUED MODEL <string>name
END [GLUED MODEL <string>name]
```

Note: This model is available for explicit and implicit analyses, but for implicit analyses, it can only be used with augmented Lagrange enforcement.

The `GLUED MODEL` command block defines a contact interaction that allows the interacting faces to move independently until they come into contact, but once they come into contact, they behave as a tied contact interaction, with no relative normal or tangential motion for the rest of the analysis. The string `name` is a user-specified name for this friction model that is used to reference this model in the interaction definitions. No command lines are needed inside the command block. A default named glued model named `glued` can be used without defining this command block.

8.3.0.11 Spring Weld Model

```
BEGIN SPRING WELD MODEL <string>name
  NORMAL DISPLACEMENT FUNCTION = <string>func_name
  NORMAL DISPLACEMENT SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL DISPLACEMENT FUNCTION = <string>func_name
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_factor(1.0)
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer> num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
 END [SPRING WELD MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The `SPRING WELD MODEL` command block defines a contact friction model that, when applied between two contact surfaces, connects a slave node to the nearest point of a corresponding master face with a spring. The spring behavior is defined by a force-displacement curve in the normal and tangential directions. If the motion of the problem generates displacement between the slave node and its corresponding master face and this motion is in purely the normal or tangential direction, the spring will fail once it passes the maximum displacement in the normal and tangential force-displacement curves, respectively. For displacements that include both normal and tangential components, the spring fails according to a failure criterion defined as the sum of the ratios of the normal and tangential components to their maximum values, raised to a power. If the criterion is greater than 1.0, the spring fails. Once the spring fails, its contact forces reduce over a number of load steps, and the contact evaluation reverts to another user-specified friction model (or frictionless contact if not specified).

In the above command block:

- The string `name` is a user-selected name for this friction model that is used to identify this model in the interaction definitions.

- The normal force-displacement curve is specified by the NORMAL DISPLACEMENT FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope. This function can be scaled by the real value scale_factor in the NORMAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0

- The tangential force-displacement curve is specified by the TANGENTIAL DISPLACEMENT FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command line in the SIERRA scope. This function can be scaled by the real value scale_factor in the TANGENTIAL DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0.

- The real value exponent in the FAILURE ENVELOPE EXPONENT command line specifies how normal and tangential failure criteria may be combined to yield failure of the weld, as described above. The default value for this exponent is 2.0.

- The FAILURE DECAY CYCLES command line describes how many cycles to ramp down the load in the spring weld after it fails through the integer value num_cycles. The default value for the number of decay cycles is 1.

- When the spring weld breaks, the friction model that contact reverts to when evaluating future node-face interactions between the surfaces is identified in the FAILED MODEL command line with the string failed_model_name. The friction model listed in this command must have been previously defined in the input file. The default value for the model used after failure is the frictionless model.

The SPRING WELD MODEL command block is very similar to the Sierra/SM SPOT WELD command block, but permits greater flexibility in specifying a different friction model to be applied after failure.

8.3.0.12 Surface Weld Model

```
BEGIN SURFACE WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [SURFACE WELD MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The SURFACE WELD MODEL command block defines a contact friction model that behaves identically to the TIED MODEL until a maximum force between the node and face of an interaction is reached in the normal direction or the tangential direction. Once this maximum force is reached, the tied contact "fails" and the friction model switches to a different friction model, as specified by the user.

In the above command block, the string name is a user-selected name for this friction model that is used to identify this model in the interaction definitions. The maximum allowed force in the normal direction is specified by the real value normal_cap in the NORMAL CAPACITY command line. The maximum allowed force in the tangential direction is specified by the real value tangential_cap in the TANGENTIAL CAPACITY command line. There are no defaults for these values. The surface weld will break when either the specified normal or tangential capacity is reached. Once the model fails, the applied forces decrease to zero over a number of time steps defined through the integer value num_cycles in the FAILURE DECAY CYCLES command line. The default for num_cycles is 1. The friction model that should be used after the weld fails is identified in the FAILED MODEL command line with the string failed_model_name. The friction model designated in the FAILED MODEL command line must be defined within the CONTACT DEFINITION command block. The default model after failure is the frictionless contact model.

8.3.0.13 Area Weld Model

```
BEGIN AREA WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer> num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [AREA WELD MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The AREA WELD MODEL command block defines a contact friction model that behaves identically to the TIED MODEL until a maximum traction between a node and face in an interaction is reached in the normal direction or the tangential direction. Once this maximum traction is reached, the tied contact "fails" and the friction model switches to a different friction model, as specified by the user. This model is identical to the SURFACE WELD MODEL command block, except that tractions are used instead of forces.

In the above command block, the string name is a user-selected name for this friction model that is used to identify this model in the interaction definitions. The maximum allowed traction in the normal direction is specified by the real value normal_cap in the NORMAL CAPACITY command line. The maximum allowed traction in the tangential direction is specified by the real value tangential_cap in the TANGENTIAL CAPACITY command line. There are no defaults for these values. The area weld will break when either the specified normal or tangential capacity is reached. Once the model fails, the applied tractions decrease to zero over a number of time steps defined through the integer value num_cycles in the FAILURE DECAY CYCLES command line. The default for num_cycles is 1. The friction model that should be used after the weld fails is identified in the FAILED MODEL command line with the string failed_model_name. The friction model designated in the FAILED MODEL command line must be defined within the CONTACT DEFINITION command block. The default model after failure is the frictionless contact model.

8.3.0.14 Adhesion Model

```
BEGIN ADHESION MODEL <string>name
  ADHESION FUNCTION = <string>func_name
  ADHESION SCALE FACTOR = <real>scale_factor(1.0)
END [ADHESION MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The `ADHESION MODEL` command block defines a friction model that behaves like frictionless contact when two surfaces are in contact, but computes an additional force between the surfaces when they are not touching. The value of the additional force is given by a user-specified function of force versus distance, where the distance is the distance between a node and the closest point on the opposing surface.

In the above command block, the string `name` is a user-selected name for this friction model that is used to identify this model in the interaction definitions. The force between surfaces that are not touching is given by the `ADHESION FUNCTION` command line, where the string `func_name` is the name of a function defined in a `DEFINITION FOR FUNCTION` command block in the SIERRA scope. The values of this function are expected to be non-negative. The function can be scaled by the real value `scale_factor` in the `ADHESION SCALE FACTOR` command line; the default for this factor is 1.0. Because contact forces are typically only given to node-face interactions if they touching, the contact search requires appropriate tolerances when this model is used. The normal and tangential tolerances specified in the interaction definitions should be set to the maximum distance at which the adhesion model should be applying force. However, setting this distance to be very large may cause excessive numbers of interactions to be identified in the search phase, causing the contact processing to be very slow and/or generate erroneous interactions.

8.3.0.15 Cohesive Zone Model

```
BEGIN COHESIVE ZONE MODEL <string>name
  TRACTION DISPLACEMENT FUNCTION = <string>func_name
  TRACTION DISPLACEMENT SCALE FACTOR = <real>scale_factor(1.0)
  CRITICAL NORMAL GAP = <real>crit_norm_gap
  CRITICAL TANGENTIAL GAP = <real>crit_tangential_gap
END [COHESIVE ZONE MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The `COHESIVE ZONE MODEL` command block defines a friction model that prevents penetration when contact surfaces are touching, but provides an additional force when the distance between the node and face in an interaction increases. This force is determined by a user-specified function. Once the distance exceeds a user-specified value in the normal direction or the tangential direction, the force is no longer applied. This model can be used to mimic the energy required to separate two surfaces that are initially touching.

In the above command block, the string `name` is a user-selected name for this friction model

that is used to identify this model in the interaction definitions. The displacement function for traction is given by the TRACTION DISPLACEMENT FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command block in the SIERRA scope. This function can be scaled by the real value scale_factor in the TRACTION DISPLACEMENT SCALE FACTOR command line; the default for this factor is 1.0. In the CRITICAL NORMAL GAP command line, the real value crit_norm_gap specifies the normal distance between the node and face past which the cohesive zone no longer provides a force. In the CRITICAL TANGENTIAL GAP command line, the real value crit_tangential_gap specifies the tangential distance between the node and face past which the cohesive zone no longer provides a force.

8.3.0.16 Junction Model

```
BEGIN JUNCTION MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CUTOFF DISTANCE FOR TANGENTIAL TRACTION = <real>distance
END [JUNCTION MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The JUNCTION MODEL command block defines a model that prevents the interpenetration of contact surfaces and that also provides normal and tangential tractions to a node-face interaction when the surfaces are not touching. The normal tractions are defined as a function of the normal distance between the node and face of an interaction, while the tangential traction is given as a function of the relative tangential velocity. The tractions are defined by user-specified functions, and the tangential tractions from this model drop to zero once the normal distance between the node and the face exceeds a critical value. This friction model provides a simple way to model threaded connections, though the THREADED MODEL defined in Section 8.3.0.10 has more flexibility.

In the above command block, the string name is a user-selected name for this friction model that is used to identify this model in the interaction blocks. The normal traction curve is specified by the NORMAL TRACTION FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command block in the SIERRA scope. This function defines a relation between the traction and the distance between the node and the face in the normal direction. This function can be scaled by the real value scale_factor in the NORMAL TRACTION SCALE FACTOR command line; the default for this factor is 1.0. Similarly, the tangential traction curve is specified by the TANGENTIAL TRACTION FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command block in the SIERRA scope. This function defines a relation between the traction and the relative velocity of the node and face in the tangential direction. This function can be scaled by the real value scale_factor in the TANGENTIAL TRACTION SCALE FACTOR command line; the default for this factor is 1.0. Once the normal distance between a node and a face using this model reaches a critical distance, the tangential traction drops to zero; this distance is specified with the

real value `distance` in the `NORMAL CUTOFF DISTANCE FOR TANGENTIAL TRACTION` command line.

8.3.0.17 Threaded Model

```
BEGIN THREADED MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION GAP FUNCTION = <string>func_name
  TANGENTIAL TRACTION GAP SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [THREADED MODEL <string>name]
```

Note: This model is available only in explicit dynamics with ACME contact

The `THREADED MODEL` command block defines a friction model that is designed to mimic a threaded interface. This model prevents interpenetration of contact surfaces, and also supplies additional tractions when the surfaces are not touching. Tensile tractions in the normal direction are given by a user-specified function of force versus distance between the node and face. Tensile tractions in the tangential direction are computed as the product of a traction tangential-displacement curve and a scaling curve that is a function of the normal displacement. Maximum normal and tangential tractions are input such that the model "fails" at a node-face interaction once they are reached. For interactions that include both normal and tangential displacements, the model failure is defined according to a failure criterion defined as the sum of the ratios of the normal and tangential traction components to their maximum capacity values, raised to a power. After failure, interactions shift to a different user-specified friction model.

In the above command block:

- The string `name` is a user-selected name for this friction model that is used to identify the model in the interaction definitions.
- The traction-displacement relation in the normal direction is specified by the `NORMAL TRACTION FUNCTION` command line, where the string `func_name` is the name of a function defined in a `DEFINITION FOR FUNCTION` command block in the SIERRA scope. This function can be scaled by the real value `scale_factor` in the `NORMAL TRACTION SCALE FACTOR` command line; the default for this factor is 1.0.
- The traction-displacement relation in the tangential direction is specified by two curves. The traction-displacement relation in the tangential direction when there is no displacement

in the normal direction is defined by the TANGENTIAL TRACTION FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command block in the SIERRA scope. This function can be scaled by the real value scale_factor in the TANGENTIAL TRACTION SCALE FACTOR command line; the default for this factor is 1.0. When the distance in the normal direction is greater than zero, the tangential traction is scaled by a function specified in the TANGENTIAL TRACTION GAP FUNCTION command line, where the string func_name is the name of a function defined in a DEFINITION FOR FUNCTION command block in the SIERRA scope. This function defines a scaling factor as a function of the normal displacement. The function can be scaled by the real value scale_factor in the TANGENTIAL TRACTION GAP SCALE FACTOR command line; the default for this factor is 1.0.

- The threaded model "fails" once the normal and tangential tractions reach a critical capacity value. The normal capacity is specified by the real value normal_cap in the NORMAL CAPACITY command line. The tangential capacity is specified by the real value tangential_cap in the TANGENTIAL CAPACITY command line. There are no default values for these parameters. These capacities are defined for pure normal or tangential displacements. In cases where there is a combination of tangential and normal displacements, a failure curve is used to determine the combination of tangential and normal tractions that determines model failure. This curve is defined as the sum of the ratios of the normal and tangential traction components to their maximum capacity values, raised to a power. The power in the function is defined by the real value exponent in the FAILURE ENVELOPE EXPONENT command line. The default value of the exponent is 2.0. Once the model fails, the applied tractions decrease to zero over a number of time steps defined through the integer value num_cycles in the FAILURE DECAY CYCLES command line. The default for num_cycles is 1. When the model exceeds the designated capacity, the contact surfaces using this model switch to a different friction model as identified in the FAILED MODEL command line with the string failed_model_name. The friction model designated in the FAILED MODEL command line must be defined within the CONTACT DEFINITION command block. The default model is the frictionless model.

8.3.0.18 PV_Dependent Model

```
BEGIN PV_DEPENDENT MODEL <string>name
  STATIC COEFFICIENT = <real>stat_coeff
  DYNAMIC COEFFICIENT = <real>dyn_coeff
  VELOCITY DECAY = <real>vel_decay
  REFERENCE PRESSURE = <real>p_ref
  OFFSET PRESSURE = <real>p_off
  PRESSURE EXPONENT = <real>p_exp
END [PV_DEPENDENT MODEL <string> name]
```

Note: This model is available only in explicit dynamics with ACME contact

The PV_DEPENDENT MODEL command block defines a friction model similar to a coulomb friction model, but which provides a frictional response that is dependent on the pressure and the

velocity. The pressure-dependent portion of the model behaves similarly to the constant friction model except that the tangential traction is given by

$$\left[\frac{p + \text{p_off}}{\text{p_ref}}\right]^{\text{p_exp}}. \tag{8.1}$$

The velocity-dependent part is given by

$$\left(\text{stat_coeff} - \text{dyn_coeff}\right) e^{\left(-\text{vel_decay}\|v\|\right)} + \text{dyn_coeff}. \tag{8.2}$$

The PV_DEPENDENT MODEL command block multiplies the pressure and velocity effects together.

In the above command block:

- The string name is a name assigned to this friction model that is used to identify the model in the interaction definitions.
- The real value p_ref in the pressure-dependent part given in Equation (8.1) is specified with the REFERENCE PRESSURE command line.
- The real value p_off in the pressure-dependent part given in Equation (8.1) is specified with the OFFSET PRESSURE command line.
- The real value p_exp in the pressure-dependent part given in Equation (8.1) is specified with the PRESSURE EXPONENT command line.
- The real value stat_coeff in the velocity-dependent part given in Equation (8.2) is specified with the STATIC COEFFICIENT command line.
- The real value dyn_coeff in the velocity-dependent part given in Equation (8.2) is specified with the DYNAMIC COEFFICIENT command line.
- The real value vel_decay in the velocity-dependent part given in Equation (8.2) is specified with the VELOCITY DECAY command line.

8.3.0.19 Hybrid Model

```
BEGIN HYBRID MODEL <string>name
  INITIALLY CLOSE = <string>close_name
  INITIALLY FAR   = <string>far_name
END [HYBRID MODEL <string>name]
```

Note: This feature is only available for Dash contact with explicit dynamics, and Dash augmented Lagrange contact for implicit analyses.

The HYBRID MODEL command block defines a single friction model that is a composite of two other models: one for faces that are initially close together, and one for faces that are initially far

apart. The friction law specified in the `INITIALLY CLOSE` command is used for faces that are close together in the initial mesh geometry. The friction law specified in the `INITIALLY FAR` command is used for faces that are far apart in the initial mesh geometry.

A common usage of the hybrid model is to tie together parts of two surfaces that are close together, while allowing the remaining parts of those surfaces to have a sliding contact interaction. If the tied model were used in that situation, the faces that are initially close would be tied together, but no contact would be enforced on the remaining faces on those surfaces, so they could pass through one another. Another potential usage of the hybrid model would be to define a high friction coefficient for faces that are initially close, and then use a lower friction coefficient once the faces start to slide away from the initial configuration.

A hybrid model is automatically created when an interaction-specific and a default friction model are both specified and the interaction-specific friction law is only applicable when the surfaces are initially close together. For example, the contact interaction definition:

```
BEGIN HYBRID MODEL hmod
  INITIALLY CLOSE = tied
  INITIALLY FAR   = frictionless
END

BEGIN INTERACTION
  SURFACES = block_1 block_2
  friction model = hmod
END
```

Is equivalent to:

```
BEGIN INTERACTION DEFAULTS
  FRICTION MODEL = frictionless
END
BEGIN INTERACTION
  SURFACES = block_1 block_2
  friction model = tied
END
```

8.3.0.20 Time Variant Model

```
BEGIN TIME VARIANT MODEL <string>name
  MODEL = <string>model DURING PERIODS <string_list>time_periods
END [TIME VARIANT MODEL]
```

Note: This feature is only available for Dash contact with explicit dynamics, and Dash augmented Lagrange contact for implicit analyses.

The `TIME VARIANT MODEL` command block does not directly define a friction model, but provides a mechanism to switch between separately defined friction models between time periods. The

only command line in the block is the MODEL command, which is used to reference a separately-defined friction model model and provide a list of time periods during which that model is active. Multiple MODEL commands may be used to activate different friction models throughout the analysis. See Section 2.5 for more information on controlling functionality by time period.

8.3.0.21 User Subroutine Friction Models

```
BEGIN USER SUBROUTINE MODEL <string>name
  INITIALIZE MODEL SUBROUTINE = <string>init_model_name
  INITIALIZE TIME STEP SUBROUTINE = <string>init_ts_name
  INITIALIZE NODE STATE DATA SUBROUTINE =
    <string>init_node_data_name
  LIMIT FORCE SUBROUTINE = <string>limit_force_name
  ACTIVE SUBROUTINE = <string>active_name
  INTERACTION TYPE SUBROUTINE = <string>interaction_name
END [USER SUBROUTINE MODEL <string>name]
```

Note: This friction model is only available with explicit ACME contact

The USER SUBROUTINE MODEL command blocks permit contact to use a user subroutine to define a friction model between surfaces. This capability is in a test phase at this time; please contact a Sierra/SM developer for more information.

In this command block:

- The string name is a user-specified name that is used to identify this model in the interaction definitions.
- The command line INITIALIZE MODEL SUBROUTINE specifies a user subroutine to initialize the friction model. The name of the subroutine is given by init_model_name.
- The command line INITIALIZE TIME STEP SUBROUTINE specifies a user subroutine to initialize the time step. The name of the subroutine is given by init_ts_name.
- The command line INITIALIZE NODE STATE DATA SUBROUTINE specifies a user subroutine to initialize the node state data. The name of the subroutine is given by init_node_data_name.
- The command line LIMIT FORCE SUBROUTINE specifies a user subroutine to provide the limit force for the friction model. The name of the subroutine is given by limit_force_name.
- The command line ACTIVE SUBROUTINE specifies a user subroutine to compute forces for an active node-face interaction. The name of the subroutine is given by active_name.
- The command line INTERACTION TYPE SUBROUTINE specifies a user subroutine to define the interaction type. The name of the subroutine is given by interaction_name.

8.4 Definition of Interactions

Each contact surface can interact with one or more other contact surfaces. Contact conditions are enforced between two surfaces by defining what is referred to here as a contact interaction. To define a contact interaction, one must specify which two surfaces should interact through contact, as well as the properties of that interaction. These properties include parameters controlling the search and the model governing the interaction between the surfaces if they are found to be in contact with each other.

Contact interactions can be specified individually between each set of surfaces between which contact is to be enforced. This works well if there are relatively few surfaces in a model and those surfaces interact with a small number of other surfaces. For complex models with many potential contact surfaces, however, individually defining contact interactions between each surface and all of the other surfaces that could potentially interact with it can be a very tedious and error-prone process.

Defining contact in complex models can be greatly simplified by using "general" contact, in which a single command is used to enforce contact between every contact surface in the model and every other contact surface with a set of default parameters for the interaction.

In addition to being enforced between two different surfaces, contact can be enforced between a surface and itself. This is known as "self" contact, and is used to prevent a part of a surface from penetrating another part of that surface when it folds in on itself. As a general rule, enforcing self contact is more expensive than enforcing contact between different surfaces.

Contact surfaces can be based either on sets of nodes, faces, or on analytic surfaces. Node-based surfaces can only interact with face-based or analytic surfaces, and cannot interact with each other. Face-based surfaces can interact with any other type of surface. Analytic surface can only interact with node-based or face-based surfaces, and cannot interact with each other.

The following sections document the commands used to define contact interactions and the properties of those interactions.

8.4.1 Default Values for Interactions

```
BEGIN INTERACTION DEFAULTS
  CONTACT SURFACES = <string list>surface_names
  SELF CONTACT = <string>ON|OFF(OFF)
  GENERAL CONTACT = <string>ON|OFF(OFF)
  AUTOMATIC KINEMATIC PARTITION = <string>ON|OFF(OFF)
  FRICTION MODEL = <string>friction_model_name|TIED|FRICTIONLESS
    (FRICTIONLESS)
  INTERACTION BEHAVIOR = <string>SLIDING|
    INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
  CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
END [INTERACTION DEFAULTS]
```

This section discusses the INTERACTION DEFAULTS command block. This command block enables contact enforcement either on all contact surfaces or on a subset of the contact surfaces. It is also used to set default parameters for contact interactions. Those defaults can be overridden for specific interactions by specifying them separately in INTERACTION blocks.

It is important to note that unless some combination of the INTERACTION DEFAULTS command block and INTERACTION command blocks (Section 8.4.2) exists in the CONTACT DEFINITION command block, enforcement will not take place. Up to this point, all command lines and command blocks have provided information to set up the search phase and have provided details for surface interaction. However, contact enforcement for surfaces—the actual removal of interpenetration between surfaces and the calculation of surface forces consistent with friction models—will not take place unless some combination of the INTERACTION DEFAULTS command block and INTERACTION command blocks is used to set up surface interactions.

Contact between surfaces requires data to describe the interaction between these surfaces. You may specify defaults for the surface interactions for some or all surface pairs by using the INTERACTION DEFAULTS command block. Within this command block, you can provide a list of surfaces that are a subset of the contact surfaces. Any pair of surfaces listed in the INTERACTION DEFAULTS command block will acquire the default values that are defined within the INTERACTION DEFAULTS command block. If you omit the CONTACT SURFACES command line, defaults in the INTERACTION DEFAULTS command block are applied to all surfaces. Any default set within an INTERACTION DEFAULTS command block can be overridden by commands in an INTERACTION command block. See Section 8.4.2.

If you consider only the use of the INTERACTION DEFAULTS command block (and not the use of the INTERACTION command block), you have three options for the surface interaction values:

- You can specify default surface interaction values for all the contact surface pairs by specifying all the contact surfaces in an INTERACTION DEFAULTS command block.
- You can specify default surface interaction values for some of the contact surface pairs by specifying a subset of the contact surfaces in an INTERACTION DEFAULTS command block.
- You can leave all interactions off by default by not specifying an INTERACTION DEFAULTS command block.

The values specified by the command lines in the INTERACTION DEFAULTS command block are applied by default to all interaction contact surfaces unless overridden by a specific interaction definition. Note that if a friction model is specified for a specific interaction, but that friction model is relevant only for some of the faces in the surfaces involved in the interaction, the friction model specified in the interaction defaults will be used for the remaining faces. For example, if a tied model is specified for an interaction, only the faces that are initially close will be tied. The remaining faces will be enforced using the default friction model. See Section 8.3.0.19 for more information.

8.4.1.1 Surface Identification

```
CONTACT SURFACES = <string list>surface_names
```

This command line identifies the contact surfaces to which the surface interaction values defined in the INTERACTION DEFAULTS command block will apply. The string list on the CONTACT SURFACES command line specifies the names of these contact surfaces. The CONTACT SURFACES command line can include any surface specified in a CONTACT SURFACE command line, a CONTACT SURFACE command block, or a SKIN ALL BLOCKS command line.

The SURFACES command line is optional. If you want the defaults to apply to all the surfaces you have defined, you will NOT use the SURFACES command line in this command block. If you want the defaults to apply to a subset of all contact surfaces, then you will list the specific set of surfaces on a SURFACES command line. The names of all the surfaces with the default values will be listed in the string list designated as surface_names.

8.4.1.2 Self-Contact and General Contact

```
SELF CONTACT = <string>ON|OFF(OFF)
GENERAL CONTACT = <string>ON|OFF(OFF)
```

The SELF CONTACT command line, if set to ON, specifies that the default values set in the command lines of the command block will apply to self-contact between the listed surfaces (or all surfaces if no surfaces are listed). The GENERAL CONTACT command line, if set to ON, specifies that the default values set in the command lines of this command block apply to contact between the listed surfaces (or all surfaces if no surfaces are listed) excluding self-contact. The default values for both of these command lines is OFF. The default values for this command line is OFF. To enforce general contact between all surfaces specified in the INTERACTION DEFAULTS command block but no self-contact, this line must be present:

```
GENERAL CONTACT = ON
```

To enforce self-contact for all surfaces specified in the INTERACTION DEFAULTS command block, this line must be present:

```
SELF CONTACT = ON
```

If no individual contact interactions have been specified with INTERACTION command blocks, contact will only be enforced if either general contact or self-contact (or both) are enabled.

If general contact or self-contact is enabled, contact enforcement may be disabled for individual surfaces (for self-contact) or pairs of surfaces (for general contact) by using the INTERACTION BEHAVIOR = NO_INTERACTION in the INTERACTION command block.

8.4.1.3 Friction Model

```
FRICTION MODEL = <string>friction_model_name|TIED|FRICTIONLESS
  (FRICTIONLESS)
```

The FRICTION MODEL command line permits the description of how surfaces interact with each other using a friction model defined in a friction-model command block (see Section 8.3). In the above command line, the string friction_model_name should match the name assigned to some friction model command block. For example, if you specified the name of an AREA WELD command block as AW1 and wanted to reference that name in the FRICTION MODEL command line, the value of friction_model_name would be AW1.

The default interaction is frictionless contact.

8.4.1.4 Automatic Kinematic Partition

```
AUTOMATIC KINEMATIC PARTITION
```

Explicit ACME only capability

If the AUTOMATIC KINEMATIC PARTITION command line is used, Sierra/SM will automatically compute the kinematic partition factors for pairs of surfaces. (See Section 8.4.2.2 for more information on kinematic partitioning.) The automatic kinematic partitions are computed from the impedance of each surface based on nodal average density and wave speed. Automatic computation of kinematic partition factors provides the best approach to exact enforcement of symmetric contact of opposing surfaces provided that these surfaces have the same mesh resolution. If the mesh resolution is disparate, you can specify the coarser meshed body as master, but generally it is better to use more contact iterations to deal with this case of a fine mesh contacting a coarse mesh.

For the interaction of any two surfaces, the sum of the partition factors for the surfaces must be 1.0. This is automatically taken care of when the AUTOMATIC KINEMATIC PARTITION command line is used. The default value for kinematic partition factors for all surfaces is 0.5.

The AUTOMATIC KINEMATIC PARTITION command line can be used to set the kinematic partitions for all interactions or to set the kinematic partitions for specific interactions. Thus the command line can appear in two different scopes:

1. The command line can be used within the INTERACTION DEFAULTS command block. In this case, all contact surface interactions defined in the command block will use the automatic kinematic partitioning scheme by default. This will override the default case that assigns a kinematic partition factor of 0.5 to all surfaces. For particular interactions, it is possible to override the use of the automatic kinematic partition factors by specifying kinematic partition values (with the KINEMATIC PARTITION command line) within the INTERACTION command blocks for those interactions.

2. The command line can be used inside an INTERACTION command block. If the automatic partitioning command line appears inside an INTERACTION command block, the kinematic partition factors for that particular interaction will be calculated by the automatic kinematic partition scheme.

Automatic kinematic partitioning is not currently operational for shell elements. If it is enabled, the kinematic partitioning factor is set to 0.5 for all contact interactions involving shell elements.

8.4.1.5 Interaction Behavior

```
INTERACTION BEHAVIOR = <string>SLIDING|
  INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
```

The INTERACTION BEHAVIOR command line specifies how the search will be done. For SLIDING contact, the search algorithm is constantly updating information that lets the code accurately track the sliding of the node over the face and any adjacent faces. The SLIDING option, which is the default, lets us handle the case where we have large relative sliding between a face and a node. A node contacting a face can slide over time by a significant amount over the face. The node can slide onto an adjacent face or onto a face on a nearby surface. For the case of INFINITESIMAL_SLIDING, search information is not updated to the extent that it is with the SLIDING option. In the case of INFINITESIMAL_SLIDING, it is assumed that there is very little slip over time of a node relative to its initial contact point on a face. Furthermore, it is assumed a node will not slide off the face that it initially contacts. The INFINITESIMAL_SLIDING option is not as accurate as the SLIDING option, but neither is it as expensive as the SLIDING option. For some cases, however, the INFINITESIMAL_SLIDING option may work quite well even though it is not as accurate as the SLIDING option. Finally, you may turn off the search completely by using the NO_INTERACTION option.

With the third option, NO_INTERACTION, you could turn off the search for all surfaces specified in the INTERACTION DEFAULTS command block. You could then turn on the search on a case-by-case basis for various contact pairs or for the self-contact of surfaces by using INTERACTION command blocks. This is a convenient way to set defaults for the friction model and automatic kinematic partitioning without turning on all the interactions. More likely, you will set contact interactions to default to the SLIDING option in the INTERACTION DEFAULTS command block, and then turn off specific contact interactions through INTERACTION command blocks.

Using the INTERACTION BEHAVIOR command line in the INTERACTION DEFAULTS command block represents a sophisticated application of this command line. Consult with Sierra/SM de-

velopers for more information about this command line used in an INTERACTION DEFAULTS command block.

8.4.1.6 Constraint Formulation

```
CONSTRAINT FORMULATION = NODE_FACE|FACE_FACE
```

The CONSTRAINT FORMULATION command is used to switch between node/face and face/face enforcement for contact constraints. If the NODE_FACE option is selected, node/face contact is used, and if the FACE_FACE option is selected, face/face contact is used. This command only has an effect when Dash contact is used. ACME only supports node/face contact. With Dash contact, face/face contact is used by default, except for the case where one of the surfaces is explicitly defined by a node set, in which case node/face contact is used.

8.4.2 Values for Specific Interactions

```
BEGIN INTERACTION [<string>name]
  SURFACES = <string_list>surfaces [EXCLUDE <string_list>surfaces]
  MASTER   = <string_list>surfaces [EXCLUDE <string_list>surfaces]
  SLAVE    = <string_list>surfaces [EXCLUDE <string_list>surfaces]
  #
  # Tolerance Commands
  CAPTURE TOLERANCE = <real>cap_tol
  NORMAL TOLERANCE = <real>norm_tol
  TANGENTIAL TOLERANCE = <real>tang_tol
  CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
  FRICTION MODEL = <string>friction_model_name|TIED|FRICTIONLESS
    (FRICTIONLESS)
  FACE MULTIPLIER = <real>face_multiplier(0.1)
  OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
  OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
  KINEMATIC PARTITION = <real>kin_part
  AUTOMATIC KINEMATIC PARTITION
  INTERACTION BEHAVIOR = <string>SLIDING|
    INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
  PUSHBACK FACTOR = <real>pushback_factor(1.0)
  TENSION RELEASE = <real>ten_release
  TENSION RELEASE FUNCTION = <string>ten_release_func
  #
  # Kinematic Enforcement Only.
  FRICTION COEFFICIENT = <real>coeff
  FRICTION COEFFICIENT FUNCTION = <string>coeff_func
END [INTERACTION <string>name]
```

The Sierra/SM contact input also permits the setting of values for specific interactions using the `INTERACTION` command block. If an `INTERACTION DEFAULTS` command block is present within a `CONTACT DEFINITION` command block, the values provided by an `INTERACTION` command block override the defined defaults. If an `INTERACTION DEFAULTS` command block is not present, only those interactions defined by `INTERACTION` command blocks are searched for contact, and values without system defaults must be specified.

The `INTERACTION` command block begins with:

```
BEGIN INTERACTION [<string>name]
```

and ends with:

```
END [INTERACTION <string>name]
```

where `name` is a name for the interaction. Note that this name is currently used only for informational output purposes and is not required.

The valid commands within an INTERACTION command block are described in Section 8.4.2.1 through Section 8.4.2.7. Section 8.4.2.11.

8.4.2.1 Surface Identification

```
#
# preferred for explicit and implicit face/face contact
SURFACES = <string_list>surfaces [EXCLUDE <string_list>surfaces]
#
# preferred for implicit node/face enforcement
MASTER   = <string_list>surfaces [EXCLUDE <string_list>surfaces]
SLAVE    = <string_list>surfaces [EXCLUDE <string_list>surfaces]
```

There are two methods to identify the surfaces described by a specific interaction.

One method is to identify all surfaces in a single line with the SURFACES command line. For explicit dynamics, the surface list must contain at least two surfaces. This command will define a symmetric two-way contact interaction between each surface on the command line and every other surface on the command line. If a contact KINEMATIC PARTITION is also specified, the surface list must contain only two surfaces (see Section 8.4.2.2).

To specify self contact, the surface must appear in the SURFACES command line twice. Each surface specified on the SURFACES command line interacts with each other surface on that line. If a surface is specified on the line twice, it indicates that a surface will also interact with itself.

The second method to identify contact surfaces is to use the MASTER and SLAVE command lines. In Sierra/SM, contact surfaces must be identified using the MASTER and SLAVE command lines for kinematic enforcement. For augmented Lagrange enforcement, either method of identifying the surfaces may be used.

The nodes of the slave surfaces are searched against the faces of the master surfaces. Each of these command lines takes as input a list of names of contact surfaces defined in the contact block. A master slave interaction will be defined between each surface in the master list and each surface in the slave list. A surface may not be present in both the master and the slave list.

Master/Slave contact always uses a kinematic partition of 1.0 thus the KINEMATIC PARTITION command may not be used with master/slave contact.

The surface named all_surfaces is a special reserved word that is equivalent to typing all contact surfaces known by the contact block into the string list. Optionally, the EXCLUDE keyword can be placed on the command line and followed by a list of surface names to exclude from the list. If the SURFACES, MASTER, or SLAVE command lines appear multiple times within a CONTACT INTERACTION block, their surface lists will be concatenated. The effect is equivalent to specifying all of the surface names on a single line. The SURFACES command line cannot be used simultaneously with the MASTER and SLAVE command lines.

The following examples demonstrate ways to identify contact surfaces involved in an interaction:

This command defines a symmetric interaction between s1 and s2:

```
SURFACES = s1 s2
```

This defines a symmetric self contact interaction between `s1` and itself:

```
SURFACES = s1 s1
```

This defines a set of symmetric interactions between `s1` and `s2`, `s1` and `s3`, `s2` and `s3`:

```
SURFACES = s1 s2 s3
```

This defines a full set of symmetric interactions between `s1` and itself, `s2` and itself, `s1` and `s2`:

```
SURFACES = s1 s1 s2 s2
```

This command defines a set of interactions between all pairs of defined surfaces in the model, with the exception of surfaces `s7` and `s8`, which have no interactions:

```
SURFACES = all_surfaces exclude s7 s8
```

This command defines a set of interactions between all pairs of defined surfaces in the model, as well interactions between each surface and itself, with the exception of surfaces `s7` and `s8`, which have no interactions:

```
SURFACES = all_surfaces all_surfaces exclude s7 s8
```

These commands define a one-way interaction between the nodes of `s1` and the faces of `m1`:

```
MASTER = m1
SLAVE = s1
```

These commands define a set of one-way interactions between the nodes of `s1` and the faces of `m1`, the nodes of `s1` and the faces of `m2`, the nodes of `s2` and the faces of `m1`, the nodes of `s2` and the faces of `m2`.

```
MASTER = m1 m2
SLAVE = s1 s2
```

These commands define that the nodes of surface `s1` are slaved to all other contact faces in the contact definition block.

```
MASTER = all_surfaces exclude s1
SLAVE = s1
```

8.4.2.2 Kinematic Partition

```
KINEMATIC PARTITION = <real>kin_part
```

Used only by explicit acme contact

To provide accurate contact evaluation, Sierra/SM typically computes two-way contact between two surfaces, where interactions are defined between the nodes of the first surface and the faces of the second surface, and also between the nodes of the second surface and the faces of the first surface. If the two surfaces have penetrated each other by a distance δ, then each of the contact evaluations will compute forces to move the surface a distance δ, so that the total resulting displacement would be 2δ if both sets of contact computations were fully applied. The `KINEMATIC PARTITION` command line defines a kinematic partition scaling factor, `kin_part`, for the two contact computations so that the total contact motion is correct. The kinematic partition factor, `kin_part`, is a value between 0.0 and 1.0. The factor scales the relative motion of the first surface, where `kin_part` = 0.0 means the first surface will move none of δ, while `kin_part` = 1.0 means the surface moves all of δ. The second surface moves the portion of δ that remains after the motion of the first surface (i.e., δ for second surface = 1.0 – `kin_part`). For instance, if `kin_part` is 0.2, the first surface will move 20% of the penetration distance (0.2δ in our example), and the second surface would move the remaining 80% (0.8δ in our example). The default value is 0.5, so that each surface would move half of the penetration distance. If `kin_part` is 0.0, the first surface does not move at all, and the second surface moves the full distance. This is exactly equivalent to a one-way master-slave contact definition, where the first surface is the master and the second is the slave. If `kin_part` is 1.0, the second surface is the master, and the first surface is the slave. Figure 8.8 illustrates how the kinematic partition factor varies from 0.0 to 1.0, with the specific example of `kin_part` being set to 0.2.

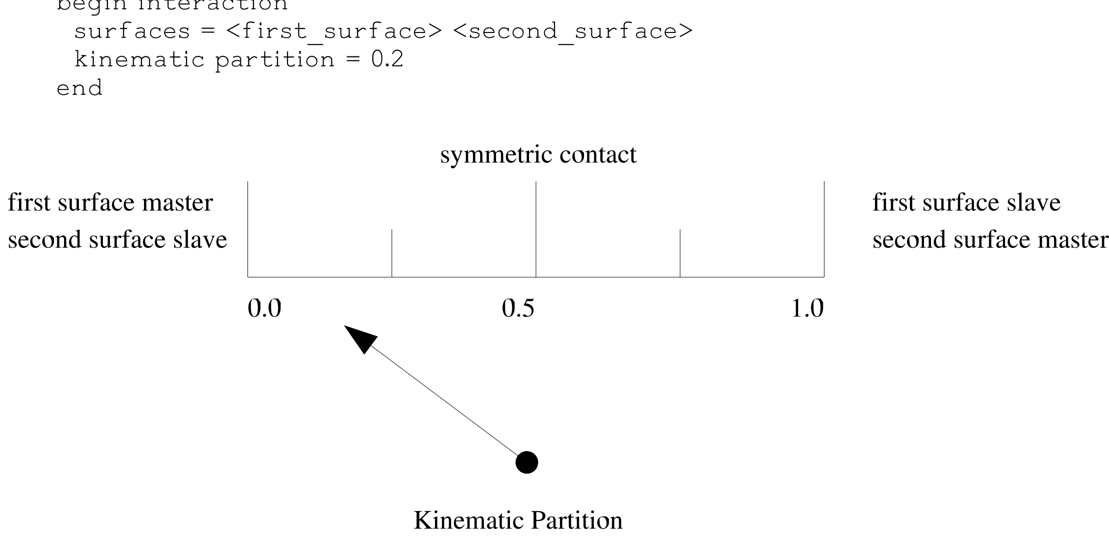


Figure 8.8: Illustration of kinematic partition values

The capability provided by the KINEMATIC PARTITION command line is important in cases where contact occurs between two materials of disparate stiffness. Physically, we would expect a material with a higher stiffness to have more of an effect in determining the position of the contact surface than a more compliant material. In this case, we want the softer material to move more of the distance, and thus it should have a higher kinematic partition factor. The appropriate kinematic partition factor can be determined in closed form; see the ACME contact library reference [1] for more information. Alternately, the AUTOMATIC KINEMATIC PARTITION capability can automatically calculate the proper kinematic partition based on the stiffness of the materials.

Another case where the kinematic partition factor has traditionally been used is when meshes with dissimilar resolutions contact each other. If an interaction is defined with a fine mesh as the master surface and a coarse mesh as a slave surface, the contact algorithms will permit nodes on the master surface to penetrate the slave surface. In these cases, such problems can be alleviated by making the coarse mesh the master surface. However, the iterative approach implemented in the enforcement can also take care of this problem and is advised. If very few iterations are chosen, an appropriate kinematic partition factor may be needed to prevent unintentional penetration due to mesh discretization.

For self-contact, the kinematic partition factor should be 0.5.

A kinematic partition factor cannot be defined for interactions that use the pure master-slave syntax (see Section 8.4.2.1).

In general, it is best to use the automatic kinematic partition option to properly compute the kinematic partition for a pair of surfaces. However, in a few cases, master-slave interactions are preferred. These cases consist of (1) interaction between an analytic surface and a deformable body, where the analytic body should be the master surface; and (2) contact between shells and solids, where the solid should be the master surface.

8.4.2.3 Tolerances

Implicit

```
CAPTURE TOLERANCE = <real>cap_tol
NORMAL TOLERANCE = <real>norm_tol
TANGENTIAL TOLERANCE = <real>tang_tol
FACE MULTIPLIER = <real>face_multiplier(0.1)
OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
```

You can set tolerances for the interaction for a specific contact surface pair or for self-contact of a surface by using the above tolerance-related command lines in an INTERACTION command block. See Section 8.5.9.1 on search tolerances and Section 8.5.4 on overlap tolerances for a complete discussion of tolerances for contact.

As indicated previously, the contact functionality in Sierra/SM uses a box defined around each face to locate nodes that may potentially contact the face. This box is defined by a tolerance normal to the face and another tolerance tangential to the face (see Figure 8.14). The code adds to these tolerances the maximum motion over a time step when identifying interactions. In the above command lines, the parameter norm_tol is the normal tolerance (defined on the NORMAL

TOLERANCE command line) for the search box and the parameter tang_tol is the tangential tolerance (defined on the TANGENTIAL TOLERANCE command line) for the search box.

The CAPTURE TOLERANCE, which should be no larger than the NORMAL TOLERANCE, is used to determine which slave nodes near a master surface should be pulled to the master surface and considered for contact. (This applies to slave nodes which have not penetrated the master surface. Slave nodes that have penetrated the master surface will be pushed to the surface regardless of the CAPTURE TOLERANCE.) If later checks determine that a slave node that was pulled to the surface (because it was nearer the surface than the CAPTURE TOLERANCE value) is in fact in tension and should be released, that slave node will not be considered as a potential contact node again during the load step as long as the node remains within the NORMAL TOLERANCE.

8.4.2.4 Friction Model

```
FRICTION MODEL = <string>fric_model_name|TIED|FRICTIONLESS
  (FRICTIONLESS)
```

You can set the friction model for the interaction for a specific contact surface pair or for self-contact of a surface by using the above command line in an INTERACTION command block. See Section 8.4.1.3 for a discussion of this command line.

8.4.2.5 Interface Material

```
INTERFACE MATERIAL = <string>int_matl_name
  MODEL = <string>int_model_name
```

Any of the cohesive zone models available for use with interface elements (see Section 5.3) can be used as interface models within contact. Interactions between faces are created by a contact search performed in the initial configuration (similar to tied contact), so cohesive zones are only created between faces that are initially contacting. The relative motion of the faces on opposing sides of an interface is governed by the behavior of the cohesive zone model.

Note that these interface models can only be used with Dash contact. Additionally, due to the need to store state variables, these models can only be used on interfaces that use node-face contact interactions.

The stiffness of the compliant joint model changes over time based on the loading history of the joint. This stiffness is not explicitly taken into account by the critical time step estimate. Thus, if the critical time step of the contact joint is lower than the critical time step used by the analysis, the analysis may become unstable. If this occurs, the user will need to either manually decrease the integration time step used by the code or change parameters in this model to decrease the stiffness of the joint.

8.4.2.6 Automatic Kinematic Partition

```
AUTOMATIC KINEMATIC PARTITION
```

You can turn on (or off) automatic kinematic partitioning for a specific contact surface pair by using the above command line in an INTERACTION command block. See Section 8.4.1.4 for a discussion of automatic kinematic partitioning.

8.4.2.7 Interaction Behavior

```
INTERACTION BEHAVIOR = <string>SLIDING|
  INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
```

You can set the search behavior for a specific contact surface pair or for self-contact of a surface by using the above command line in an INTERACTION command block. See Section 8.4.1.5 for a discussion of this command line.

A particular use of this command line in this particular command block is to set the interaction behavior to NO_INTERACTION. This deactivates enforcement between the surfaces specified in the INTERACTION command block.

8.4.2.8 Constraint Formulation

```
CONSTRAINT FORMULATION = NODE_FACE|FACE_FACE
```

The CONSTRAINT FORMULATION command is used to switch between node/face and face/face enforcement for contact constraints. See Section 8.4.2.8 for a discussion of this command line.

8.4.2.9 Pushback Factor

```
PUSHBACK FACTOR = <real>pushback_factor
```

Implicit contact feature.

The command line PUSHBACK FACTOR can be used to set the fraction of the gap to be removed in a contact model problem. The default value is 1.0 which removes the entire gap in one contact model problem. Setting the pushback factor to 0.25 will result in the gap being removed in 4 contact model problems.

8.4.2.10 Tension Release

```
TENSION RELEASE = <real>ten_release
```

The command line TENSION RELEASE can be used to set a traction threshold below which slave nodes that have come into contact with master faces will not be released. When this value is set and a slave node is in tension, it will only be allowed to pull away from the master surface if the slave node's traction is greater than the TENSION RELEASE tolerance.

8.4.2.11 Tension Release Function

```
TENSION RELEASE FUNCTION = <real>ten_release_func
```

The command line TENSION RELEASE FUNCTION provides a way for the tension release threshold to be set using a function from the input file. If the TENSION RELEASE line command is also present, the threshold used by the code will be the TENSION RELEASE value multiplied by the value obtained from the function.

8.4.2.12 Friction Coefficient

```
FRICTION COEFFICIENT = <real>coeff
```

You can set the coefficient of friction for the interaction for a specific contact surface pair by using the above command line in an INTERACTION command block.

8.4.2.13 Friction Coefficient Function

```
FRICTION COEFFICIENT FUNCTION = <real>coeff_func
```

The command line FRICTION COEFFICIENT FUNCTION provides a way for the friction coefficient to be set using a function from the input file. If the FRICTION COEFFICIENT line command is also present, the friction coefficient used by the code will be the FRICTION COEFFICIENT value multiplied by the value obtained by the function.

8.4.3 Interaction Behavior for Particle Element Blocks

Element blocks containing particle elements (SPH, Mass Particle, or peridynamics) undergo special treatment, as described below.

Because the SPH and peridynamics element algorithms already define interactions between those elements, it is typically not desirable to enforce self-contact within those element blocks. Self-contact is thus disabled by default for SPH and peridynamics blocks.

In addition, contact between blocks of SPH elements is also disabled by default. The SPH algorithm contains its own contact-like algorithm to govern interactions between SPH particles, even in different element blocks.

Elements that are converted to particles are placed in a new element block. The name of the new element block is the name of the parent block, with `_particles` appended. For example, the particles created from `block_1` are placed in a new element block named `block_1_particles`. The `SKIN ALL BLOCKS` command creates a new contact surface containing the particles that has the same name as the particle block (`block_1_particles` in this example). By default, the contact surface for the particles has the same interactions as does the parent block, subject to the exceptions listed above for interactions between particle blocks.

Any of the above defaults can be overridden by defining individual interactions. For example, if `block_1` and `block_2` are SPH particle blocks, they will not interact though contact by default. However, the following command block can be included to cause contact to be enforced between them.

```
BEGIN INTERACTION
  SURFACES = block_1 block_2
END
```

If a single contact surface is defined by combining multiple types of objects, the special particle-specific default behavior described in this section is disabled. For example, the following command block will turn on self-contact for all entities within the surface `s1`, including between the particles in `block_2_particles`.

```
BEGIN INTERACTION DEFAULTS
  SELF CONTACT = ON
END
CONTACT SURFACE s1 CONTAINS block_1 block_2_particles surface_3
```

8.5 Contact Algorithm Options

8.5.1 Contact Library

```
SEARCH = ACME|DASH(ACME)
```

Dash and ACME are two separate search and enforcement algorithms for defining explicit dynamics or implicit contact. Generally any existing ACME or Dash contact command block may be converted to a Dash or ACME command block simply by using this one command. However, Dash will ignore some ACME options and ACME will ignore some Dash options.

8.5.1.1 How Dash and ACME differ

Dash offers a facet-based enforcement algorithm. By default, Dash enforces face-face constraints. With face-face constraints, contact attempts to compute the total volume of overlap between two facet sets and remove the overlap. Dash uses a reduced order area integration scheme for calculation and removal of volumetric overlaps. Both ACME and Dash offer a node-face based enforcement algorithm. In node-face contact enforcement only the nodes of one body are kept from penetrating the faces of another body.

In some cases, face-face constraints are more robust than node-face constraints and can more accurately solve certain classes of problems. Face-face constraints often work better than node-face for problems with high loading rates and stiffening materials. Node-face contact enforcement tends to aggravate hourglass modes when used on these types of problems. Additionally, problems in which tolerancing issues, multiple interaction issues, or topology restrictions cause node-face constraints to perform poorly may be run more robustly with face-face constraints.

For frictional behavior, however, face-face constraints are generally less accurate than node-face. Face-face constraints are less able to exactly capture the transition from sticking to slipping behavior and other such frictional details than is node-face.

Dash is designed to work correctly for a wide variety of problems with minimal user intervention. It is generally recommended that Dash be used with a minimal set of inputs, for example `skin all blocks`, `self contact = on`, and/or `general contact = on`, and definition of the needed interaction friction models. Dash ignores tangential tolerances and has no kinematic partition option. For more details (see discussion in Section 8.5.1.2).

For explicit contact, Dash uses an iterative augmented Lagrange stiffness enforcement scheme. Dash enforcement explicitly balances momentum by always applying equal and opposite forces. In contrast, ACME uses an iterative kinematic enforcement scheme. The kinematic enforcement scheme can more exactly remove gaps, but may not exactly satisfy force balance at a contact interface. With sufficient iterations, both the Dash and ACME enforcement schemes will converge to the same results.

For implicit contact, the Dash contact search can be used with either kinematic or augmented Lagrange enforcement. When Dash is used with the implicit kinematic enforcement algorithm,

only node-face interactions are enforced, and the default behavior is changed from face-face to node-face.

Dash represents lofted structural elements differently than ACME. Dash can only represent shells via lofted geometry and has no option to treat shells as if they have no thickness for contact. Dash adds the edge faces to lofted shells, allowing edge and edge to face contact to function correctly on shell elements. Dash also handles beam on beam contact via lofted geometry. Beam objects are turned into representative hexagonal prisms that have roughly the same cross sectional area that the beam elements have. Beam contact is functional for edge-on or end-on cases.

These differences in the way structural elements are treated for contact make Dash significantly more capable than ACME in this area. ACME cannot treat any contact between shell edges and other entities, and cannot handle any contact with beam edges.

8.5.1.2 Current Dash Usage Guidelines

Use Dash for remeshing problems. ACME does not perform well on changing topologies. Dash is more robust on problems with element death. ACME often must artificially kill a large number of additional elements other than those which have reached a user defined death criteria. ACME requires these additional element removals to work around certain surface topology restrictions. Dash has no topology restrictions so does not need to kill any extra elements to maintain a valid contact topology.

When performing contact involving shells, beams, and particles, Dash turns the elements into lofted volumes. ACME is able to loft shells and particles, but cannot loft beams.

Both ACME and Dash support basic friction models (tied, Coulomb friction, frictionless, etc.) However, the other friction models (spring weld, surface weld, area weld, adhesion, cohesive zone, junction, threaded, pv_dependent, and user subroutine) are not yet available for use with Dash.

Prefer using block skinning with Dash, and avoid defining contact surfaces based on side sets. Dash uses ray tracing algorithms to determine if certain contact points are inside or outside of bodies. If the surfaces used for contact with Dash do not define fully enclosed bodies, it is difficult for the algorithm to determine wither a point is inside or outside a body.

Dash accepts the same surface definitions as does ACME. Dash also uses identical syntax for interaction definition. The relevant contact block options for Dash are:

8.5.2 Dash Specific Options

```
BEGIN DASH OPTIONS
  INTERACTION DEFINITION SCHEME = EXPLICIT|AUTOMATIC (AUTOMATIC)
  SEARCH LENGTH SCALING = <real>scale(0.15)
  ACCURACY LEVEL = <real>accuracy(1.0)
  SUBDIVISION LEVEL = <int>sublevel(0)
END
```

The `DASH OPTIONS` command block contains a few options unique to Dash. Generally Dash is designed to run accurately with as few user specified parameters as possible, however, some extra options are available.

The `INTERACTION DEFINITION SCHEME`, command controls how contact interactions are defined. By default all Dash interactions are one sided. As the full volume of overlap is removed there is really no concept of master face or slave face. By default Dash automatically picks an optimal enforcement order for each interaction pair based on face size. However, the ability exists to explicitly use the potentially symmetric interactions defined in the input deck by supplying the `EXPLICIT` option to this command. This may allow better enforcement in some circumstances, particularly tied contact.

The `SEARCH LENGTH SCALING` how Dash computes automatic contact tolerances. The search tolerance is a multiplier on the face characteristic length. If faces move farther than the factor supplied in the `SEARCH LENGTH SCALING` command line times the face characteristic length in a single time step, contact could be lost. Thus, extremely high velocity contacts may require use of higher values of this parameter. Note that the default values used should be sufficient for impact speeds below the material sound speed. This parameter can set be set to zero to force the contact algorithm use the user provided search normal tolerances rather than internally computed automatic tolerances.

The `ACCURACY LEVEL` command provides a single dial to change various numerical parameters in the code related to the trade off between algorithmic computation time and algorithm accuracy. The default value is one, large values will tend to increase accuracy while increasing computation time and lower values will tend to decrease accuracy while decreasing computation time. At higher levels of accuracy the contact algorithm will allow less material interpenetration to occur, provide more accurate iterations of frictional forces, and be less sensitive to numerical noise and error.

The `SUBDIVISION LEVEL` command provides a means to force a slave triangular face to be subdivided. Each quadrilateral face is originally split into four triangular faces. A subdivision level splits each tri-face edge into `sublevel` +1 segments. An increased subdivision level give a more accurate solution because the overlap volume is more accurately calculated. This increased accuracy, however, comes with an increased computational cost.

8.5.3 Enforcement

```
ENFORCEMENT = <string>AL|KINEMATIC(AL)
```

The ENFORCEMENT command line controls whether kinematic or Augmented Lagrange (AL) enforcement will be used in the current CONTACT DEFINITION block.

The AL option enforces contact using an Augmented Lagrange algorithm. This option can be used only in conjunction with the Dash search, which is enabled using the SEARCH = DASH command (See Section 8.5.1). and is the default option when using the Dash search. The FRICTION MODEL commands are necessary with this command and varying contact models can be enforced within the same CONTACT DEFINITION block.

If kinematic enforcement is enabled (by using the KINEMATIC option), the enforcement algorithm and the friction model are linked, and only one friction model can be used per CONTACT DEFINITION block. If a combination of tied, frictionless, and frictional contact is to be enforced, multiple CONTACT DEFINITION blocks must be included in the input file.

8.5.4 Remove Initial Overlap

```
BEGIN REMOVE INITIAL OVERLAP
  OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
  OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
  SHELL OVERLAP ITERATIONS = <integer>max_iter(10)
  SHELL OVERLAP TOLERANCE = <real>shell_over_tol(0.0)
END REMOVE INITIAL OVERLAP
```

Note: Explicit dynamics only capability.

Meshes supplied for finite element analyses frequently have some level of initial mesh overlap, where finite element nodes rest inside the volume of other elements. These overlaps may cause initial forces that are nonphysical and produce erroneous stress waves and energy. Initial overlap removal is a mechanism to modify the initial mesh to attempt to remove overlaps in surfaces defined for contact via the `REMOVE INITIAL OVERLAP` command block.

The process used to remove the initial overlap for three-dimensional solid elements involves changing the original coordinates of nodes on contact surfaces. Changing the coordinates yields a new mesh with the overlap removed; the overlap removal adds no initial stresses. Normal and tangential tolerances may specified by the user for all the contact surfaces in the `REMOVE INITIAL OVERLAP` command block. It is also possible to specify overlap normal and tangential tolerances on each surface pairing separately in the `INTERACTION` command block. In other words, overlap removal tolerances specified in `INTERACTION` command blocks will overwrite the tolerances specified in the `REMOVE INITIAL OVERLAP` command block. See Section 8.4.2 for details. The `REMOVE INITIAL OVERLAP` command block only removes overlaps that are detected among the surfaces defined for contact, not all surfaces in the mesh.

Overlap tolerances are used to designate a box around each surface pair to search for overlaps. If the amount of overlap in the mesh is larger than the box defined by those tolerances then the overlap will not be found and not be removed. However, if the specified tolerances are larger than an element length in the analysis, the overlap removal mechanism may invert elements, leading to analysis failure. This has two ramifications. First, the tolerances must be carefully specified to correct mesh overlaps and to not invert elements. Second, this mechanism is unable to remove initial overlaps that are greater than an element length. In such cases, the overlap must be removed manually by modifying the mesh. The mesh modification done by the `REMOVE INITIAL OVERLAP` feature changes the meshed geometry, and thus can change the mass and time step of affected elements. The mesh returned in the results file includes the changed coordinates and should be checked to ensure that the modifications are acceptable. A summary of the overlap that is removed is reported in the log file. (See Section 1.7 for a discussion of the log file.) The log file lists each block in which the initial overlap has been removed as well as the maximum amount of overlap removed on any node for each of these blocks. Additionally, you can request that a nodal variable called `REMOVED_OVERLAP` be written to the results file. See Section 9.2.1.1 for a discussion of the output of nodal variables to the results file.

ACME Only: For contact with lofted geometry, a slightly different approach can be used. Because the thickness of a shell must be preserved when shell lofting is requested, removing the initial overlap between nested shells becomes an iterative process whereby shell locations are adjusted to

remove the overlap. This process is approximate and may not remove all the overlap in all cases. It is advised to check the corrected mesh to make sure that the mesh modifications are acceptable. In the input, two additional input lines, `SHELL OVERLAP ITERATIONS` and `SHELL OVERLAP TOLERANCE`, may be needed to properly remove the initial overlap.

Note, if automatic kinematic partitioning is being used, the overlap algorithm will use the symmetric kinematic partition value of 0.5.

- ACME Only: The `SHELL OVERLAP ITERATIONS` command line controls the maximum number of iterations that will be used by the overlap removal mechanism to resolve nested shells. By default, the value of `max_iter` is 10. If the mesh has only a few layers of shells that may overlap, a value of 10 should suffice. However, if the mesh has a number of layers of shells that may overlap, this value may need to be larger.
- ACME Only: The `SHELL OVERLAP TOLERANCE` command line specifies an amount of overlap that is permitted to be left in the shell elements. This helps to limit the actual number of iterations required to remove the shell overlap, and to spread any remaining overlap over a number of shells instead of concentrating it all in a single shell. If the default value of 0.0 for the shell overlap tolerance is used, iteration continues until either all the overlap is removed or the maximum number of iterations is reached. If a nonzero value for the shell overlap tolerance is used, iteration continues until the tolerance is reached or the maximum number of iterations is reached. Note that the overlap removal process is only done once during an analysis, so a large number of iterations will only affect the first time step, not every time step.

The `SHELL OVERLAP ITERATIONS` and `SHELL OVERLAP TOLERANCE` commands have no meaning for analyses that do not have shell elements.

The code will output a warning if any initial overlap still exists after the overlap removal algorithms complete. Un-removed initial overlap may still exist if there was excessive overlap in the initial configuration that could not be removed without inverting elements. The amount of initial overlap present on the elements can be visualized by outputting the `overlap_volume_ratio` field, which is defined on all elements of blocks being skinned for contact.

8.5.5 Angle for Multiple Interactions

```
MULTIPLE INTERACTIONS = <string>ON|OFF(ON)
MULTIPLE INTERACTIONS WITH ANGLE = <real>angle_in_deg(60.0)
```

ACME Explicit Dynamics only:

When a node lies on the edge of a body, that node may need to support contact interactions with more than one face at the same time. For instance, see Figure 8.9. In Figure 8.9a, three blocks are shown, with a single node identified. Through contact, this node can interact with both the block on the upper right and the block on the bottom. If the node only supports a single interaction, then it will be arbitrarily considered for contact between one of the blocks, but not the other, as in Figure 8.9b. In this case, contact enforcement will prevent penetration into the lower block, but may permit penetration into the upper right block. The proper way to deal with this case is shown in Figure 8.9c, where multiple interactions are considered at the node.

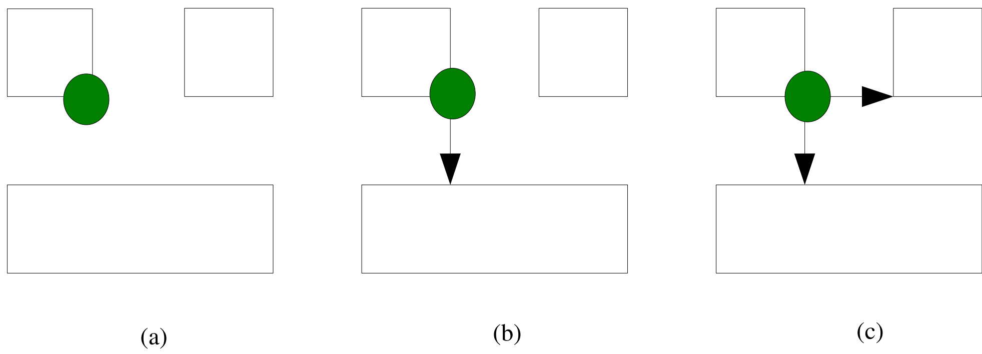


Figure 8.9: Illustrations of multiple interactions at a node: (a) initial configuration with node of interest identified (b) single interaction (c) multiple interactions

By default, Sierra/SM permits multiple interactions at a node. However, these multiple interactions may incur extra cost in the contact algorithm by increasing the number of interactions in enforcement. Also, a local search algorithm (see Section 8.5.9), which uses various contact tracking approaches, may operate more efficiently when the node can only have one interaction. Finally, multiple interactions may lead to instabilities that can be eliminated by switching to single interactions. For these reasons, the `MULTIPLE INTERACTIONS` command line allows the user to choose whether multiple interactions should be considered at a node. A value of `OFF` indicates that a node can have only one interaction. This value affects all interactions in a contact definition. Sierra/SM does not currently have the capability to force single interactions for some surface pairs while allowing multiple interactions for other surface pairs.

When the `MULTIPLE INTERACTIONS` command line is `ON`, the number of interactions that can be considered at a node is dependent on the measure of curvature of those faces that are connected to the node. If the angle between two faces on which the node is attached is small, then only one interaction is allowed. However, in cases where the angle between the faces is large enough such that they form a discrete corner, multiple interactions are considered. The contact algorithms can

properly handle only a limited number of interactions per node (currently three), so it is generally feasible to properly define interactions at a node, e.g., at the corner of a block.

The critical angle for multiple interactions is set with the `MULTIPLE INTERACTIONS WITH ANGLE` command line, where `angle` is the angle over which an edge is considered sharp. If the angle between adjoining faces is greater than this critical angle, multiple interactions can be created. By default, this critical angle is 60 degrees, which works well for most analyses. This value can be changed in the contact input if needed.

8.5.6 Surface Normal Smoothing

```
BEGIN SURFACE NORMAL SMOOTHING
  ANGLE = <real>angle_in_deg(60.0)
  DISTANCE = <real>distance(0.01)
  RESOLUTION = <string>NODE|EDGE(NODE)
END SURFACE NORMAL SMOOTHING
```

ACME Only:

Surface normal smoothing is a feature that is primarily used in implicit contact.

Finite element discretization often results in models with faceted edges, while the true geometry of the part is actually smoothly curved. If the faces of adjacent finite elements on a surface have differing normals, the discontinuities at the edges between those faces can cause problems with contact. These discontinuities in the face normals are particularly troublesome with an implicit code such as Adagio, which uses an iterative solver to obtain a converged solution at every step. If a node is in contact near an edge with a normal discontinuity, the node may slide back and forth between the two neighboring faces during the iterations. Because the normal directions of the two faces differ, this can make it difficult to converge on a solution to this discontinuous contact problem.

Surface normal smoothing is a technique that creates a smooth variation in the normal near edges. The normal varies linearly from the value on one face to the value on the other face over a distance that spans the edge. A smoothly varying normal at the edge makes it much easier for an iterative solver to obtain a converged solution in the case where a node has penetrated near the edge of a face.

Explicit dynamics does not use an iterative solver and thus does not encounter the difficulties associated with face normal discontinuities. Consequently, the `SURFACE NORMAL SMOOTHING` command block is not typically useful for explicit models, though is available in both explicit and implicit contact.

If the `SURFACE NORMAL SMOOTHING` command block is present, this feature is activated. There are three optional commands that can be used within this block to control the behavior of normal smoothing.

- The `ANGLE` command is used to control whether smoothing should occur between neighboring faces. If the angle between two faces is less than the specified angle (given in degrees), smoothing is activated between them. Otherwise, the discontinuity is considered to be a feature of the model rather than an artifact of meshing, and they are not smoothed. The default value for `angle` is 60.
- The `DISTANCE` command specifies the distance as a fraction of the face size over which smoothing should occur. The specified value can vary from 0 to 1. The default value for `distance` is 0.01.
- The `RESOLUTION` command specifies the method used to determine the smoothed normal direction. The default `NODE` option uses a node-based algorithm to fit a smooth curve, while

the `EDGE` option uses an edge-based algorithm.

8.5.7 Eroded Face Treatment

```
ERODED FACE TREATMENT = <string>ALL|NONE(ALL)
```

The `ERODED FACE TREATMENT` command line is used to define what happens to newly exposed element faces when a contact surface erodes because of element death. This command line applies to the case in which a contact surface has been generated by the skinning of an element block (Section 8.2). Suppose we have a contact block that has been skinned to create a contact surface, and let us consider an element that contributes a face to the original contact surface. If this element is killed at some point by element death, the death of this element exposes new faces. If the `ALL` option in the command line has been selected, any newly exposed faces will be included in the updated contact definition. If the `NONE` option is used, the faces exposed by element death will not be included in the updated contact surface. Both options will remove any faces on killed elements from the contact definition. The default option is `ALL`.

8.5.8 Lofted Surface Options

```
BEGIN SURFACE OPTIONS
  SURFACE = <string_list>surface_names
  REMOVE SURFACE = <string_list>removed_surface_names
  BEAM RADIUS = <real>radius
  BEAM LOFTED SHAPE = SQUARE|HEXAGON|OCTAGON (HEXAGON)
  PARTICLE RADIUS = <real>radius
  PARTICLE LOFTED SHAPE = TETRAHEDRON|CUBE|OCTAHEDRON|ICOSAHEDRON
    (ICOSAHEDRON)
  LOFTING ALGORITHM = <string>ON|OFF(ON)
  COINCIDENT SHELL TREATMENT = <string>DISALLOW|IGNORE|
    SIMPLE(DISALLOW)
  COINCIDENT SHELL HEX TREATMENT = <string>DISALLOW|
    IGNORE|TAPERED|EMBEDDED(DISALLOW)
  CONTACT SHELL THICKNESS =
    ACTUAL_THICKNESS|LET_CONTACT_CHOOSE(ACTUAL_THICKNESS)
  ALLOWABLE SHELL THICKNESS TO ELEMENT SIZE RATIOS =
    <real>lower_bound(0.1) TO <real>upper_bound(1.0)
END [SURFACE OPTIONS]
```

The SURFACE and REMOVE SURFACE command lines allow for the specification of a subset of contact surfaces that this command block refers to. These command lines are optional, by default the commands in the surface options block apply to all contact surfaces. If the SURFACE command line is used by itself, only those surfaces listed will be effected. If the REMOVE SURFACE command line is used by itself, all contact surfaces except those listed will be effected.

The BEAM RADIUS command can be used to explicitly set the radius of the lofted beam elements. by default contact represents lofted beam elements as a six sided prism. The radius of the vertexes of the hexagons on the ends of the beam is the square root of the beam cross sectional area divided by pi. I.e., the beam cross sectional area is converted into a circle, and then the hexagon end cap of the beam roughly represents this circle.

The BEAM LOFTED SHAPE command changes how the beam is represented. Objects with fewer facets tend to be less accurate, but may run faster.

The PARTICLE RADIUS command can be used to explicitly set the radius of the lofted particle elements. By default, contact represents lofted particle elements as twenty-sided sided icosahedrons having radii equal to half of the value of the sph_radius field.

The BEAM LOFTED SHAPE can change how the finely the particle sphere is represented. Objects with fewer facets tend to be less accurate, but may result in improved run times.

The CONTACT SHELL THICKNESS command line controls whether lofted shell contact should be enforced using the actual thickness of elements, or using a pseudo thickness that is computed by the contact algorithm. The default option to this command, ACTUAL_THICKNESS, causes the actual element thickness to be used.

The LET_CONTACT_CHOOSE option to the CONTACT SHELL THICKNESS command tells the contact algorithm to create lofted geometries out of the shells that are more appropriate for contact.

Shells that are very thick or very thin in relation to their in-plane dimension can be problematic in contact. Thin shells may require a large number of time sub-steps in the contact algorithm to ensure that objects do not pass completely through one another. Shells that are thick in relation to their in-plane dimension can produce ill-defined geometries as seen in Figure 8.10. The `LET_CONTACT_CHOOSE` option can help alleviate these problems. The `CONTACT SHELL THICKNESS` command is functional only in Dash contact.

By default, when the `LET_CONTACT_CHOOSE` option is used, lofted geometry thicknesses are ensured to be within 0.1 and 1.0 times the in-plane dimension of the shell. If the contact-appropriate thickness already lies within those bounds, the lofted thickness will not be changed. The permissible bounds for the ratios of contact-appropriate thickness to actual thickness can be set using the `lower_bound` and `upper_bound` parameters in the `ALLOWABLE SHELL THICKNESS TO ELEMENT SIZE RATIOS` command. The `ALLOWABLE SHELL THICKNESS` command is functional only in Dash contact.

In ACME contact the contact on shell elements can occur on either the meshed shell geometry, i.e., ignoring any shell thickness, or on the "lofted" geometry, i.e., a geometry that includes the thickness of the shell.

The `LOFTING ALGORITHM` command line determines whether contact on a shell should be done on the lofted geometry or on the original shell geometry. If the value of the `LOFTING ALGORITHM` command line is set to `ON`, shell contact uses the lofted geometry; if it set to `OFF`, shell contact uses the original shell geometry. This command only effects ACME contact.

The `COINCIDENT SHELL TREATMENT` command line identifies how shells that share the same nodes should be treated. If the `DISALLOW` option is selected, (the default), then any time that shells in contact are detected to share all the same nodes, the code will abort with an error message indicating which elements were found to be coincident. The `DISALLOW` option should be used if you do not want any coincident shells to be considered in the analysis. The option operates essentially as a check on the mesh. If the `IGNORE` option is selected, any contact faces attached to coincident shells are ignored for contact. This option is only provided as a backup approach if undiagnosed code problems arise from coincident shells. If such a case occurs, the `IGNORE` option may permit the user to continue with an analysis while the code team diagnoses the problem. The `SIMPLE` option enables coincident shells to be processed correctly. If lofting has been enabled and the `SIMPLE` option is selected, the thickness of the lofted coincident shell is taken as the largest thickness of all the coincident shells. If lofting is off and the `SIMPLE` option is selected, the coincident shell is treated as if only one of the shells is present. This command only effects ACME contact.

The `COINCIDENT SHELL HEX TREATMENT` command line has a function similar to that of the `COINCIDENT SHELL TREATMENT` command line. The `COINCIDENT SHELL HEX TREATMENT` command line, however, identifies how shells that are fully coincident with the hex elements are treated. If the `DISALLOW` option is selected (the default), then any time that a shell in contact is detected to share all the same nodes with the face of a continuum element, the code will abort with an error message indicating which elements were found to be coincident. The `DISALLOW` option should be used if you do not want any shells coincident with hexes to be considered in the analysis. The option operates essentially as a check on the mesh. If the `IGNORE` option is selected, any contact faces attached to shells that are coincident with faces of continuum elements

are ignored for contact. This option is only provided as a backup approach if undiagnosed code problems arise from coincident shells and continuum elements. If such a case occurs, the `IGNORE` option may permit the user to continue with an analysis while the code team diagnoses the problem. The `TAPERED` and `EMBEDDED` options permit shells that are coincident with faces of continuum elements to be processed in contact. The `TAPERED` option does two things: it includes for contact any faces that are on the free surface and ignores faces sandwiched between the shell and the continuum element, and it automatically adjusts the lofting of the surfaces to provide a smooth transition between shells that are not coincident with the faces of the continuum elements and those that are coincident with the faces of the continuum elements. The `EMBEDDED` option includes for contact both free surface faces and those that are between the coincident shells and faces of the continuum elements; the option does not adjust thicknesses to make smooth transitions between shells that are not coincident with faces of continuum elements and those that are coincident with faces of continuum elements. In general, the `TAPERED` option is preferred; only use the `EMBEDDED` option if the `TAPERED` option causes a code problem. This command only effects ACME contact.

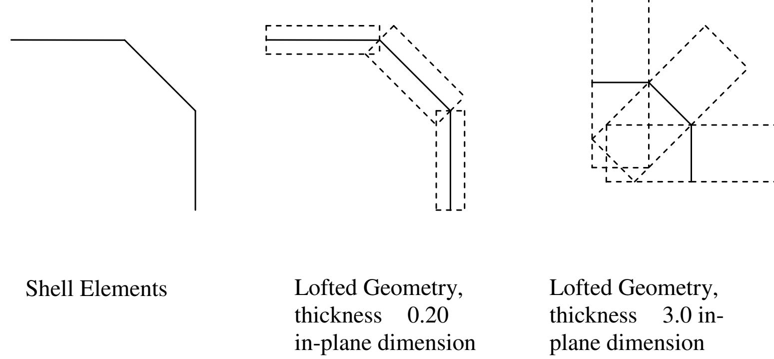


Figure 8.10: Example lofted geometries produced by shell lofting

8.5.8.1 Examples of lofted geometry

Lofted quad and triangular shells are shown in Figure 8.11.

Available lofted beam representations are shown in Figure 8.12.

Available lofted particle element and node representations are shown in Figure 8.13.

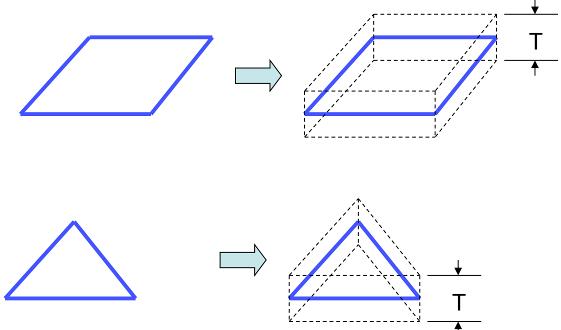


Figure 8.11: Lofted shell geometry

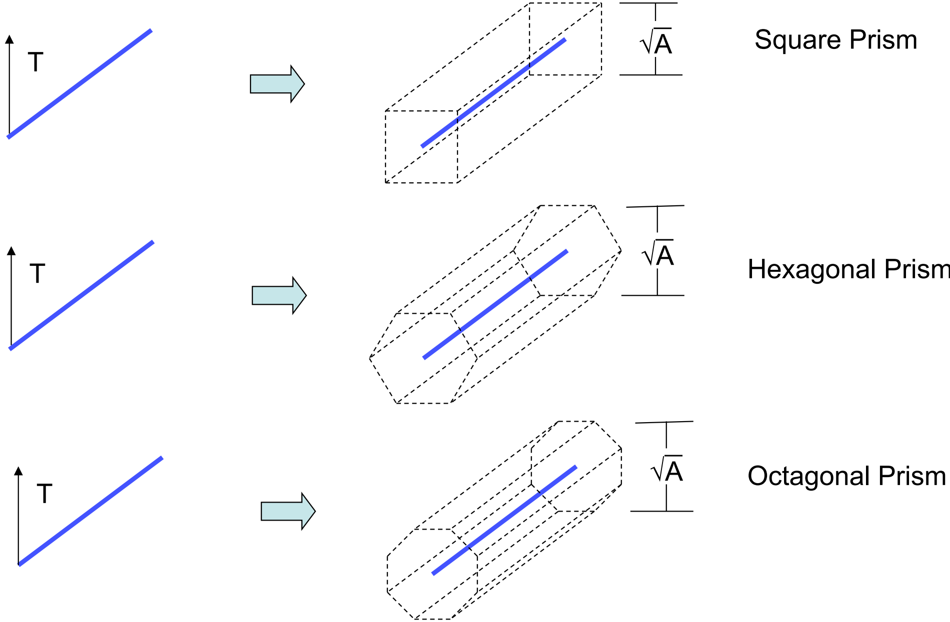


Figure 8.12: Lofted beam geometry

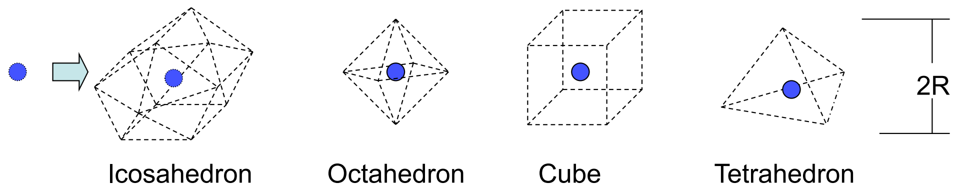


Figure 8.13: Lofted particle geometry

8.5.9 Search Options

```
BEGIN SEARCH OPTIONS [<string>name]
  #
  # Search tolerances
  SEARCH TOLERANCE = <string>AUTOMATIC|USER_DEFINED(AUTOMATIC)
  NORMAL TOLERANCE = <real>norm_tol
  TANGENTIAL TOLERANCE = <real>tang_tol
  FACE MULTIPLIER = <real>face_multiplier(0.1)
  CAPTURE TOLERANCE = <real>cap_tol          Implicit
  TENSION RELEASE = <real>ten_release        Implicit
  SLIP PENALTY = <real>slip_pen              Implicit
END [SEARCH OPTIONS <string>name]
```

Contact involves a search phase and an enforcement phase. The contact search algorithm used to detect interactions between contact surfaces is often the most computationally expensive part of an analysis. The user can exert some control over how the search phase is carried out via the `SEARCH OPTIONS` command block. By selecting different options in this command block, the user can make trade-offs between the accuracy of the search and computing time.

The most accurate approach to the search phase is a global search at every time step. For a global search, a box is drawn around each face. The box depends on the shape of the face, the location of the face in space, and search tolerances. Now suppose we want to determine whether some node has penetrated that face. We must first determine if the node lies in one or more boxes that surround a face. This search, although done with an optimal algorithm, is still time consuming. The search must be done for all nodes that may be in contact with a face. A less accurate approach for the search phase is to use what is called a local tracking algorithm. For the tracking algorithm approach, we first do a global search. When a node has contacted a face in the global search, we record the face (or faces) contacted by the node. Instead of using the global search on subsequent time steps, we simply rely on the record of the node-face interactions to compute the contact forces. The last face contacted by a node in the global search is assumed to remain in contact with that node for subsequent time steps. In actuality, the node may slide off the face it was contacting at the time of the global search. In this case, faces that share an edge with the original contact face are searched to determine whether they (the edge adjacent faces) are in contact with the node. If the node moves across a corner of the face (rather than an edge), we may lose the contact interaction for the node until the next global search. If we lose the contact interaction, we lose some of the accuracy in the contact calculations until we do the next global search. Furthermore, it is possible that additional nodes may actually come into contact in the time steps between global searches. These nodes are typically caught during the next global search, but inaccuracies can result from missing the exact time of contact. The tracking algorithm, under certain circumstances, can work quite well even though it is less accurate. We can encounter analyses where we can set the number of intervals (time steps) between global searches to a relatively small number (5) and lose only a few or none of the node-to-face contacts between global searches. Likewise, we can encounter analyses where we can set the interval between global searches to a large number (100 or more) and lose only a few or none of the node-to-face contacts between global searches. Finally, we can encounter problems where we may only have to do one global search at the beginning and rely

solely on the tracking information for the rest of the problem (without losing any contact). What search approach is best for your problem depends on the geometry of your structure, the loads on your structure, and the amount of deformation of your structure. This section tells you how to control the search phase for your specific problem.

The SEARCH OPTIONS command block begins with the input line:

```
BEGIN SEARCH OPTIONS [<string>name]
```

and ends with:

```
END [SEARCH OPTIONS <string>name]
```

The name for the command block is optional.

Without a SEARCH OPTIONS command block, the default search with associated default search parameters is used for all contact pairs. If you want to override the default search method for all contact pairs, you should add a SEARCH OPTIONS command block. By adding a SEARCH OPTIONS command block, you establish a new set of global defaults for the search for all contact pairs.

The valid command lines within a SEARCH OPTIONS command block are described in Section 8.5.9 and Section 8.5.9.1. The values specified by these commands are applied by default to all interaction contact surfaces, unless overridden by a specific interaction definition.

8.5.9.1 Search Tolerances

```
SEARCH TOLERANCE = <string>AUTOMATIC|USER_DEFINED(AUTOMATIC)
NORMAL TOLERANCE = <real>norm_tol
TANGENTIAL TOLERANCE = <real>tang_tol
FACE MULTIPLIER = <real>face_multiplier(0.1)
```

As indicated previously, the contact functionality in Sierra/SM uses a box defined around each face to locate nodes that may potentially contact the face. This box is defined by a tolerance normal to the face and another tolerance tangential to the face (see Figure 8.14). The code adds to these tolerances the maximum motion over a time step when identifying interactions. In the above command lines, the parameter norm_tol is the normal tolerance (defined on the NORMAL TOLERANCE command line) for the search box and the parameter tang_tol is the tangential tolerance (defined on the TANGENTIAL TOLERANCE command line) for the search box.

By default, Sierra/SM will automatically calculate normal and tangential tolerances based on the minimum characteristic length multiplied by the value input by the FACE MULTIPLIER command. The face multiplier is 0.1. The automatic tolerances add the maximum motion over a time step just like the user defined tolerances. If you leave automatic search on and also specify normal and/or tangential tolerances with the NORMAL TOLERANCE and TANGENTIAL TOLERANCE command lines, the larger of the two (automatic or user specified) tolerances will be used. For example,

suppose you specify a normal tolerance of 1.0×10^{-3} and the automatic tolerancing computes a normal tolerance of 1.05×10^{-3}. Then Sierra/SM will use a normal tolerance of 1.05×10^{-3}.

When the USER_DEFINED option is specified for the SEARCH TOLERANCE command line, these normal and tangential tolerances must be specified. If these tolerances are not specified, code execution will be terminated with an error.

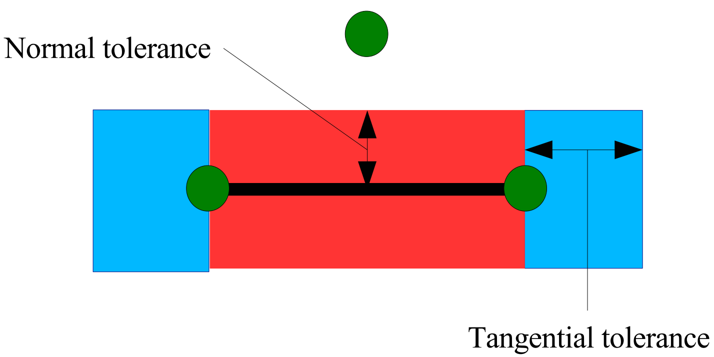


Figure 8.14: Illustration of normal and tangential tolerances

Both of these tolerances are absolute distances in the same units as the analysis. The proper tolerances are problem dependent. If a normal or tangential tolerance is specified in the SEARCH OPTIONS command block, they apply to all interactions. These default search tolerances can be overwritten for a specific interaction by specifying a value for the normal tolerance and/or tangential tolerance for that interaction inside the INTERACTION command block (see Section 8.4.2).

8.5.10 User Search Box

```
BEGIN USER SEARCH BOX <string>name
  #
  # box center point
  CENTER = <string>center_point
  X DISPLACEMENT FUNCTION = <string>x_disp_function_name
  Y DISPLACEMENT FUNCTION = <string>y_disp_function_name
  Z DISPLACEMENT FUNCTION = <string>z_disp_function_name
  X DISPLACEMENT SCALE FACTOR = <real>x_disp_scale_factor
  Y DISPLACEMENT SCALE FACTOR = <real>y_disp_scale_factor
  Z DISPLACEMENT SCALE FACTOR = <real>z_disp_scale_factor
  #
  # box lengths
  X EXTENT FUNCTION = <string>x_extent_function_name
  Y EXTENT FUNCTION = <string>y_extent_function_name
  Z EXTENT FUNCTION = <string>z_extent_function_name
  #
END [USER SEARCH BOX <string>name]
```

Note: This option is available only for Dash contact.

User-defined search boxes can be used to improve the efficiency of the contact search in some situations. The `USER SEARCH BOX` command can be used to define the coordinates of a rectangular box aligned with the global coordinate system that contains the regions where contact will be enforced. Multiple user search boxes can be defined.

Commands used within the `USER SEARCH BOX` command block define the location of the center of the box, a prescribed displacement of the center, and the size of the box in the three global coordinate directions. These are documented in the following sections.

If the `USER SEARCH BOX` command is used, contact will not be enforced outside the set of user-defined search boxes specified by the user. These search boxes should be defined over an area larger than the actual contact zone to handle cases of large motion.

In general, problems having a large area in which the analyst wishes to ignore contact (such as an area of particle collisions outside the scope of interest) or in which the area of contact is a known patch that encompasses a small subset of the faces in the contact surfaces are the most likely to experience enhanced performance with user-defined search boxes.

8.5.10.1 Search Box Location

```
CENTER = <string>center_point
X DISPLACEMENT FUNCTION = <string>x_disp_function_name
Y DISPLACEMENT FUNCTION = <string>y_disp_function_name
Z DISPLACEMENT FUNCTION = <string>z_disp_function_name
X DISPLACEMENT SCALE FACTOR = <real>x_disp_scale_factor
Y DISPLACEMENT SCALE FACTOR = <real>y_disp_scale_factor
```

```
Z DISPLACEMENT SCALE FACTOR = <real>z_disp_scale_factor
```

The location of a user search box is specified by a combination of a center point location and a set of extents in the global coordinates. The commands listed here define the location of the center of the search box as a function of time.

The CENTER command is used to specify a point, where center_point is the name of a point that is defined externally to this command block using the DEFINE POINT command (See Section 2.1.6).

The X DISPLACEMENT FUNCTION, Y DISPLACEMENT FUNCTION, and Z DISPLACEMENT FUNCTION command lines refer to functions that define the motion of the search box center in the X, Y, and Z directions as a function of time. These commands are required.

The X DISPLACEMENT SCALE FACTOR, Y DISPLACEMENT SCALE FACTOR, and Z DISPLACEMENT SCALE FACTOR command lines are optional, and can be used to provide scale factors that are multiplied by the corresponding displacement functions.

8.5.10.2 Search Box Size

```
X EXTENT FUNCTION = <string>x_extent_function_name
Y EXTENT FUNCTION = <string>y_extent_function_name
Z EXTENT FUNCTION = <string>z_extent_function_name
```

The size of the user search box is defined by the X, Y, and Z extent functions, which are required. These are specified using the X EXTENT FUNCTION, Y EXTENT FUNCTION, and Z EXTENT FUNCTION command lines. Each of these functions defines the total length of the box in the respective coordinate direction as a function in time. The search box extends half the distance in the negative and positive directions from the center point.

8.5.11 Enforcement Options

```
BEGIN ENFORCEMENT OPTIONS [<string>name]
  MOMENTUM BALANCE ITERATIONS = <integer>num_iterations(5)
  NUM GEOMETRY UPDATE ITERATIONS =<integer>num_updates(1)

END [ENFORCEMENT OPTIONS <string>name]
```

Contact, as previously indicated, involves a search phase and an enforcement phase. The user can exert some control over how the enforcement phase is carried out via the ENFORCEMENT OPTIONS command block. By selecting different options in this command block, the user can make trade-offs between solution accuracy and computing time. The ENFORCEMENT OPTIONS command block begins with the input line:

```
BEGIN ENFORCEMENT OPTIONS [<string>name]
```

and is terminated with the input line:

```
END [ENFORCEMENT OPTIONS <string>name]
```

The name for the command block, name, is optional

Only a single ENFORCEMENT OPTIONS command block is permitted within a CONTACT DEFINITION command block. Without an ENFORCEMENT OPTIONS command block, the default enforcement algorithm with associated default enforcement options is used for all contact pairs. If you want to override the defaults for enforcement for all contact pairs, you should add an ENFORCEMENT OPTIONS command block. By adding this command block, you establish a new set of global defaults for enforcement for all contact pairs. You can override some of these global defaults for enforcement for a contact pair by inserting certain command lines in the INTERACTION command block (see Section 8.4.2) for that contact pair. It is possible, therefore to tailor the enforcement approach for individual contact pairs.

Currently, the enforcement option is of limited use. Options for user control will be expanded in future versions of Sierra/SM.

The momentum balance algorithm for enforcement of the contact constraints uses an iterative process to ensure incremental momentum balance over a time step. Rather than making one pass to compute contact forces for node push-back, several passes are made to more accurately compute the normal contact force and, subsequently, the tangential (frictional) contact forces. The number of passes (iterations) is set by the value num_iterations in the MOMENTUM BALANCE ITERATIONS command line. The default value for the number of iterations is 5. This value is generally acceptable for removing overlap in the mesh. To get accurate results in a global sense in analyses that use friction, a value of 10 is more appropriate. To get accurate contact response at individual points in analyses with friction, a value of 20 or greater may be needed. Note that as

the number of iterations increases, the expense of enforcement increases. Thus a user can balance execution speed and accuracy with this command line, though care must be taken to ensure that the appropriate level of accuracy is attained. This command line affects only the enforcement phase of the contact. A single search phase is used for contact detection, but the enforcement phase uses an iterative process.

The `NUM GEOMETRY UPDATE ITERATIONS` defines the number of geometry updates contact will take. When using geometry updates contact iterations will be redefined in the new predicted geometry increasing accuracy, but also significantly increasing cost.

8.5.12 Legacy Contact

This section describes the use of the contact library written for JAS3D. This library is referred to as the Legacy Contact Library or LCLib.

Adagio has two contact search and enforcement options. The first option, the default, is to use ACME for the contact search and Adagio's own enforcement library. The second option is to use JAS3D's contact library for both the search and the enforcement. While the capabilities of the two options are nominally the same, one may perform better than the other for certain problems.

To invoke the use of LCLib, include the line command

```
JAS MODE
```

in the region.

When using LCLib, only one `CONTACT DEFINITION` block is allowed in the region. The `ENFORCEMENT` line command must invoke the `FRICTIONAL` type. If frictionless contact is desired for an interaction, set the `FRICTION COEFFICIENT` to zero. If tied contact is desired for an interaction, set the `FRICTION COEFFICIENT` to −1.

8.6 Active Periods

```
ACTIVE PERIODS = <string list>period_names
```

If the CONTACT DEFINITION command block is present in the region, it is active for all periods by default. The contact definition can be activated for specific time periods with the ACTIVE PERIODS command line. See Section 2.5 for more information about this optional command line.

8.7 Contact-Specific Outputs

Contact variables can be output to provide information about enforcement of contact interactions. Currently, information on only one interaction at each node is provided. If a node has more than one interaction, the last one in its internal interaction list is reported.

Nodal contact variables that can be output are listed in Table 9.8. Where applicable, names of the equivalent variables in JAS3D are given in parentheses at the end of the description. The variables can be output in history files or results files; see Chapter 9 for more information on outputting nodal variables. Note that currently the variables cannot be calculated at output time so the first time they are output a request is made to calculate them. This means that the first output step where they are to appear the data will be all zero. A work around for this is to have at least one output step in which these variables appear before their values are needed.

If Dash contact is used, the contact output variables are not computed by default. To have those variables available for output with Dash contact, it is necessary to include the command `COMPUTE CONTACT VARIABLES = ON` in the contact definition command block.

8.7.1 Contact Debugging

```
BEGIN DEBUG
  VISUALIZE CONTACT FACETS = OFF|ON(OFF)
  OUTPUT CONTACT SURFACES AND CONSTRAINTS = OFF|ON(OFF)
  OUTPUT LEVEL = <integer>level(0)
END
```

Note: This option is available only for Dash contact.

The debug contact block enables additional information to be output by the code to help debug what may be going wrong with a contact problem.

The `OUTPUT LEVEL` command controls the amount of information written to the log file. At the default level of zero, only minimal information is included. At level 1 or greater, more information is printed to the log file such as the composition of the different contact surfaces and which friction model is being used for which surface pairing.

The `VISUALIZE CONTACT FACETS` command in the `DEBUG` command block can be used to plot the actual contact facets and lofted contact geometries that contact is being evaluated on. When this command is set to `ON` extra element blocks and node blocks will be added to the output mesh to represent the contact geometries. Note that use of this option will significantly increase the size of the results file and moderately increase the run time. he naming convention used for these surfaces is to prepend the prefix `CSURF_` to the original contact surface name. For example, if a contact surface is produced by skinning `block_1` then the a side set will be created with the name `CSURF_block_1`. This is a Dash only option.

The `OUTPUT CONTACT SURFACES AND CONSTRAINTS` command block can be used to plot interactions into the mesh output file as sets of surfaces and nodes. Viewing the interactions can help debug contact problems. Additionally these interactions can be imported into another code such as Salinas.

8.8 Examples

This section has several example problems. We present the geometric configuration for the problems and the appropriate command lines to describe contact for the problems.

8.8.0.1 Example 1

Our first example problem has two blocks that come into contact due to initial velocity conditions. Block 1 has an initial velocity equal to $v1$, and block 2 has an initial velocity equal to $v2$. The geometric configuration for this problem is shown in Figure 8.15.

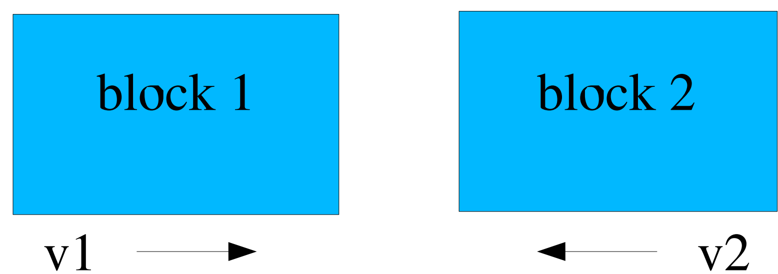


Figure 8.15: Problem with two blocks coming into contact

The simplest input for this problem will be named `EXAMPLE1` and is shown as follows:

```
BEGIN CONTACT DEFINITION EXAMPLE1
  #
  # enforcement option
  ENFORCEMENT = KINEMATIC
  #
  # define contact surfaces
  SKIN ALL BLOCKS = ON

  # contact surfaces
  CONTACT SURFACE B1 CONTAINS BLOCK_1
  CONTACT SURFACE B2 CONTAINS BLOCK_2

  # set interactions
  BEGIN INTERACTION DEFAULTS
    GENERAL CONTACT = ON
  END INTERACTION DEFAULTS
  # set interactions
  BEGIN INTERACTION EX1
    MASTER = B1
    SLAVE = B2
    CAPTURE TOLERANCE = 1.0E-3
```

```
    NORMAL TOLERANCE = 1.0E-3
    TANGENTIAL TOLERANCE = 1.0E-3
  END INTERACTION EX1
END
```

In our example, the `SKIN ALL BLOCKS` command line with its parameter set to `ON` will create a surface named `surface_1` (from the skinning of `block_1` and a surface named `surface_2` (from the skinning of `block_2`).

All the normal and tangential tolerances will be set automatically in the above example. Frictionless contact is assumed. The kinematic partition factor defaults to 0.5 for both surfaces, `surface_1` and `surface_2`.

If you omitted the `INTERACTION DEFAULTS` command block with `GENERAL CONTACT` set to `ON`, then contact enforcement would not take place.

In this example, we define kinematic contact with the default being to enforce frictionless contact.

Now, let us consider the same problem (two blocks coming into contact) in which the contact definition for the problem is not defined simply by using all the default settings. With frictional contact. The input for this variation of our two-block problem will be named `EXAMPLE1A` and is shown as follows:

```
BEGIN CONTACT DEFINITION EXAMPLE1A
  #
  # enforcement option
  ENFORCEMENT = KINEMATIC
  #
  # define contact surfaces
  SKIN ALL BLOCKS = ON

  # friction model definition
  BEGIN CONSTANT FRICTION MODEL ROUGH
    FRICTION COEFFICIENT = 0.5
  END CONSTANT FRICTION MODEL ROUGH
  # contact surfaces
  CONTACT SURFACE B1 CONTAINS BLOCK_1
  CONTACT SURFACE B2 CONTAINS BLOCK_2
  # search options
  BEGIN SEARCH OPTIONS
    NORMAL TOLERANCE = 1.0E-3
    TANGENTIAL TOLERANCE = 1.0E-3
  END SEARCH OPTIONS

  # set interactions
  BEGIN INTERACTION EX1A
    MASTER = B1
    SLAVE = B2
    FRICTION MODEL = ROUGH
```

```
        CAPTURE TOLERANCE = 1.0E-3
        NORMAL TOLERANCE = 1.0E-3
        TANGENTIAL TOLERANCE = 1.0E-3
      END INTERACTION EX1A
    END
```

As is the case of the EXAMPLE1 command block, the SKIN ALL BLOCKS command line with its parameter set to ON will create a surface named surface_1 (from the skinning of block_1) and a surface named surface_2 (from the skinning of block_2).

For EXAMPLE1A, we want to have frictional contact between the two blocks. For the frictional contact, we define a constant friction model with a CONSTANT FRICTION MODEL command block. We name this model ROUGH. specify the coefficient of friction as 0.5.

The SEARCH OPTIONS we have set values for the normal and tangential tolerances. The option to compute the search tolerance automatically has been left on. The larger of the two values—an automatically computed tolerance or the user-specified tolerance—will be selected as the search tolerance during the search phase.

In the INTERACTION DEFAULTS command block, we select the friction model ROUGH on the FRICTION MODEL command line. As in the case of the EXAMPLE1 command block, if you omitted the INTERACTION DEFAULTS command block with GENERAL CONTACT set to ON, contact enforcement would not take place.

8.8.0.2 Example 2

Our second example problem has three blocks that come into contact due to initial velocity conditions. Block 1 has an initial velocity equal to $v1$, and block 3 has an initial velocity equal to $v3$. The geometric configuration for this problem is shown in Figure 8.16.

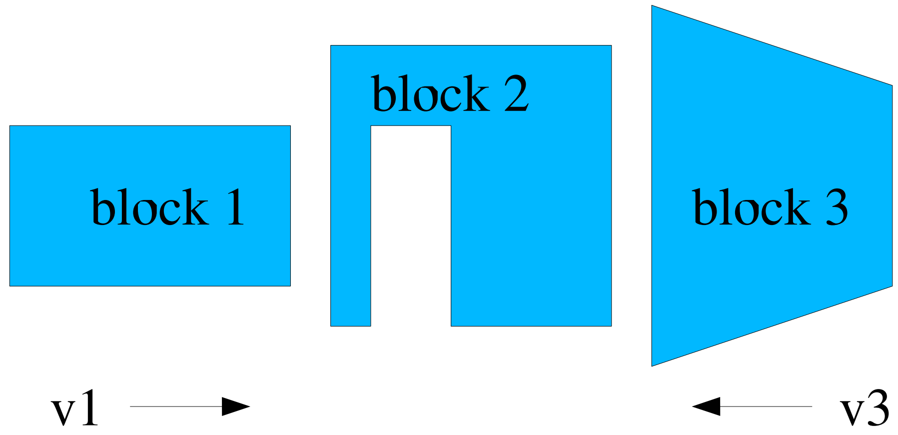


Figure 8.16: Problem with three blocks coming into contact

The input for this three-block problem will be named EXAMPLE2 and is shown as follows:

```
BEGIN CONTACT DEFINITION EXAMPLE2
  #
  # enforcement
  ENFORCEMENT = KINEMATIC
  # define contact surfaces
  CONTACT SURFACE surface_1 CONTAINS block_1
  CONTACT SURFACE surface_2 CONTAINS block_2
  CONTACT SURFACE surf_3 CONTAINS surface_3

  BEGIN CONSTANT FRICTION MODEL ROUGH
    FRICTION COEFFICIENT = 0.5
  END CONSTANT FRICTION MODEL ROUGH
  #
  # search options
  BEGIN SEARCH OPTIONS
    GLOBAL SEARCH INCREMENT = 10
    NORMAL TOLERANCE = 1.0E-3
    TANGENTIAL TOLERANCE = 1.0E-3
  END SEARCH OPTIONS
  #
  # set interactions
  BEGIN INTERACTION DEFAULTS
    FRICTION MODEL = ROUGH
    GENERAL CONTACT = ON
    SELF CONTACT = ON
  END INTERACTION DEFAULTS

  # set specific interaction
  BEGIN INTERACTION S2TOS3
    SURFACES = surface_2 surf_3
    KINEMATIC PARTITION = 0.4
    NORMAL TOLERANCE = 0.5E-3
    TANGENTIAL TOLERANCE = 0.5E-3
    FRICTION MODEL = FRICTIONLESS
  END INTERACTION S2TOS3

  # set interaction
  BEGIN INTERACTION S1TOS2
    MASTER = surface_2
    SLAVE = surface_1
    NORMAL TOLERANCE = 1.0E-3
    TANGENTIAL TOLERANCE = 1.0E-3
  END INTERACTION S2TOS3

  # set interaction
  BEGIN INTERACTION S2TOS3
    MASTER = surface_2
```

```
      SLAVE = surf_3
      NORMAL TOLERANCE = 0.5E-3
      TANGENTIAL TOLERANCE = 0.5E-3
    END INTERACTION S2TOS3
  END
```

For the EXAMPLE2 command block, we have defined three surfaces. The first surface, surface_1, is obtained by skinning block_1. The second surface, surface_2 is obtained by skinning block_2. The third surface, surf_3, is the user-defined surface surface_3. The user-defined surface, surface_3, can contain a subset of the external element faces that define block_3 or all the external element faces that define block_3.

The SEARCH OPTIONS command block sets the interval between global searches to 10; the default value is 5. Also, in this command block, we have set values for the normal and tangential tolerances. The option to compute the search tolerance automatically has been left on. The larger of the two values—an automatically computed tolerance or the user-specified tolerance—will be selected as the search tolerance during the search phase.

In the INTERACTION DEFAULTS command block, we select the friction model ROUGH on the FRICTION MODEL command line. Both GENERAL CONTACT and SELF CONTACT are set to ON in the EXAMPLE2 command block. For this problem, block_2 can undergo self-contact. Setting GENERAL CONTACT to ON will enforce contact between surface_1 and surface_2, surface_2 and surf_3, and surface_1 and surf_3. Setting SELF CONTACT to ON will enforce self-contact for all three of the surfaces.

For this particular example, we want to override some of the Sierra/SM default values for surface interaction and some of the default values for surface interaction set by the INTERACTION DEFAULTS command block for the interaction between surface_2 and surf_3. To override default values, we use an INTERACTION command block and indicate that it applies to surface_2 and surf_3 with a SURFACES command line. We override the Sierra/SM default for the kinematic partition factor by using a KINEMATIC PARTITION command line with the kinematic partition parameter set to a value of 0.4. We override the normal and tangential tolerances and the friction model set in the INTERACTION command block. The normal and tangential tolerances for interaction between surface_2 and surf_3 is set to 0.5E-3 rather than the global value of 1.0E-3. The friction model for interaction between surface_2 and surf_3 is set to FRICTIONLESS rather than the default value of ROUGH.

8.9 Common Contact Algorithmic Issues

This sections describes some of the common problem issues that may be encountered when setting up contact.

Contact constraints are one type of constraint among many possible types of constraints. Other constraint types include MPCs 7.12.7, kinematic constraints 7, and welds 7.12. Contact constraints can potentially conflict with these other constraint types causing unexpected behavior. Refer to Appendix E for information on how conflicting constraints are handled.

Tied contact that occurs over a substantial distance is not guaranteed to preserve energy, or angular momentum at the interface. The issue becomes more problematic as the distance between tied objects becomes a substantial fraction of the face size of the contact faces. It is recommended that only objects that are next to each other be tied.

Lofted objects that are large compared to the element size. For example lofting of shells that are thicker than they are wide or beams that with a large radius compared to the element length.

The speed of contact is based primarily on the number of nodes and faces in the contact surfaces and, to a much lesser extent, on the number of interactions specified. Consequently, choosing the third approach above is not likely to reduce the run time significantly. Choosing excessively large search tolerances will greatly increase the run time required by contact.

8.10 References

1. Brown, K. H., R. M. Summers, M. W. Glass, A. S. Gullerud, M. W. Heinstein, and R. E. Jones. *ACME: Algorithms for Contact in a Multiphysics Environment*, API Version, 2.2, SAND2004-5486. Albuquerque, NM: Sandia National Laboratories, October 2001. pdf.

2. Heinstein, M. W., and T. E. Voth. *Contact Enforcement for Explicit Transient Dynamics*, Draft SAND report. Albuquerque, NM: Sandia National Laboratories, 2005.

Chapter 9

Output

Sierra/SM produces a variety of output. This chapter discusses how to control the four major types of output: results output, history output, heartbeat output, and restart output. Results output lets the user select a set of variables (internal, user-defined, or some combination thereof). If the user selects a nodal variable such as displacement for results output, the displacements for all the nodes in a model will be output to a results file. If the user selects an element variable such as stress for results output, the stress for all elements in the model that calculate this quantity (stress) will be output. The history output option lets the user select a very specific set of information for output. For example, if you know that the displacement at a particular node is critical, then you can select only the displacement at that particular node as history output. The heartbeat output is similar to the history output except that the output is written to a text file instead of to a binary (exodusII [1]) file. The restart output is written so that any calculation can be halted at some arbitrary analysis time and then restarted at this time. The user has no control over what is written to the restart file. When a restart file is written, it must be a complete state description of the calculations at some given time. A restart file contains a great deal of information and is typically much larger than a results file. You need to carefully limit how often a restart file is written.

Section 9.1 describes the syntax for requesting output variables. Section 9.2 describes the results output. Included in the results output is a description of commands for user-defined output (Section 9.2.2). User-defined output lets the user postprocess analysis results as the code is running to produce a reduced set of output information. Section 9.3 describes the history output, Section 9.4 describes the heartbeat output, and Section 9.5 describes the restart output. All four types of output (results, history, heartbeat, and restart) can be synchronized for analyses with multiple regions. This scheduling functionality is discussed in Section 9.6. Section 9.7 describes the commands for performing output variable interpolation. Section 9.8 defines the output options for global variables, which are defined at the region scope. Finally, in Section 9.9, there is a list of key variables.

Unless otherwise noted, the command blocks and command lines discussed in Chapter 9 appear in the region scope.

9.1 Syntax for Requesting Variables

Variables may be accessed in the code either in whole or by component. Values at specific components or integration points of multi-component variables may be accessed via parenthesis syntax. Parenthesis syntax may be applied to results output, history output, element death, or any other command where variable names are specified. Values of single-component variables indexed in some other way may be accessed with underscore syntax, which is primarily applicable to rigid body fields as discussed in Section 9.1.4.

Parenthesis syntax is a variable name of the form:

```
<string>var_name[(<index>component[,<integer>integration_point)]]
```

For a variable named `var`, a variable name of the form `var(A,B)` asks for the `A` component of the variable at integration point `B`. If a variable is a vector, `x`, `y`, or `z` may be specified as the component. If a variable is 3x3 tensor, `xx`, `yy`, `zz`, `xy`, `xz`, `yz`, `yx`, `zx`, or `zy` may be specified as the component. For other types of variables components of the variable may be requested through an integer index.

The characters `:` and `*` are wild cards if used for specifying either the component or the integration point. `var(:,B)` asks for all components of var at integration point B. `var(A,:)` asks for component A of var at all integration points. `var(:,:)` asks for all components of var at all integration points. `var` is shorthand for `var(:,:)`.

`var(A)` will behave slightly differently depending on the nature of the variable. If the variable has multiple components, then `Var(A)` is treated like `var(A,:)`. If a variable has one and only one component then it is assumed that A refers to the integration point number rather than the component number and `Var(A)` is treated like `var(1,A)`.

9.1.1 Example 1

Let stress be a tensor defined on a single integration point element and a displacement vector be defined at all model nodes. The following output variable specification:

```
element stress as str
nodal displacement as disp
```

asks for all the components of the stress tensor on elements and all components of the displacement vector on nodes. The code would write the following variables to the output file:

```
str_xx
str_yy
str_zz
str_xy
str_xz
```

```
str_yz
disp_x
disp_y
disp_z
```

If only the yy component of stress is desired, either of the following could be used:

```
element stress(yy) as my_yy_str1
element stress(2) as my_yy_str2
```

If only the z component of displacement is desired either of the following could be specified:

```
nodal displacement(z) as my_z_disp1
nodal displacement(3) as my_z_disp2
```

Note, index 2 of a tensor corresponds to the yy component of the tensor and index 3 of a vector corresponds to the z component of the vector.

9.1.2 Example 2

Let stress be a tensor defined on each integration point of a three integration point element. Let eqps be a scalar material state variable also defined at each element integration point. To ask for all stress components on all integration points and all eqps data at all integration points the following could be specified:

```
element stress as str
element eqps as eqps
```

Which would output the variables:

```
str_xx_1, str_yy_1, str_zz_1, str_xy_1, str_xz_1, str_yz_1
str_xx_2, str_yy_2, str_zz_2, str_xy_2, str_xz_2, str_yz_2
str_xx_3, str_yy_3, str_zz_3, str_xy_3, str_xz_3, str_yz_3
eqps_1, eqps_2, eqps_2
```

To ask for just the stress tensor and value of eqps on the second integration point, the following syntax can be used:

```
element stress(:,2) as str_intg2
element eqps(2) as eqps_intg2
```

This would output the variables:

```
str_intg2_xx
str_intg2_yy
str_intg2_zz
str_intg2_xy
str_intg2_xz
str_intg2_yz
eqps_intg2
```

To ask for the `xy` component of stress on all integration points any of the following could be used:

```
element  stress(xy,*) as str_xy_all
element  stress(xy,:) as str_xy_all
element  stress(xy)   as str_xy_all
```

Any of the above would output:

```
str_xy_all_1
str_xy_all_2
str_xy_all_3
```

9.1.3 Other command blocks

The parenthesis syntax described above for results output can also be used in most other commands involving variable names. For example, to kill elements based on yy stress or z displacement the following could be specified:

```
begin element death
  criterion is element value of stress(yy) > 1000
  criterion is average nodal value of displacement(z) > 3.0
end
```

9.1.4 Rigid Body Variables

Variables of rigid bodies are provided as separate components of the rigid body fields. To access variables of rigid bodies you must output the desired field component(s), and to access the field component of individual rigid bodies an underscore syntax is employed (the parenthesis syntax applies to multi-component fields and integration points). The underscore syntax simply takes the desired field component and appends it with an underscore and the desired rigid body name.

For example the following lines in a history output block (see Section 9.3)

```
global ax_rb1
global displz_rb4
```

would output to the history file the variables

```
ax_rb1
displz_rb4
```

which are the translational acceleration in the x-direction of rigid body rb1 and the translational displacement in the z-direction of rigid body rb4, respectively (see Table 9.2).

9.2 Results Output

The results output capability lets you select some set of variables that will be written to a file at various intervals. As previously indicated, all the values for each selected variable will be written to the results file. (The interval at which information is written can be changed throughout the analysis time.) The name of the results file is set in the RESULTS OUTPUT command block.

9.2.1 Exodus Results Output File

```
BEGIN RESULTS OUTPUT <string>results_name
  DATABASE NAME = <string>results_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
    (ON|TRUE|YES)
  TITLE <string>user_title
  NODE VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name] ...
    <string>variable_name [AS
    <string>dbase_variable_name]
    | NODAL VARIABLES = <string>variable_name
        [AS <string>dbase_variable_name] ...
        <string>variable_name [AS
        <string>dbase_variable_name]
  NODESET VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name] ...
    <string>variable_name [AS
    <string>dbase_variable_name]
    | NODESET VARIABLES = <string>variable_name
        [AS <string>dbase_variable_name]
        INCLUDE|ON|EXCLUDE <string list>nodelist_names
        ... <string>variable_name
        [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
        <string list>nodelist_names
  FACE VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name] ...
    <string>variable_name [AS
    <string>dbase_variable_name]
    | FACE VARIABLES = <string>variable_name
        [AS <string>dbase_variable_name]
        INCLUDE|ON|EXCLUDE <string list>surface_names
        ...  <string>variable_name
        [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
        <string list>surface_names
  ELEMENT VARIABLES = <string>variable_name
    [AS <string>dbase_variable_name] ...
    <string>variable_name [AS
    <string>dbase_variable_name]
    | ELEMENT VARIABLES = <string>variable_name
        [AS <string>dbase_variable_name]
        INCLUDE|ON|EXCLUDE <string list>block_names
        ... <string>variable_name
        [AS dbase_variable_name] INCLUDE|ON|EXCLUDE
        <string list>block_names
  GLOBAL VARIABLES =  <string>variable_name
```

```
        [AS <string>dbase_variable_name] ...
        <string>variable_name [AS
        <string>dbase_variable_name]
      OUTPUT MESH = EXPOSED_SURFACE|BLOCK_SURFACE
      COMPONENT SEPARATOR CHARACTER = <string>character|NONE
      INCLUDE = <string>list_of_included_entities
      EXCLUDE = <string>list_of_excluded_entities
      START TIME = <real>output_start_time
      TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
      AT TIME <real>time_in INCREMENT =
        <real>time_increment_dt
      ADDITIONAL TIMES = <real>output_time1
        <real>output_time2 ...
      AT STEP <integer>step_begin INCREMENT =
        <integer>step_increment
      ADDITIONAL STEPS = <integer>output_step1
        <integer>output_step2 ...
      TERMINATION TIME = <real>termination_time_value
      SYNCHRONIZE_OUTPUT
      USE OUTPUT SCHEDULER <string>scheduler name
      OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
        SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
        SIGKILL|SIGILL|SIGSEGV
    END [RESULTS OUTPUT <string>results_name]
```

Explicit (margin note for TIMESTEP ADJUSTMENT INTERVAL)

You can specify a results file, the results to be included in this file, and the frequency at which results are written by using a RESULTS OUTPUT command block. The command block appears inside the region scope.

More than one results file can be specified for an analysis. Thus for each results file, there will be one RESULTS OUTPUT command block. The command block begins with:

```
BEGIN RESULTS OUTPUT <string>results_name
```

and is terminated with:

```
END [RESULTS OUTPUT <string>results_name]
```

where results_name is a user-selected name for the command block. Nested within the RESULTS OUTPUT command block is a set of command lines, as shown in the block summary given above. The first two command lines listed (DATABASE NAME and DATABASE TYPE) give pertinent information about the results file. The command line

```
DATABASE NAME = <string>results_file_name
```

gives the name of the results file with the string results_file_name. If the results file is to appear in the current directory and is named job.e, this command line would appear as:

```
DATABASE NAME = job.e
```

If the results file is to be created in some other directory the command line must include the path to that directory.

Two metacharacters can appear in the name of the results file. If the %P character is found in the name, it will be replaced with the number processors being used for the run. For example, if you are running on 1024 processors and use the name results-%P/job.e, then the name would be expanded to results-1024/job.e and the actual results files would be results-1024/job.e.1024.0000 to results-1024/job.g.1024.1023. The other recognized metacharacter is %B which is replaced with the base name of the input file containing the input commands. For example, if the commands are in the file my_analysis_run.i and the results database name is specified as %B.e, then the results would be written to the file my_analysis_run.e.

If the results file does not use the Exodus II format [1], you must specify the format for the results file using the command line:

```
DATABASE TYPE = <string>database_type(exodusII)
```

Currently, both the Exodus II database and the XDMF database [2] are supported in Presto and Adagio. Exodus II is more commonly used than XDMF. Other options may be added in the future.

The OVERWRITE command line can be used to prevent the overwriting of existing results files.

```
OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
      (ON|TRUE|YES)
```

The OVERWRITE command line allows only a single value. If you set the value to FALSE, NO, or OFF, the code will terminate before existing results files can be overwritten. If you set the value to TRUE, YES, or ON, then existing results files can be overwritten (the default status). Suppose, for example, that we have an existing results file named job21.e. Suppose also that we have an input file with a RESULTS OUTPUT command block that contains the OVERWRITE command line set to ON and the DATABASE NAME command line set to:

```
DATABASE NAME = job21.e
```

If you run the code under these conditions, the existing results file job21.e will be overwritten.

Whether or not results files are overwritten is also impacted by the use of the automatic read and write option for restart files described in Section 9.5.1.1. If you use the automatic read and write option for restart files, the results files, like the restart files, are automatically managed. The automatic read and write option in restart adds extensions to file names and prevents the overwriting of any existing restart or results files. For the case of a user-controlled read and write of restart files (Section 9.5.1.2) or of no restart, however, the OVERWRITE command line is useful for preventing the overwriting of results files.

You may add a title to the results file by using the TITLE command line. Whatever you specify for the user_title will be written to the results file. Some of the programs that process the results file (such as various SEACAS programs [3]) can read and display this information.

The other command lines that appear in the RESULTS OUTPUT command block determine the type and frequency of information that is output. Descriptions of these command lines follow in Section 9.2.1.1 through Section 9.2.1.18.

9.2.1.1 Output Nodal Variables

```
NODE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | NODAL VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name] ...
      <string>variable_name [AS
      <string>dbase_variable_name]
```

Any nodal variable in Sierra/SM can be selected for output in the results file by using a command line in one of the two forms shown above. The only difference between the two forms is the use of `NODE VARIABLES` or `NODAL VARIABLES`. The string `variable_name` is the name of the nodal variable to output. The string `variable_name` can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

For the above two command lines, any nodal variable requested for output is output for all nodes.

It is possible to specify an alias for any of the nodal variables by using the `AS` specification. Suppose, for example, you wanted to output the external forces in Sierra/SM, which are defined as `force_external`, with the alias `f_ext`. You would then enter the command line:

```
NODE VARIABLES = force_external AS f_ext
```

In this example, the external force is a vector quantity. For a vector quantity at a node, suffixes are appended to the variable name (or alias name) to denote each vector component. The results database would have three variable names associated with the external force: `f_ext_x`, `f_ext_y`, and `f_ext_z`. You can change the component separator, an underscore in this example, by using the `COMPONENT SEPARATOR CHARACTER` command line (see Section 9.2.1.7).

The `NODE VARIABLES` command line can be used an arbitrary number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one nodal variable for output on a command line, as indicated by the ellipsis in the command line format. In the following example, two nodal variables are specified for output. Note that the internal forces are defined as `force_internal`.

```
NODE VARIABLES = force_external force_internal
```

Aliases can be specified for each of the variables in a single command line. Thus, If you wanted to output the alias `f_ext` for external forces and also wanted to output the alias `f_int` for internal forces, you would enter the command line:

```
NODE VARIABLES = force_external AS f_ext
    force_internal AS f_int
```

The specification of an alias is optional.

9.2.1.2 Output Node Set Variables

```
NODESET VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | NODESET VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>nodelist_names
      ... <string>variable_name
      [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
      <string list>nodelist_names
```

A nodal variable may be defined on a subset of the total set of nodes defining a model. A nodal variable that is defined only on a subset of nodes is referred to as a node set variable. The NODESET VARIABLES command line lets you specify a node set variable for output to the results file.

There are two forms of the NODESET VARIABLES command line. Either form will let you output a node set variable.

The first form of the command line is as follows:

```
NODESET VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS <string>dbase_variable_name]
```

Here, the string variable_name is a node set variable associated with one or more node sets. In this form, the node set variable is output for all node sets associated with that node set variable.

It is possible to specify an alias in the results file for any of the node set variables by using the AS option. Suppose, for example, you wanted to output the node set variable force_nsetype, but have that variable have the name fnsetype in the results file. You would then enter the command line:

```
NODESET VARIABLES = force_nsetype AS fnsetype
```

The NODESET VARIABLES command line can be used an arbitrary number of times within a RESULTS OUTPUT command block. It is also possible to specify more than one node set variable for output on a command line, as indicated by the ellipsis in the command line format. In the following example, two node set variables are specified for output. Here, the second node set variable is defined as force_nsetype2.

```
NODESET VARIABLES = force_nsetype force_nsetype2
```

Aliases can be specified for each of the variables in a single command line. Thus, if you wanted to output the alias fnsetype for node set variable force_nsetype and also wanted to output the alias fnsetype2 for node set variable force_nsetype2, you would enter the command line:

```
NODESET VARIABLES = force_nsetype AS fnsetype
    force_nsetype2 AS fnsetype2
```

The specification of an alias is optional.

The second form of the command line is as follows:

```
NODESET VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name]
  INCLUDE|ON|EXCLUDE <string list>nodelist_names
  ... <string>variable_name [AS <string>dbase_variable_name]
  INCLUDE|ON|EXCLUDE <string list>nodelist_names
```

This form of the NODESET VARIABLES command line is similar to the first, except that the user can control which node sets are used for output. The user can include a specific list of node sets for output by using the INCLUDE option or the ON option. (The keyword INCLUDE is synonymous with the keyword ON.) Alternatively, the user can exclude a specific list of node sets for output by using the EXCLUDE option.

Suppose that the node set variable force_nsetype from the above example has been defined for nodelist_10, nodelist_11, nodelist_20, and nodelist_21. If we only want to output the node set variable for node sets nodelist_10, nodelist_11, and nodelist_21, then we could specify the NODESET command line as follows:

```
NODESET VARIABLES = force_nsetype AS fnsetype
     INCLUDE nodelist_10, nodelist_11, nodelist_21
```

(In the above command line, the keyword ON could be substituted for INCLUDE.) Alternatively, we could use the command line:

```
NODESET VARIABLES = force_nsetype AS fnsetype
     EXCLUDE nodelist_20
```

In the above command lines, an alias for a node set can be substituted for a node set identifier. For example, if center_case is an alias for nodelist_10, then the string center_case could be substituted for nodelist_10 in the above command lines. Because a node set identifier is a mesh entity, the alias for the node set identifier would be defined via an ALIAS command line in a FINITE ELEMENT MODEL command block.

Note that the list of identifiers uses a comma to separate one node set identifier from the next node set identifier.

9.2.1.3 Output Face Variables

```
FACE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | FACE VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>surface_names
      ...  <string>variable_name
```

```
[AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
<string list>surface_names
```

A variable may be defined on some set of faces that constitute a surface. A variable defined on a set of faces is referred to as a face variable. The `FACE VARIABLES` command line lets you specify a face variable for output to the results file.

There are two forms of the `FACE VARIABLES` command line. Either form will let you output a face variable.

The first form of the command line is as follows:

```
FACE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
```

Here, the string `variable_name` is a face variable associated with one or more surfaces. In this form, the face variable is output for all surfaces associated with that face variable.

It is possible to specify an alias in the results file for any face variable by using the `AS` option. Suppose, for example, you wanted to output a face variable `pressure_face`, but have that variable have the name `pressuref` in the results file. You would then enter the command line:

```
FACE VARIABLES = pressure_face AS pressuref
```

The `FACE VARIABLES` command line can be used an arbitrary number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one face variable for output on a command line, as indicated by the ellipsis in the command line format. In the following example, two face variables are specified for output. Here, the second face variable is defined as `scalar_face2`.

```
FACE VARIABLES = pressure_face scalar_face2
```

Aliases can be specified for each of the variables in a single command line. Thus, If you wanted to output the alias `pressuref` for face variable `pressure_face` and also wanted to output the alias `scalarf2` for face variable `scalar_face2`, you would enter the command line:

```
FACE VARIABLES = pressure_face AS pressuref
    scalar_face2 AS scalarf2
```

The specification of an alias is optional.

The second form of the command line is as follows:

```
FACE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name]
  INCLUDE|ON|EXCLUDE <string list>surface_names
  ...  <string>variable_name
  [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
  <string list>surface_names
```

This form of the FACE VARIABLES command line is similar to the first, except that the user can control which surfaces are used for output. The user can include a specific list of surfaces for output by using the INCLUDE option or the ON option. (The keyword INCLUDE is synonymous with the keyword ON.) Alternatively, the user can exclude a specific list of surfaces for output by using the EXCLUDE option.

Suppose that the face variable pressure_face from the above example has been defined for surface_10, surface_11, surface_20, and surface_21. If we only want to output the face variable for surface_10, surface_11, and surface_21, then we could specify the FACE VARIABLES command line as follows:

```
FACE VARIABLES = pressure_face AS pressuref
     INCLUDE surface_10, surface_11,
     surface_21
```

(In the above command line, the keyword ON could be substituted for INCLUDE.) Alternatively, we could use the command line:

```
FACE VARIABLES = pressure_face AS pressuref
     EXCLUDE surface_20
```

In the above command lines, an alias for a surface can be substituted for a surface identifier. For example, if center_case is an alias for surface_10, then the string center_case could be substituted for surface_10 in the above command lines. Because a surface identifier is a mesh entity, the alias for the surface identifier would be defined via an ALIAS command line in a FINITE ELEMENT MODEL command block.

Note that the list of identifiers uses a comma to separate one surface identifier from the next surface identifier.

9.2.1.4 Output Element Variables

```
ELEMENT VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | ELEMENT VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>block_names
      ... <string>variable_name
      [AS dbase_variable_name] INCLUDE|ON|EXCLUDE
      <string list>block_names
```

Any element variable in Sierra/SM can be selected for output in the results file by using the ELEMENT VARIABLES command line.

There are two forms of the ELEMENT VARIABLES command line. Either form will let you output an element variable.

The first form of the command line is as follows:

```
ELEMENT VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
```

Here, the string `variable_name` is the name of the element variable to output. The string `variable_name` can be a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

In the first form of the `ELEMENT VARIABLES` command line, the element variable is output for all element blocks that have the element variable as a defined variable. For example, all the solid elements have `stress` as a defined variable. If you had a mesh consisting of hexahedral and tetrahedral elements and requested output of the element variable `stress`, then `stress` would be output for all element blocks consisting of hexahedral and tetrahedral elements.

It is possible to specify an alias for any of the element variables by using the `AS` specification. Suppose, for example, you wanted to output the stress in Sierra/SM, which is defined as `stress`, with the alias `str`. You would then enter the command line:

```
ELEMENT VARIABLES = stress AS str
```

In this example, stress is a symmetric tensor quantity. For a symmetric tensor quantity, suffixes are appended to the variable name (or alias name) to denote each symmetric tensor component. The results database would have six variable names associated with the stress: `stress_xx`, `stress_yy`, `stress_zz`, `stress_xy`, `stress_yz`, and `stress_zx`. You can change the tensor component separator, an underscore in this example, by using the `COMPONENT SEPARATOR CHARACTER` command line (see Section 9.2.1.7).

The `ELEMENT VARIABLES` command line can be used an arbitrary number of times within a `RESULTS OUTPUT` command block. It is also possible to specify more than one element variable for output on a command line, as indicated by the ellipsis in the command line format. In the following example, two element variables are specified for output. Here, the second element variable is defined as `left_stretch`.

```
ELEMENT VARIABLES = stress left_stretch
```

Aliases can be specified for each of the variables in a single command line. Thus, If you wanted to output the alias `str` for element variable `stress` and also wanted to output the alias `strch` for face variable `left_stretch`, you would enter the command line:

```
ELEMENT VARIABLES = stress AS str
     left_stretch AS strch
```

The specification of an alias is optional.

The second form of the command line is as follows:

```
ELEMENT VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name]
  INCLUDE|ON|EXCLUDE <string list>block_names
  ... <string>variable_name
```

```
[AS dbase_variable_name] INCLUDE|ON|EXCLUDE
<string list>block_names
```

This form of the ELEMENT VARIABLES command line is similar to the first, except that the user can control which element blocks are used for output. The user can include a specific list of element blocks for output by using the INCLUDE option or the ON option. (The keyword INCLUDE is synonymous with the keyword ON.) Alternatively, the user can exclude a specific list of element blocks for output by using the EXCLUDE option.

Suppose that the element variable stress from the above example exists for element blocks block_10, block_11, block_20, and block_21. If we only want to output the element variable for block_10, block_11, and block_21, then we could specify the ELEMENT VARIABLES command line as follows:

```
ELEMENT VARIABLES = stress AS str
     INCLUDE block_10, block_11,
     block_21
```

(In the above command line, the keyword ON could be substituted for INCLUDE.) Alternatively, we could use the command line:

```
ELEMENT VARIABLES = stress AS str
     EXCLUDE block_20
```

In the above command lines, an alias for an element block can be substituted for an element block identifier. For example, if center_case is an alias for block_10, then the string center_case could be substituted for block_10 in the above command lines. Because an element block identifier is a mesh entity, the alias for the element block identifier would be defined via an ALIAS command line in a FINITE ELEMENT MODEL command block. Note that the list of identifiers uses a comma to separate one element block identifier from the next element block identifier.

For multi-integration point elements, quantities from the integration points are appended with a numerical index indicating the integration point. A suffix ranging from 1 to the number of integration points is attached to the quantity to indicate the corresponding integration point. The suffix is padded with leading zeros. If the number of integration points is less than 10, the suffix has the form _*i*, where *i* ranges from 1 to the number of integration points. If the number of integration points is greater than or equal to 10 and less than 100, the sequence of suffixes takes the form _01, _02, _03, and so forth. Finally, if the number of integration points is greater than or equal to 100, the sequence of suffixes takes the form _001, _002, _003, and so forth. As an example, if the von Mises stress is requested for a shell element with 15 integration points, then the quantities von_mises_01, von_mises_02, ..., von_mises_15 are output for the shell element.

In the above discussion concerning the format for output at multiple integration points, the underscore character preceding the integration point number can be replaced by another delimiter or the underscore character can be eliminated by use of the COMPONENT SEPARATOR CHARACTER command line (see Section 9.2.1.7).

Shell tensor quantities transform_shell_stress, transform_shell_strain and transform_shell_rate_of_deformation may be transformed to a user specified shell local co-rotational coordinate system (i.e. an in-plane coordinate system that rotates with the shell

element) for output using the ORIENTATION shell section command. If no orientation is specified, these in-plane stresses and strains are output in the default orientation. See Section 6.2.4 for more details.

9.2.1.5 Subsetting of Output Mesh

A specified subset of the entities (element blocks, nodesets, surfaces) in the mesh can be output to the results database using the INCLUDE or EXCLUDE commands. The syntax is:

```
INCLUDE = <string>list_of_included_entities
EXCLUDE = <string>list_of_excluded_entities
```

Either command can appear multiple times within the results output block, but a specific entity cannot be specified in both. If the INCLUDE command is specified, the results database will only contain the listed entities of that type; if the EXCLUDE command is specified, the results database will contain all entities of the type listed except for the listed entities. If the model has surfaces or nodesets, only the portion of the surfaces or nodesets on the selected element blocks will be output.

9.2.1.6 Output Mesh Selection

```
OUTPUT MESH = EXPOSED_SURFACE|BLOCK_SURFACE
```

The OUTPUT MESH command provides a way to reduce the amount of data that is written to the results database. There are two options that can be selected:

EXPOSED_SURFACE Only output the element faces that make up the “skin” of the finite element model; no internal nodes or elements will be written to the results database. The element results variables will be applied to the skin faces. If the mesh is visualized without any cutting planes, the display should look the same as if the original full mesh were visualized; however, the amount of data written to the output file can be much less than is needed if the full mesh were output.

BLOCK_SURFACE This option is similar to the EXPOSED_SURFACE option except that the skinning process is done an element block at a time instead of for the full model. In this option, faces shared between element blocks will appear in the output model.

9.2.1.7 Component Separator Character

```
COMPONENT SEPARATOR CHARACTER = <string>character|NONE
```

The component separator character is used to separate an output-variable base name from any suffixes. For example, the variable stress can have the suffixes xx, yy, etc. By default, the base name is separated from the suffixes with an underscore character so that we have stress_xx, stress_yy, etc. in the results output file.

You can replace the underscore as the default separator by using the above command line. If you wanted to use the period as the separator, then you would use the following command line:

```
COMPONENT SEPARATOR CHARACTER = .
```

For our example with stress, the stress components would then appear in the results output file as stress.xx, stress.yy, etc. If the stress is for a shell element, there is also an integration point suffix preceded, by default, with an underscore. The above command line also resets the underscore character that precedes the integration point suffix. For our example with the stress base name and the underscore replaced by the period, the results file would have stress.xx.01, stress.xx.02, etc., for the shell elements.

You can eliminate the separator with an empty string or NONE.

9.2.1.8 Output Global Variables

```
GLOBAL VARIABLES =  <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
```

Any global variable in Sierra/SM can be selected for output in the results file by using the GLOBAL VARIABLES command line. The string variable_name is the name of the global variable. The string variable_name can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

Kinetic, external, hourglass, and internal energies can be requested as the sum over the entire mesh or as sums over individual element blocks as noted in Section 9.9. For example, if the mesh contains element blocks with IDs 100 and 200, the kinetic and hourglass energy summed over each of these blocks individually can be requested with the commands

```
GLOBAL VARIABLES = ke_block100 as ke100
GLOBAL VARIABLES = ke_block200 as ke200
GLOBAL VARIABLES = hge_block100 as hge100
GLOBAL VARIABLES = hge_block200 as hge200
```

and the total kinetic energy as

```
GLOBAL VARIABLES = kinetic_energy as ke
```

Kinetic, internal, and external energies are computed as nodal values. Since nodes may be shared by element blocks, the total sum of an energy quantity over individual element blocks will not necessarily be equal to the global sum requested using the kinetic_energy, internal_energy, or external_energy variable names.

With the AS specification, you can specify the variable and select an alias for this variable in the results file. Suppose, for example, you wanted to output the time steps in Sierra/SM, which are identified as timestep, with the alias tstep. You would then enter the command line:

```
GLOBAL VARIABLES = timestep AS tstep
```

The GLOBAL VARIABLES command line can be used an arbitrary number of times within a RESULTS OUTPUT command block. It is also possible to specify more than one global variable for output on a command line. If you also wanted to output the CPU time, which is defined as cpu_time, with the alias cpu, you would enter the command line:

```
GLOBAL VARIABLES = timestep as tstep cpu_time as cpu
```

The specification of an alias is optional.

9.2.1.9 Set Begin Time for Results Output

```
START TIME = <real>output_start_time
```

Using the START TIME command line, you can write output to the results file beginning at time output_start_time. No results will be written before this time. If other commands set times for results (AT TIME, ADDITIONAL TIMES) that are less than output_start_time, those times will be ignored, and results will not be written at those times.

9.2.1.10 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value steps in the TIMESTEP ADJUSTMENT INTERVAL command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, results are output at times closest to the specified output times.

9.2.1.11 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by time_begin, results will be output every time increment given by the real value time_increment_dt.

9.2.1.12 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 9.2.1.11, you can use the ADDITIONAL TIMES command line to specify an arbitrary number of additional output times.

9.2.1.13 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by `step_begin`, results will be output every step increment given by the integer value `step_increment`.

9.2.1.14 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 9.2.1.13, you can use the `ADDITIONAL STEPS` command line to specify an arbitrary number of additional output steps.

9.2.1.15 Set End Time for Results Output

```
TERMINATION TIME = <real>termination_time_value
```

Results will not be written to the results file after time `termination_time_value`. If other commands set times for results (`AT TIME`, `ADDITIONAL TIMES`) that are greater than `termination_time_value`, those times will be ignored, and results will not be written at those times.

9.2.1.16 Synchronize Output

```
SYNCHRONIZE OUTPUT
```

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the `SYNCHRONIZE OUTPUT` command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the `USE OUTPUT SCHEDULER` command line can also synchronize output between regions, the `SYNCHRONIZE OUTPUT` command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with `SYNCHRONIZE OUTPUT` specified will also synchronize its output with the output of the external code.

The `SYNCHRONIZE OUTPUT` command can be used with other output scheduling commands such as time-based or step-based output specifications.

9.2.1.17 Use Output Scheduler

```
USE OUTPUT SCHEDULER <string>scheduler_name
```

In an analysis with multiple regions, it can be difficult to synchronize output such as results files. To help synchronize output for analyses with multiple regions, you can define an OUTPUT SCHEDULER command block at the SIERRA scope. The scheduler can then be referenced in the RESULTS OUTPUT command block via the USE OUTPUT SCHEDULER command line. The string scheduler_name must match a name used in an OUTPUT SCHEDULER command block. See Section 9.6 for a description of using this command block and the USE OUTPUT SCHEDULER command line.

9.2.1.18 Write Results If System Error Encountered

```
OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
  SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
  SIGKILL|SIGILL|SIGSEGV
```

The OUTPUT ON SIGNAL command line is used to initiate the writing of a results file when the system encounters a type of system error. Only one error type in the list of error types should be entered for this command line. Generally, these system errors cause the code to terminate before the code can add any current results output (results output past the last results output time step) to the results output file. If the code encounters the specified type of error during execution, a results file will be written before execution is terminated.

This command line can also be used to force the writing of a results file at some point during execution of the code. Suppose the command line

```
OUTPUT ON SIGNAL = SIGUSR2
```

is included in the input file. While the code is running, a user can execute (from the keyboard) the system command line

kill -s SIGUSR2 *pid*

to terminate execution and force the writing of a results file. In the above system command line, *pid* is the process identifier, which is an integer.

Note that the OUTPUT ON SIGNAL command line is primarily a debugging tool for code developers.

9.2.2 User-Defined Output

```
BEGIN FILTER <string>filter_name
  ACOEFF = <real_list>a_coeff
  BCOEFF = <real_list>b_coeff
  INTERPOLATION TIME STEP = <real>ts
END [FILTER]

BEGIN USER OUTPUT
  # mesh-entity set commands
  NODE SET = <string_list>nodelist_names
  SURFACE = <string_list>surface_names
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list> surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # compute result commands
  COMPUTE GLOBAL <string>result_var_name AS
    SUM|AVERAGE|MAX|MIN|MAX ABSOLUTE VALUE
    OF NODAL|ELEMENT <string>source_var_name
  COMPUTE NODAL <string>result_var_name AS
    MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
    OF NODAL <string>source_var_name
  COMPUTE ELEMENT <string>result_var_name AS
    MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
    OF ELEMENT <string>source_var_name
  COMPUTE ELEMENT <string>result_var_name AS
    MAX|MIN|AVERAGE OF ELEMENT <string>source_var_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>subroutine_name |
    ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # copy command
  COPY ELEMENT VARIABLE <string>ev_name TO NODAL VARIABLE
    <string>nv_name
  #
```

```
    # variable transformation command
    TRANSFORM NODAL|ELEMENT VARIABLE <string>variable_name
      TO COORDINATE SYSTEM <string>coord_sys_name
      AS <string> transformed_name
    #
    # data filtering
    FILTER <string>new_var FROM NODAL|ELEMENT <string>source_var
      USING <string>filter_name
    #
    # compute for element death
    COMPUTE AT EVERY TIME STEP
    #
    # additional command
    ACTIVE PERIODS = <string list>period_names
    INACTIVE PERIODS = <string list>period_names
  END [USER OUTPUT]
```

The USER OUTPUT command block lets the user generate specialized output information derived from analysis results such as element stresses, displacements, and velocities. For example, the USER OUTPUT command block could be used to sum the contact forces in a particular direction in the global axes and on a certain surface to give a net resultant contact force on that surface. In this example, we essentially postprocess contact information and reduce it to a single value for a surface (or set of surfaces). This, then, is one of the purposes of the USER OUTPUT command block—to postprocess analysis results as the code is running and produce a reduced set of specialized output information. The USER OUTPUT command block offers an alternative to writing out large quantities of data and then postprocessing them with an external code to produce specialized output results.

Another use of the USER OUTPUT command block is to generate variables that can be used for element death. In explicit dynamics analyses, an element can be killed by using a criterion based on a user variable defined in the USER OUTPUT command block. If user variable is to be used in this manner, the COMPUTE AT EVERY TIME STEP command must be used to ensure that the variables gets computed at every step. See Section 9.2.2.7.

There are three options for calculating user-defined quantities. In the first option, a single command line in the command block is used to compute reductions of variables on subsets of the mesh. This option makes use of one of the compute result command lines. For instance, the above example of the contact force is a case where we can accomplish the desired result simply by using the COMPUTE GLOBAL command line. In the second option, the command block specifies a user subroutine to run immediately preceding output to calculate any desired variable. This option makes use of a NODE SET, SURFACE, or ELEMENT BLOCK SUBROUTINE command line. Finally, there is an option to copy an element variable for an element to the nodes associated with the element, via the COPY ELEMENT VARIABLE command line. This copy option is a specialized option that has been made available primarily for creating results files for some of the postprocessing tools used with Sierra/SM. You can use only one of the three options—compute global result, user subroutine, or copy—in a given command block.

For the compute result option, a user-defined variable is automatically generated. This user-defined variable is given whatever name the user selects for `results_var_name` in the above specification for any of the three compute result command lines. Parenthesis syntax (Section 9.1) may be used to define reductions on specific integration points or components of a variable. By default, a reduction operation operates on each integration point. For example, if the compute global command was used to average values of element stress it would average the values of stress at all integration points of all elements. If the command was used to average `stress(:,1)`, the result would only be the average of stress on the first integration points.

If the user subroutine or copy option is used, the user will need to define some type of user variable with the `USER VARIABLE` command block described in Section 11.2.4.

User-defined variables, whether they are generated via the compute result option or the `USER VARIABLE` command block, are not automatically written to a results or history file. If the user wants to output any user-defined variables, these variables must be referenced in a results or history output specification (see Section 9.2.1 and Section 9.3, which describe the output of variables to results files and history files, respectively).

The `USER OUTPUT` command block contains five groups of commands—mesh-entity set, compute result, user subroutine, copy, and compute for element death. Each of these command groups is basically independent of the others. In addition to the command lines in the five command groups, there is an additional command line: `ACTIVE PERIODS`. The following sections provide descriptions of the different command groups and the `ACTIVE PERIODS` command line.

9.2.2.1 Mesh-Entity Set Commands

The `mesh-entity set commands` portion of the `USER OUTPUT` command block specifies the nodes, element faces, or elements associated with the variable to be output. This portion of the command block can include some combination of the following command lines:

```
NODE SET = <string_list>nodelist_names
SURFACE = <string_list>surface_names
BLOCK = <string_list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list> surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes, element faces, or elements. See Section 7.1.1 for more information about the use of these command lines for mesh entities. There must be at least one `NODE SET`, `SURFACE`, `BLOCK`, or `INCLUDE ALL BLOCKS` command line in the command block.

9.2.2.2 Compute Result Commands

The compute result commands are used to compute new variables by performing operations on an existing variable. Currently four general forms of the compute results command are supported:

```
COMPUTE GLOBAL <string>result_var_name AS
  SUM|AVERAGE|MAX|MIN|MAX ABSOLUTE VALUE
  OF NODAL|ELEMENT <string>source_var_name
COMPUTE NODAL <string>result_var_name AS
  MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
   OF NODAL <string>source_var_name
COMPUTE ELEMENT <string>result_var_name AS
  MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
   OF ELEMENT <string>source_var_name
COMPUTE ELEMENT <string>result_var_name AS
  MAX|MIN|AVERAGE OF ELEMENT <string>source_var_name
```

The compute global result command returns a single global value or a set of global values by examining the current values for a named nodal or element variable and then calculating the output according to a user-specified operation. A single global value, for example, might be the maximum value of one of the stress components of all the elements in our specified set; a set of global values would be the maximum value of each stress component of all elements in our specified set.

The compute nodal and element variable commands compute a new nodal or element field by operating on an existing field. These commands can be used for computing such things as the maximum stress in each element over the course of the analysis.

In the above command lines, the following definitions apply:

- The string `result_var_name` is the name of a new variable in which the computed results are stored. To output this variable in a results file, a heartbeat file, or a history file, you will simply use whatever you have selected for `results_var_name` as the variable name in the output block.
- Many different global reduction methods are available for computing global include variables: `SUM`, `AVERAGE`, `MAX`, `MIN`, and `MAX ABSOLUTE VALUE`. `SUM` adds the variable value of all included mesh entities. `AVERAGE` takes the average value of the variable over all included mesh entities. `MAX` finds the maximum value over all included mesh entities. `MIN` finds the minimum value over all included mesh entities. `MAX ABSOLUTE VALUE` finds the maximum absolute value of the quantity rather than the maximum signed value of a quantity.
- Three different over time methods are available for computing nodal or element variables: `MAX OVER TIME`, `MIN OVER TIME`, and `ABSOLUTE VALUE MAX OVER TIME`. These options compute the max, min, or absolute max over time of values in the source variable.
- If both the source variable and target variable are element variables, and `MAX`, `MIN`, or `AVERAGE` is used, then the operation computes a single element variable based on a reduction of the values at all integration points for that element.

- The source variable used to compute a global variable must be either a nodal quantity or an element quantity, as specified by the NODAL or ELEMENT option. The variable source variable used to compute a nodal or element variable must be of the same type as the result variable.

- The string source_var_name is the name of the variable used to compute the result variable. (see Section 9.9 for a listing of available code variables).

- Standard component syntax may also be used in the source_variable_name to specify operating on sub-components of a given variable (such as only stress(xx)).

The following is an example of using the COMPUTE command line to compute the net *x*-direction reaction force:

```
COMPUTE GLOBAL wall_x_reaction AS SUM OF NODAL reaction(x)
```

The following is an example of using the COMPUTE command line to compute the maximum force seen by the spot welds:

```
  COMPUTE NODAL max_spotn AS MAXIMUM OVER TIME OF NODAL
spot_weld_normal_force
```

The following is an example of using the COMPUTE command line to compute the average value of eqps in an element with multiple integration points:

```
COMPUTE ELEMENT eqps_avg AS AVERAGE OF ELEMENT eqps
```

9.2.2.3 User Subroutine Commands

If the user subroutine option is used, the user-defined output quantities will be calculated by a subroutine that is written by the user explicitly for this purpose. The subroutine will be called by Sierra/SM at the appropriate time to perform the calculations. User subroutines allow for more generality in computing user-defined results than the COMPUTE GLOBAL command line. Suppose, for example, you had an analytic solution for a problem and wanted to compute the difference between some analytic value and a corresponding computed value throughout an analysis. The user subroutine option would allow you to make this comparison. The full details for user subroutines are given in Chapter 11.

The following command lines are related to the user subroutine option:

```
NODE SET SUBROUTINE = <string>subroutine_name |
  SURFACE SUBROUTINE = <string>subroutine_name |
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
```

```
    = <string>param_value
```

The user subroutine option is invoked by using the NODE SET SUBROUTINE command line, the SURFACE SUBROUTINE command line, or the ELEMENT BLOCK SUBROUTINE command line. The particular command line selected depends on the mesh-entity type of the variable for which the result quantities are being calculated. For example, variables associated with nodes would be calculated by using a NODE SET SUBROUTINE command line, variables associated with faces by using a SURFACE SUBROUTINE command line, and variables associated with elements by using the ELEMENT BLOCK SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user. A user subroutine in the USER OUTPUT command block returns no values. Instead, it performs its operations directly with commands such as aupst_put_nodal_var, aupst_put_elem_var, and aupst_put_global_var. See Chapter 11 for further discussion of these various put commands.

Following the selected command line (NODE SET SUBROUTINE, SURFACE SUBROUTINE, or ELEMENT BLOCK SUBROUTINE) are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided in Chapter 11.

Importantly, to implement the user subroutine option and output the calculated information, you would also need to do the following:

1. Create the user-defined variable with a USER VARIABLE command block.
2. Calculate the results for the user-defined variable in the user subroutine.
3. Write the results for the user-defined variable to an output file by referencing it in a RESULTS OUTPUT command block and/or a HISTORY OUTPUT command block and/or a HEARTBEAT OUTPUT command block. In the RESULTS OUTPUT command block, you would use a NODAL command line, an ELEMENT command line, or a GLOBAL command line, depending on how you defined the variable in the USER VARIABLE command block. Similarly, in the HISTORY OUTPUT or HEARTBEAT OUTPUT command block, you would use the applicable form of the variable command line, depending on how you defined the variable in the USER VARIABLE command block.

9.2.2.4 Copy Command

```
COPY ELEMENT VARIABLE <string>ev_name TO NODAL VARIABLE
  <string>nv_name
```

The COPY ELEMENT VARIABLE command line copies the value of an element variable to a node associated with the element. The element variable to be copied is specified by ev_name; the name of the nodal variable to which the value is being transferred is nv_name. The nodal variable must be specified as a user-defined variable.

9.2.2.5 Variable Transformation Command

```
TRANSFORM NODAL|ELEMENT VARIABLE <string>variable_name
  TO COORDINATE SYSTEM <string>coord_sys_name
  AS <string> transformed_name
```

The TRANSFORM NODAL|ELEMENT VARIABLE command line transforms a nodal vector (displacement, velocity, acceleration, etc) or an element tensor (stress, strain, etc.) from components in the global coordinate system to components in the coordinate system defined in a BEGIN COORDINATE SYSTEM command block having the name coord_sys_name (see Section 2.1.8). The transformed variables will be output to the results file as transformed_name.

Warning: This command cannot be used to transform shell element tensors which are not computed in the global coordinate system. Output of shell stress and strain components in a user-defined, local co-rotational coordinate system are obtained with the element variables transform_shell_stress and transform_shell_strain as described in Section 9.2.1.

9.2.2.6 Data Filtering Commands

```
BEGIN FILTER <string>filter_name
  ACOEFF = <real_list>a_coeff
  BCOEFF = <real_list>b_coeff
  INTERPOLATION TIME STEP = <real>ts
END [FILTER]
```

```
BEGIN USER OUTPUT
  FILTER <string>new_var FROM NODAL|ELEMENT <string>source_var
    USING <string>filter_name
END
```

The user output FILTER command creates a new variable “new_var” by performing an on-the-fly frequency filter of the named element or nodal variable “source_var”. The BEGIN FILTER command block defines a filter with the name “filter_name”. The filter defined by the begin filter command block is then referenced by the USER OUTPUT filter command.

In the BEGIN FILTER command block the A and B filtering coefficients are defined with the ACOEFF and BCOEFF command lines. The filter must define at least one A and one B coefficient, there is no maximum number coefficients and the length of A and B do not need to match.

Filter operations assume that the data given at a constant time step. The explicit dynamics time step will tend to vary during the analysis. The INTERPOLATION TIME STEP command is used to linearly interpolate the data that is being produced at a non-constant time step down to some specified constant time step. The interpolation time step must be larger than zero and ideally

should be specified such that it is smaller than the smallest time step with which the computations will iterate.

One way to obtain the filtering coefficients is with MATLAB. The following is an example of defining a third order Butterworth filter with a pass frequency of 100Hz at data interpolated to a time step of 1.0e-5 seconds. The filter is then used to filter acceleration histories of the nodes to 100Hz. The MATLAB code below will give the desired filtering coefficients. Note that the full 16-digit precision of the coefficients returned by MATLAB should be used. If truncated precision numbers are used, the filters can potentially be unstable.

```
clear;
format long e;
passFrequency = 100;
interp_ts = 1.0e-5;
butterCoeff = 2.0*interp_ts*passFrequency;
[bcoeff,acoeff] = butter(3,butterCoeff);
acoeff
bcoeff
```

The computed filtering coefficients can be used in a `USER OUTPUT` block as show below. If the analysis time step always remains above 1.0e-5 the filter will be valid. If the analysis time step drops below 1.0e-5 there could be aliasing issues, and a smaller interpolation time step should be specified.

```
BEGIN FILTER filt_100Hz
  ACOEFF = 1.000000000000000e+00 -2.987433650055722e+00 $
           2.974946132665442e+00 -9.875122361107358e-01
  BCOEFF = 3.081237301416628e-08  9.243711904249885e-08 $
           9.243711904249885e-08  3.081237301416628e-08
  INTERPOLATION TIME STEP = 1.0e-5
END
BEGIN USER OUTPUT
  FILTER ax100Hz FROM NODAL acceleration(x) USING filt_100Hz
  FILTER ay100Hz FROM NODAL acceleration(y) USING filt_100Hz
  FILTER az100Hz FROM NODAL acceleration(z) USING filt_100Hz
END
```

9.2.2.7 Compute at Every Step Command

```
COMPUTE AT EVERY TIME STEP
```

If this command line appears in the `USER OUTPUT` command block, a user-defined variable in the command block will be written at every time step. Section 11.2.4 discusses user-defined variables.

In explicit dynamics analyses, user-defined variables can be used as criteria for element death. In that case, the `COMPUTE AT EVERY TIME STEP` command must be used to force those variables to be computed at every time step. See Section 6.5 for more information on element death.

9.2.2.8 Additional Command

The ACTIVE PERIODS or INACTIVE PERIODS command lines can appear as an option in the USER OUTPUT command block:

```
ACTIVE PERIODS = <string list>period_names
```

```
INACTIVE PERIODS = <string list>period_names
```

These command lines determine when the boundary condition is active. See Section 2.5 for more information about this optional command line.

9.3 History Output

```
BEGIN HISTORY OUTPUT <string>history_name
  DATABASE NAME = <string>history_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
    (ON|TRUE|YES)
  TITLE <string>user_title
  #
  # for global variables
  GLOBAL <string>variable_name
    [AS <string>history_variable_name]
  #
  # for mesh entity - node, edge, face,
  # element - variables
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
    AS <string>history_variable_name
  #
  # for nearest point output of mesh entity - node,
  # edge, face, element - variables
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    NEAREST LOCATION <real>global_x,
      <real>global_y>, <real>global_z
    AS <string>history_variable_name
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  SYNCHRONIZE_OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
END [HISTORY OUTPUT <string>history_name]
```

Explicit (margin marker beside TIMESTEP ADJUSTMENT INTERVAL)

A history file gives nodal variable results (displacements, forces, etc.) for specific nodes, edge variable results for specific edges, face variable results for specific faces, element results (stress, strain, etc.) for specific elements, and global results at specified times. You can specify a history file, the results to be included in this file, and the frequency at which results are written by using

a HISTORY OUTPUT command block. The command block appears inside the region scope. For history output, you will typically work with node and element variables, and, on some occasions, face variables.

More than one history file can be specified for an analysis. For each history file, there will be one HISTORY OUTPUT command block. The command block for a history file description begins with:

```
BEGIN HISTORY OUTPUT <string>history_name
```

and is terminated with:

```
END [HISTORY OUTPUT <string>history_name]
```

where history_name is a user-selected name for the command block. Nested within the HISTORY OUTPUT command block are a set of command lines, as shown in the block summary given above. The first two command lines listed (DATABASE NAME and DATABASE TYPE) give pertinent information about the history file. The command line

```
DATABASE NAME = <string>history_file_name
```

gives the name of the history file with the string history_file_name. If the history file is to appear in the current directory and is named job.h, this command line would appear as:

```
DATABASE NAME = job.h
```

If the history file is to be created in some other directory, the command line would have to show the path to that directory.

Two metacharacters can appear in the name of the history file. If the %P character is found in the name, it will be replaced with the number processors being used for the run. For example, if you are running on 1024 processors and use the name history-%P/job.h, then the name would be expanded to history-1024/job.h. The other recognized metacharacter is %B which is replaced with the base name of the input file containing the input commands. For example, if the commands are in the file my_analysis_run.i and the history database name is specified as %B.h, then the history would be written to the file my_analysis_run.h.

If the history file does not use the Exodus II format [1], you must specify the format for the history file using the command line:

```
DATABASE TYPE = <string>database_type(exodusII)
```

Currently, both the Exodus II database and the XDMF database [2] are supported in Presto and Adagio. Exodus II is more commonly used than XDMF. Other options may be added in the future.

The OVERWRITE command line can be used to prevent the overwriting of existing history files.

```
OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
     (ON|TRUE|YES)
```

The OVERWRITE command line allows only a single value. If you set the value to FALSE, NO, or OFF, the code will terminate before existing history files can be overwritten. If you set the value to TRUE, YES, or ON, then existing history files can be overwritten (the default status). Suppose, for example, that we have an existing history file named job21.h. Suppose also that we have an

input file with a HISTORY OUTPUT command block that contains the OVERWRITE command line set to ON and the DATABASE NAME command line set to:

```
DATABASE NAME = job21.h
```

If you run the code under these conditions, the existing history file job21.h will be overwritten.

Whether or not history files are overwritten is also impacted by the use of the automatic read and write option for restart files described in Section 9.5.1.1. If you use the automatic read and write option for restart files, the history files, like the restart files, are automatically managed. The automatic read and write option in restart adds extensions to file names and prevents the overwriting of any existing restart or history files. For the case of a user-controlled read and write of restart files (Section 9.5.1.2) or of no restart, however, the OVERWRITE command line is useful for preventing the overwriting of history files.

You may add a title to the history file by using the TITLE command line. Whatever you specify for the user_title will be written to the history file. Some of the programs that process the history file (such as various SEACAS programs [3]) can read and display this information.

The other command lines that appear in the HISTORY OUTPUT command block determine the type and frequency of information that is output. Descriptions of these command lines follow in Section 9.3.1 through Section 9.3.12. Note that the command lines for controlling the frequency of history output (in Section 9.3.1 through Section 9.3.12) are the same as those for controlling the frequency of results output. These frequency-related command lines are repeated here for convenience.

9.3.1 Output Variables

The GLOBAL, NODE (or NODAL), EDGE, FACE, or ELEMENT command line is used to select variables for output in the history file. One of several types of variables can be selected for output. The form of the command line varies depending on the type of variable that is selected for output.

9.3.1.1 Global Output Variables

```
GLOBAL <string>variable_name
  [AS <string>history_variable_name]
```

This form of the command line lets you select any global variable for output in the history file. The variable is selected with the string variable_name. The string variable_name is the name of the global variable and can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

You can also specify a name, history_variable_name, for the selected entity following the AS keyword. For example, suppose you want to output the CPU time (cpu_time) as cpu in the history file. The command line to accomplish this would be:

```
GLOBAL cpu_time AS cpu
```

The specification of an alias is optional for output of a global variable.

9.3.1.2 Mesh Entity Output Variables

```
NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
  AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
  AS <string>history_variable_name
```

This form of the command line lets you select any nodal, edge, face, or element variable for a specific mesh entity for output in the history file. For example, this form of the command line will let you pick the displacement at a specific node and output the displacement to the history file using an alias that you have chosen.

For this form of the command line, the mesh entity type is set to NODE (or NODAL), EDGE, FACE, or ELEMENT depending on the variable (set by variable_name) to be output. If the mesh entity type is set to NODE (or NODAL), EDGE, or FACE, the string variable_name can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4). If the mesh entity type is set to ELEMENT, the string variable_name can be a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4.

Selection of a specific mesh entity follows the AT keyword. You select a mesh entity type (NODE [or NODAL], EDGE, FACE, or ELEMENT) followed by the specific integer identifier, entity_id, for the mesh entity. You must specify a name, history_variable_name, for the selected entity following the AS keyword. For example, suppose you want to output the accelerations at node 88. The command line to obtain the accelerations at node 88 for the history file would be:

```
NODE ACCELERATION AT NODE 88 AS accel_88
```

where accel_88 is the name that will be used for this history variable in the history file.

Note that either the keyword NODE or NODAL can be used for nodal quantities.

As an example, the command line to obtain the von Mises stress for element 1024 for the history file would be:

```
ELEMENT VON_MISES AT ELEMENT 1024 AS vm_1024
```

where vm_1024 is the name that will be used for this history variable in the history file.

9.3.1.3 Nearest Point Output Variables

```
NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
  NEAREST LOCATION <real>global_x,
    real<global_y>, real<global_z>
  AS <string>history_variable_name
```

This form of the command line lets you select any nodal, edge, face, or element variable for output in the history file using a nearest point criterion. The command line described in this subsection is an alternative to the command line described in the preceding section, Section 9.3.1.2, for obtaining history output. The command line in this section or the command line in Section 9.3.1.2 produces history files with variable information. The difference in these two command lines (Section 9.3.1.3 and Section 9.3.1.2) is simply in how the variable information is selected.

For the above form of the command line, the mesh entity type is set to NODE (or NODAL), EDGE, FACE, or ELEMENT depending on the variable (set by variable_name) to be output. If the mesh entity type is set to NODE (or NODAL), EDGE, or FACE, the string variable_name can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4). If the mesh entity type is set to ELEMENT, the string variable_name can be a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

The specific mesh entity used for output is determined by global coordinates specified by the NEAREST LOCATION keyword and its associated input parameters—global_x, global_y, global_z. The specific mesh entity chosen for output is as follows:

- If the mesh entity has been set to NODE (or NODAL), the node in the mesh selected for output is the node whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters global_x, global_y, and global_z.
- If the mesh entity has been set to EDGE, the edge in the mesh selected for output is the edge with a center point (the average location of the two end points of the edge) whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters global_x, global_y, and global_z.
- If the mesh entity has been set to FACE, the face in the mesh selected for output is the face with a centroid whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters global_x, global_y, and global_z.
- If the mesh entity has been set to ELEMENT, the element in the mesh selected for output is the element with a centroid whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters global_x, global_y, and global_z.

Note that, in all the above cases, the original model coordinates are used when selecting the nearest entity, not the current coordinates.

You must specify a name, history_variable_name, for the selected entity following the AS keyword. As an example, suppose you want to output the accelerations at a node closest to the point with global coordinates (1012.0, 54.86, 103.3141). The command line to obtain the accelerations at the node closest to this location for the history file would be:

```
NODE ACCELERATION
     NEAREST LOCATION 1012.0, 54.86, 103.3141 AS accel_near
```

where accel_near is the name that will be used for this history variable in the history file.

Note that either the keyword NODE or NODAL can be used for nodal quantities.

9.3.2 Outputting History Data on a Node Set

It is commonly desired to output history data on a single-node node set. If a mesh file is slightly modified, the node and element numbers will completely change. The node associated with a node set, however, remains the same, i.e., the node in the node set retains the same initial geometric

location with the same connectivity to other elements even when its node number changes. Therefore, we might want to specify the history output for a node set with a single node rather than with the global identifier for a node. This can easily be accomplished, as follows:

```
 begin user output
   node set = nodelist_1
   compute global disp_ns_1 as average of nodal displacement
 end

begin history output
  global disp_ns_1
end
```

If nodelist_1 contains only a single node, the history output variable disp_ns_1 will contain the displacement for the single node in the node set. If nodelist_1 contains multiple nodes, the average displacement of the nodes will be output.

9.3.3 Set Begin Time for History Output

```
START TIME = <real>output_start_time
```

Using the START TIME command line, you can write history variables to the history file beginning at time output_start_time. No history variables will be written before this time. If other commands set times for history output (AT TIME, ADDITIONAL TIMES) that are less than output_start_time, those times will be ignored, and history output will not be written at those times.

9.3.4 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value steps in the TIMESTEP ADJUSTMENT INTERVAL command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, history variables are output at times closest to the specified output times.

9.3.5 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by time_begin, history variables will be output every time increment given by the real value time_increment_dt.

9.3.6 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 9.3.5, you can use the ADDITIONAL TIMES command line to specify an arbitrary number of additional output times.

9.3.7 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by step_begin, history variables will be output every step increment given by the integer value step_increment.

9.3.8 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 9.3.7, you can use the ADDITIONAL STEPS command line to specify an arbitrary number of g

9.3.9 Set End Time for History Output

```
TERMINATION TIME = <real>termination_time_value
```

History output will not be written to the history file after time termination_time_value. If other commands set times for history output (AT TIME, ADDITIONAL TIMES) that are greater than termination_time_value, those times will be ignored, and history output will not be written at those times.

9.3.10 Synchronize Output

```
SYNCHRONIZE OUTPUT
```

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of history data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the history output block. If a history block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A history block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

9.3.11 Use Output Scheduler

```
USE OUTPUT SCHEDULER <string>scheduler_name
```

In an analysis with multiple regions, it can be difficult to synchronize output such as history files. To help synchronize output for analyses with multiple regions, you can define an OUTPUT SCHEDULER command block at the SIERRA scope. The scheduler can then be referenced in the HISTORY OUTPUT command block via the USE OUTPUT SCHEDULER command line. The string scheduler_name must match a name used in an OUTPUT SCHEDULER command block. See Section 9.6 for a description of using this command block and the USE OUTPUT SCHEDULER command line.

9.3.12 Write History If System Error Encountered

```
OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
  SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
  SIGKILL|SIGILL|SIGSEGV
```

The OUTPUT ON SIGNAL command line is used to initiate the writing of a history file when the system encounters a type of system error. Only one error type in the list of error types should be entered for this command line. Generally, these system errors cause the code to terminate before the code can add any current history output (history output past the last history output time step) to the history file. If the code encounters the specified type of error during execution, a history file will be written before execution is terminated.

This command line can also be used to force the writing of a history file at some point during execution of the code. Suppose the command line

```
OUTPUT ON SIGNAL = SIGUSR2
```

is included in the input file. While the code is running, a user can execute (from the keyboard) the system command line

kill -s SIGUSR2 *pid*

to terminate execution and force the writing of a results file. In the above system command line, *pid* is the process identifier, which is an integer.

Note that the `OUTPUT ON SIGNAL` command line is primarily a debugging tool for code developers.

9.4 Heartbeat Output

```
BEGIN HEARTBEAT OUTPUT <string>heartbeat_name
  # Can also use predefined streams "cout", "stdout",
  # "cerr", "clog", "log", "output", or "outputP0"
  STREAM NAME = <string>heartbeat_file_name
  #
  # Specify whether heartbeat file will be in spyhis (cth)
  # format, or default format
  FORMAT = SPYHIS|DEFAULT
  #
  # for global variables
  GLOBAL <string>variable_name
    [AS <string>heartbeat_variable_name]
  #
  # for mesh entity - node, edge, face,
  # element - variables
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
    AS <string>heartbeat_variable_name
  #
  # for nearest point output of mesh entity - node,
  # edge, face, element - variables
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    NEAREST LOCATION <real>global_x,
      <real>global_y>, <real>global_z
    AS <string>heartbeat_variable_name
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  SYNCHRONIZE_OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
  PRECISION = <integer>precision
  LABELS = <string>OFF|ON
  LEGEND = <string>OFF|ON
  TIMESTAMP FORMAT <string>timestamp_format
```

Explicit (margin note beside TIMESTEP ADJUSTMENT INTERVAL)

```
    MONITOR = <string>RESULTS|RESTART|HISTORY
  END [HEARTBEAT OUTPUT <string>heartbeat_name]
```

The heartbeat output is text output file that gives:

- nodal variable results (displacements, forces, etc.) for specific nodes,
- edge variable results for specific edges,
- face variable results for specific faces,
- element results (stress, strain, etc.) for specific elements, and
- global results

at specified times.

Known Issue: User defined variables (see Section 11.2.4) are not currently supported with heartbeat output.

The output is written as text instead of the binary history output. You can specify a heartbeat file, the results to be included in this file, the formatting of the output, and the frequency at which results are written by using a `HEARTBEAT OUTPUT` command block. The command block appears inside the region scope. For heartbeat output, you will typically work with global, node, and element variables, and, on some occasions, face variables.

More than one heartbeat file can be specified for an analysis. For each heartbeat file, there will be one `HEARTBEAT OUTPUT` command block. The command block for a heartbeat file description begins with

```
BEGIN HEARTBEAT OUTPUT <string>heartbeat_name
```

and is terminated with

```
END [HEARTBEAT OUTPUT <string>heartbeat_name]
```

where `heartbeat_name` is a user-selected name for the command block. Nested within the `HEARTBEAT OUTPUT` command block are a set of command lines, as shown in the block summary given above. The first command line listed (`STREAM NAME`) gives pertinent information about the heartbeat file. The command line

```
STREAM NAME = <string>heartbeat_file_name
```

gives the name of the heartbeat file with the string `heartbeat_file_name`. If the file already exists, it is overwritten. If the heartbeat file is to appear in the current directory and is named `job.h`, this command line would appear as

```
STREAM NAME = job.h
```

If the heartbeat file is to be created in some other directory, the command line would have to show the absolute path to that directory.

In addition to specifying a specific filename, there are several predefined streams that can be specified. The predefined streams are:

- 'cout' or 'stdout' specifies standard output;
- 'cerr', 'stderr', 'clog', or 'log' specifies standard error;
- 'output' or 'outputP0' specifies Sierra's standard output which is redirected to the file specified by the '-o' option on the command line.

Two metacharacters can appear in the name of the heartbeat file. If the `%P` character is found in the name, it will be replaced with the number processors being used for the run. For example, if you are running on 1024 processors and use the name `heartbeat-%P/job.h`, then the name would be expanded to `heartbeat-1024/job.h`. The other recognized metacharacter is `%B` which is replaced with the base name of the input file containing the input commands. For example, if the commands are in the file `my_analysis_run.i` and the heartbeat stream name is specified as `%B.h`, then the heartbeat data would be written to the file `my_analysis_run.h`.

The other command lines that appear in the `HEARTBEAT OUTPUT` command block determine the type, frequency, and format of information that is output. Descriptions of these command lines follow in Section 9.4.1 through Section 9.4.14. Note that the command lines for controlling the frequency of heartbeat output (in Section 9.4.3 through Section 9.4.12) are the same as those for controlling the frequency of results and history output. These frequency-related command lines are repeated here for convenience.

9.4.1 Output Variables

The `GLOBAL`, `NODE` (or `NODAL`), `EDGE`, `FACE`, or `ELEMENT` command line is used to select variables for output in the heartbeat file. One of several types of variables can be selected for output. The form of the command line varies depending on the type of variable that is selected for output.

9.4.1.1 Global Output Variables

```
GLOBAL <string>variable_name
  [AS <string>heartbeat_variable_name]
```

This form of the command lets you select any global variable for output in the heartbeat file. The variable is selected with the string `variable_name`. The string `variable_name` is the name of the global variable and can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4). The `variable_name` can also specify `time`, `timestep`, or `step` to output the current simulation time, timestep, or execution step, respectively.

You can also specify a name, `heartbeat_variable_name`, for the selected entity following the `AS` keyword. For example, suppose you want to output the CPU time (`cpu_time`) as `cpu` in the heartbeat file. The command line to accomplish this would be:

```
GLOBAL cpu_time AS cpu
```

The specification of an alias is optional for a global variable.

9.4.1.2 Mesh Entity Output Variables

```
NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
  AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
  AS <string>heartbeat_variable_name
```

This form of the command lets you select any nodal, edge, face, or element variable for a specific mesh entity for output in the heartbeat file. For example, this command line will let you pick the displacement at a specific node and output the displacement to the heartbeat file using an alias that you have chosen.

For this form of the command, the mesh entity type is set to NODE (or NODAL), EDGE, FACE, or ELEMENT depending on the variable (set by variable_name) to be output. If the mesh entity type is set to NODE (or NODAL), EDGE, or FACE, the string variable_name can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4). If the mesh entity type is set to ELEMENT, the string variable_name can be a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

Selection of a specific mesh entity follows the AT keyword. You select a mesh entity type (NODE [or NODAL], EDGE, FACE, or ELEMENT) followed by the specific integer identifier, entity_id, for the mesh entity. You must specify a name, heartbeat_variable_name, for the selected entity following the AS keyword. For example, suppose you want to output the accelerations at node 88. The command line to obtain the accelerations at node 88 for the heartbeat file would be:

```
NODE ACCELERATION AT NODE 88 AS accel_88
```

where accel_88 is the name that will be used for this heartbeat variable in the heartbeat file.

Note that either the keyword NODE or NODAL can be used for nodal quantities.

As an example, the command line to obtain the von Mises stress for element 1024 for the heartbeat file would be:

```
ELEMENT VON_MISES AT ELEMENT 1024 AS vm_1024
```

where vm_1024 is the name that will be used for this heartbeat variable in the heartbeat file.

9.4.1.3 Nearest Point Output Variables

```
NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
  NEAREST LOCATION <real>global_x,
    real<global_y>, real<global_z>
  AS <string>heartbeat_variable_name
```

This form of the command lets you select any nodal, edge, face, or element variable for output in the heartbeat file using a nearest point criterion. The command line described in this subsection

is an alternative to the command line described in the preceding section, Section 9.4.1.2, for obtaining heartbeat output. The command line in this section or the command line in Section 9.4.1.2 produces heartbeat files with variable information. The difference in these two command lines (Section 9.4.1.3 and Section 9.4.1.2) is simply in how the variable information is selected.

For the above form of the command, the mesh entity type is set to `NODE` (or `NODAL`), `EDGE`, `FACE`, or `ELEMENT` depending on the variable (set by `variable_name`) to be output. If the mesh entity type is set to `NODE` (or `NODAL`), `EDGE`, or `FACE`, the string `variable_name` can be either a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4). If the mesh entity type is set to `ELEMENT`, the string `variable_name` can be a variable listed in Section 9.9 or a user-defined variable (see Section 9.2.2 and Section 11.2.4).

The specific mesh entity used for output is determined by global coordinates specified by the `NEAREST LOCATION` keyword and its associated input parameters—`global_x`, `global_y`, `global_z`. The specific mesh entity chosen for output is as follows:

- If the mesh entity has been set to `NODE` (or `NODAL`), the node in the mesh selected for output is the node whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters `global_x`, `global_y`, and `global_z`.
- If the mesh entity has been set to `EDGE`, the edge in the mesh selected for output is the edge with a center point (the average location of the two end points of the edge) whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters `global_x`, `global_y`, and `global_z`.
- If the mesh entity has been set to `FACE`, the face in the mesh selected for output is the face with a centroid whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters `global_x`, `global_y`, and `global_z`.
- If the mesh entity has been set to `ELEMENT`, the element in the mesh selected for output is the element with a centroid whose initial position is nearest the input global X, Y, and Z coordinates specified with the parameters `global_x`, `global_y`, and `global_z`.

Note that, in all the above cases, the original model coordinates are used when selecting the nearest entity, not the current coordinates.

You must specify a name, `heartbeat_variable_name`, for the selected entity following the `AS` keyword. As an example, suppose you want to output the accelerations at a node closest to the point with global coordinates (1012.0, 54.86, 103.3141). The command line to obtain the accelerations at the node closest to this location for the heartbeat file would be:

```
NODE ACCELERATION
     NEAREST LOCATION 1012.0, 54.86, 103.3141 AS accel_near
```

where `accel_near` is the name that will be used for this heartbeat variable in the heartbeat file.

Note that either the keyword `NODE` or `NODAL` can be used for nodal quantities.

9.4.2 Outputting Heartbeat Data on a Node Set

It is commonly desired to output heartbeat data on a single-node node set. If a mesh file is slightly modified, the node and element numbers will completely change. The node associated with a node set, however, remains the same, i.e., the node in the node set retains the same initial geometric location with the same connectivity to other elements even when its node number changes. Therefore, we might want to specify the heartbeat output for a node set with a single node rather than with the global identifier for a node. This can easily be accomplished, as follows:

```
 begin user output
   node set = nodelist_1
   compute global disp_ns_1 as average of nodal displacement
 end

begin heartbeat output
  global disp_ns_1
end
```

If `nodelist_1` contains only a single node, the heartbeat output variable `disp_ns_1` will contain the displacement for the single node in the node set. If `nodelist_1` contains multiple nodes, the average displacement of the nodes will be output.

9.4.3 Set Begin Time for Heartbeat Output

```
START TIME = <real>output_start_time
```

Using the `START TIME` command line, you can write heartbeat variables to the heartbeat file beginning at time `output_start_time`. No heartbeat variables will be written before this time. If other commands set times for heartbeat output (`AT TIME`, `ADDITIONAL TIMES`) that are less than `output_start_time`, those times will be ignored, and heartbeat output will not be written at those times.

9.4.4 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value `steps` in the `TIMESTEP ADJUSTMENT INTERVAL` command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, heartbeat variables are output at times closest to the specified output times.

9.4.5 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by `time_begin`, heartbeat variables will be output every time increment given by the real value `time_increment_dt`.

9.4.6 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 9.4.5, you can use the `ADDITIONAL TIMES` command line to specify an arbitrary number of additional output times.

9.4.7 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by `step_begin`, heartbeat variables will be output every step increment given by the integer value `step_increment`.

9.4.8 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 9.3.7, you can use the `ADDITIONAL STEPS` command line to specify an arbitrary number of g

9.4.9 Set End Time for Heartbeat Output

```
TERMINATION TIME = <real>termination_time_value
```

Heartbeat output will not be written to the heartbeat file after time `termination_time_value`. If other commands set times for heartbeat output (`AT TIME`, `ADDITIONAL TIMES`) that are greater than `termination_time_value`, those times will be ignored, and heartbeat output will not be written at those times.

9.4.10 Synchronize Output

```
SYNCHRONIZE OUTPUT
```

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of heartbeat data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the heartbeat output block. If a heartbeat block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A heartbeat block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

9.4.11 Use Output Scheduler

```
USE OUTPUT SCHEDULER <string>scheduler_name
```

In an analysis with multiple regions, it can be difficult to synchronize output such as heartbeat files. To help synchronize output for analyses with multiple regions, you can define an OUTPUT SCHEDULER command block at the SIERRA scope. The scheduler can then be referenced in the HEARTBEAT OUTPUT command block via the USE OUTPUT SCHEDULER command line. The string scheduler_name must match a name used in an OUTPUT SCHEDULER command block. See Section 9.6 for a description of using this command block and the USE OUTPUT SCHEDULER command line.

9.4.12 Write Heartbeat On Signal

```
OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
  SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
  SIGKILL|SIGILL|SIGSEGV
```

The OUTPUT ON SIGNAL command line is used to initiate the writing of a heartbeat file when the system encounters the specified signal. The signal can either occur as the result of a system error, or the user can explicitly send the specified signal to the application (See the system documentation man pages for “signal” or “kill” for more information). Only one signal type in the list of signal types should be entered for this command line. Generally, these signals cause the code to terminate before the code can add any current heartbeat output (heartbeat output past the last heartbeat output

time step) to the heartbeat file. If the code encounters the specified type of error during execution, a heartbeat file will be written before execution is terminated.

This command line can also be used to force the writing of a heartbeat file at some point during execution of the code. Suppose the command line

```
OUTPUT ON SIGNAL = SIGUSR2
```

is included in the input file. While the code is running, a user can execute (from the keyboard) the system command line

`kill -s SIGUSR2` *pid*

to force the writing of a results file. In the above system command line, *pid* is the process identifier, which is an integer.

Note that the `OUTPUT ON SIGNAL` command line is primarily a debugging tool for code developers.

9.4.13 Heartbeat Output Formatting Commands

There are several command lines for the heartbeat section that modify the formatting of the heartbeat text output. The default output for the heartbeat data consists of a line beginning with a timestamp showing the current wall-clock time followed by multiple columns of data, for example:

```
Begin HeartBeat Region_1_Heartbeat
  Stream Name = output
  At Step 0, Increment = 10

  precision is 5

  global step
  global timestep as dt
  global time
  global total_energy as te
End
```

```
+[12:18:51] step=240, dt=3.13933e-04, time=7.56578e-02, te=4.02795e-06
+[12:18:51] step=250, dt=3.13933e-04, time=7.87971e-02, te=1.32125e-06
+[12:18:51] step=260, dt=3.13933e-04, time=8.19365e-02, te=6.88142e-07
+[12:18:51] step=270, dt=3.13933e-04, time=8.50758e-02, te=3.93574e-06
+[12:18:52] step=280, dt=3.13933e-04, time=8.82151e-02, te=7.46609e-06
+[12:18:52] step=290, dt=3.13933e-04, time=9.13545e-02, te=1.03856e-05
+[12:18:52] step=300, dt=3.13933e-04, time=9.44938e-02, te=1.36822e-05
+[12:18:52] step=310, dt=3.13933e-04, time=9.76331e-02, te=1.64630e-05
```

The above example begins each line with a timestamp followed by five labeled data columns. The precision of the real data is 5. There is no legend in the above example. This format can be modified with the following commands.

9.4.13.1 CTH SpyHis output format

```
FORMAT = SPYHIS|DEFAULT
```

If the `FORMAT=SPYHIS` is specified, then the heartbeat output will be formatted such that it can be processed with the CTH SpyHis application which is a post-processor for time-history data.

9.4.13.2 Specify floating point precision

```
PRECISION = <integer>precision
```

By default, the real data is written with a precision of 5 which gives 5 digits following the decimal point. This can be altered with the `PRECISION` command. If the command line `PRECISION = 2` is specified, then the above data would look like:

```
Begin HeartBeat Region_1_Heartbeat
  ...
  precision = 2
  ...
End
```

```
+[12:18:51] step=240, dt=3.14e-04, time=7.57e-02, te=4.03e-06
+[12:18:51] step=250, dt=3.14e-04, time=7.88e-02, te=1.32e-06
+[12:18:51] step=260, dt=3.14e-04, time=8.19e-02, te=6.88e-07
```

Note that the precision applies to all real data; it is not possible to specify a different precision for each variable.

9.4.13.3 Specify Labeling of Heartbeat Data

```
LABELS = <string>OFF|ON
```

The above example shows the default output which consists of a label and the data separated by "=". The existence of the labels is controlled with the `LABELS` command. If `LABELS = OFF` is specified, then the above data would look like:

```
Begin HeartBeat Region_1_Heartbeat
  ...
  labels = off
  precision = 2
  ...
End
```

```
+[12:17:37] 240, 3.14e-04, 7.57e-02, 4.03e-06
+[12:17:37] 250, 3.14e-04, 7.88e-02, 1.32e-06
+[12:17:38] 260, 3.14e-04, 8.19e-02, 6.88e-07
```

9.4.13.4 Specify Existence of Legend for Heartbeat Data

```
LEGEND = <string>OFF|ON
```

Outputting the data without labels can make it easier to work with the data in a spreadsheet program or other data manipulation program, but with no labels, it is difficult to determine what the data really represents. The `LEGEND` output will print a line at the beginning of the heartbeat output identifying the data in each column. For example:

```
    Begin HeartBeat Region_1_Heartbeat
      ...
      legend = on
      labels = off
      precision = 2
      ...
    End

+[12:17:37] Legend: step, dt, time, te
+[12:17:37] 240, 3.14e-04, 7.57e-02, 4.03e-06
+[12:17:37] 250, 3.14e-04, 7.88e-02, 1.32e-06
+[12:17:38] 260, 3.14e-04, 8.19e-02, 6.88e-07
```

9.4.13.5 Specify format of timestamp

```
TIMESTAMP FORMAT <string>"timestamp_format"
```

Each line of the heartbeat output is preceded by a timestamp which shows the wall-clock time at the time that the line was output. This can be useful to verify that the code is still running and producing output and to determine how fast the code is running. The default timestamp is in the format "[12:34:56]" which is specified by the format "[%H:%M:%S]". If a different format is desired, it can be specified with the `TIMESTAMP FORMAT` command line. The format must be surrounded by double or single quotes and the format is defined to be the string between the first single or double quote and the last matching quote type. If you want to modify the format, see the documentation for the UNIX `strftime` command for details on how to specify the format. The example below shows a timestamp format delimited by "`{`" and "`}`". The timestamp consists of a ISO-8601 date format followed by the current time.

```
...
    timestamp format "{%F %H:%M:%S}"
...
+{2008-03-17 09:26:17} 2212, 1.34244e-06, 2.96948e-03, 2.96948e-03
+{2008-03-17 09:26:17} 2213, 1.34244e-06, 2.97082e-03, 2.97082e-03
+{2008-03-17 09:26:17} 2214, 1.34244e-06, 2.97216e-03, 2.97216e-03
+{2008-03-17 09:26:17} 2215, 1.34244e-06, 2.97350e-03, 2.97350e-03
+{2008-03-17 09:26:17} 2216, 1.34244e-06, 2.97485e-03, 2.97485e-03
```

9.4.14 Monitor Output Events

```
MONITOR = <string>RESULTS|RESTART|HISTORY
```

It is sometimes a benefit to know when the code has written a new set of data to one of the other output files (restart output, history output, or results output). The heartbeat output will report this data if the `MONITOR` command line is specified. Each time output is performed to any of the monitored output types, a line will be written to the heartbeat file specifying the timestamp, the simulation time and step, and the label name of the output type. For example:

```
begin results output my_results
  at step 0, increment = 10
...
end results output results

begin heartbeat data hb
  stream name = stdout
  monitor = results
  labels = off
  legend = on
  timestamp format "%F %H:%M:%S "
  at step 0, increment = 2
  global step
  global timestep as dt
  global time
  element spring_engineering_strain at \#
      element 1 as sp1
 end
```

Will give the following output:

```
....
+2008-03-17 10:03:22  718, 1.34244e-06, 9.63871e-04, 9.63871e-04
-2008-03-17 10:03:22  Results data written at time = 0.00096656,
step = 720. my_results
+2008-03-17 10:03:22  720, 1.34244e-06, 9.66556e-04, 9.66556e-04
+2008-03-17 10:03:22  722, 1.34244e-06, 9.69241e-04, 9.69241e-04
+2008-03-17 10:03:22  724, 1.34244e-06, 9.71926e-04, 9.71926e-04
+2008-03-17 10:03:22  726, 1.34244e-06, 9.74611e-04, 9.74611e-04
+2008-03-17 10:03:22  728, 1.34244e-06, 9.77296e-04, 9.77296e-04
-2008-03-17 10:03:22  Results data written at time = 0.00097998,
step = 730. my_results
+2008-03-17 10:03:22  730, 1.34244e-06, 9.79981e-04, 9.79981e-04
....
```

9.5 Restart Data

```
BEGIN RESTART DATA <string>restart_name
  DATABASE NAME = <string>restart_file
  INPUT DATABASE NAME = <string>restart_input_file
  OUTPUT DATABASE NAME = <string>restart_output_file
  DATABASE TYPE = <string>database_type(exodusII)
  OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
    (ON|TRUE|YES)
  START TIME = <real>restart_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  OVERLAY COUNT = <integer>overlay_count
  CYCLE COUNT = <integer>cycle_count
  SYNCHRONIZE_OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
  OPTIONAL
END [RESTART DATA <string>restart_name]
```

Explicit (applies to TIMESTEP ADJUSTMENT INTERVAL)

You can specify restart files, either to be written to or read from, and the frequency at which restarts are written by using a RESTART DATA command block. The command block appears inside the region scope. To initiate a restart, the RESTART TIME command line (see Section 2.1.3.1) or the RESTART command line (see Section 2.1.3.2) must also be used. These command lines appear in the SIERRA scope.

NOTE: In addition to the times at which you request restart information to be written, restart information is automatically written when an element inverts.

The RESTART DATA command block begins with the input line:

```
BEGIN RESTART DATA <string>restart_name
```

and is terminated with:

```
END [RESTART DATA <string>restart_name]
```

where restart_name is a user-selected name for the RESTART DATA command block.

Nested within the RESTART DATA command block are a set of command lines, as shown in the

block summary given above.

We begin the discussion of the RESTART DATA command block with various options regarding the use of restart in general. In Section 9.5.1, you will learn how to use the DATABASE NAME, INPUT DATABASE NAME, OUTPUT DATABASE NAME, DATABASE TYPE, and OPTIONAL command lines. Usage of the first three of these command lines is tied to the two restart-related command lines RESTART and RESTART TIME, which are found in the SIERRA scope.

Section 9.5.2 discusses use of the OVERWRITE command line, which will prevent or allow the overwriting of existing restart files. (Note that this command line also appears in the command blocks for results output and history output.)

The other command lines that appear in the RESTART DATA command block determine the frequency at which restarts are written. Descriptions of these command lines follow in Section 9.5.3 through Section 9.5.14. Note that the command lines for controlling the frequency of restart output are the same as those for controlling the frequency of results output and history output. These frequency-related command lines are repeated here for convenience.

9.5.1 Restart Options

```
DATABASE NAME = <string>restart_file
INPUT DATABASE NAME = <string>restart_input_file
OUTPUT DATABASE NAME = <string>restart_output_file
DATABASE TYPE = <string>database_type(exodusII)
OPTIONAL
```

You can read from and create restart files in an automated fashion, the preferred method, or you can carefully control how you read from and create restart files. In our discussion of the overall options for the use of restart, we begin with the first three command lines listed above (DATABASE NAME, INPUT DATABASE NAME, and OUTPUT DATABASE NAME). All three of these command lines specify a parameter that is a file name or a directory path and file name. If the parameter begins with the "/" character, it is an absolute path; otherwise, the path to the current directory will be prepended to the parameter on the command line. Suppose, for example, that we want to work with a restart file named component.rst in the current directory. If we are using the DATABASE NAME command line, then this command line would appear as:

```
DATABASE NAME = component.rst
```

To read or create files in some other directory, the command line must include the path to that directory. The directory must exist, it will not be created.

The DATABASE NAME command line will let you read restart information and write restart information to the same file. Section 9.5.1.1 through Section 9.5.1.4 show how this command line is used in particular instances.

You can specify a restart file to read from by using the command line:

```
INPUT DATABASE NAME = <string>restart_input_file
```

You can specify a restart file to write to by using the command line:

```
OUTPUT DATABASE NAME = <string>restart_output_file
```

Note that you must use either a DATABASE NAME command line or the INPUT DATABASE NAME command line/OUTPUT DATABASE NAME command line pair, but not both, in a RESTART DATA command block.

Two metacharacters can appear in the name of the restart file. If the %P character is found in the name, it will be replaced with the number processors being used for the run. For example, if you are running on 1024 processors and use the name restart-%P/job.rs, then the name would be expanded to restart-1024/job.rs and the actual restart files would be restart-1024/job.rs.1024.0000 to restart-1024/job.rs.1024.1023. The other recognized metacharacter is %B which is replaced with the base name of the input file containing the input commands. For example, if the commands are in the file my_analysis_run.i and the restart database name is specified as %B.rs, then the restart data would be written to or read from the file my_analysis_run.rs.

If the restart file does not use the Exodus II format [1], you must specify the format for the results file using the DATABASE TYPE command line:

```
DATABASE TYPE = <string>database_type(exodusII)
```

Currently, the Exodus II database and the XDMF database [2] are supported in Presto and Adagio. Exodus II is more commonly used than XDMF. Other options may be added in the future.

In certain coupled physics analyses in which there are multiple regions, only a subset of the regions may have a restart database associated with them. The OPTIONAL command (Section 9.5.1) is used to tell the application that it is acceptable to restart the analysis even though a region does not have an associated restart database. Note that this is only allowed in analyses containing multiple regions; if there is only a single region, it must have a restart database in order to restart.

9.5.1.1 Automatic Read and Write of Restart Files

You can use the restart option in an automated fashion by using a combination of the RESTART command line in the SIERRA scope and the DATABASE NAME command line in the RESTART DATA command block. This automated use of restart can best be explained by an example. We will use a two-processor example and assume all files will be in our current directory.

The option of automated restart will not only manage the restart files to prevent overwriting, it will also manage the results files and history files to prevent overwriting. In the example we give, we will assume our run includes a RESULTS OUTPUT command block with the command line

```
DATABASE NAME = rslt.e
```

to generate results files with the root file name rslt.e. We will also assume a run includes a HISTORY OUTPUT command block with the command line

```
DATABASE NAME = hist.h
```

to generate history files with the root file name hist.h.

For the first run in our restart sequence, we will have the command line

```
RESTART = AUTOMATIC
```

in the SIERRA scope of our input file. In a TIME STEPPING command block, which is embedded in a TIME CONTROL command block (Section 3.1.1) in the procedure scope of our input file, we will have the command line:

```
START TIME = 0.0
```

In the TIME CONTROL command block we will have the command line

```
TERMINATION TIME = 2.5E-3
```

to set the limits for the begin and end times of the first restart run. These time-related command lines should not be confused with the START TIME and TERMINATION TIME command lines that appear in the RESTART DATA command block.

Finally, for the first run in our restart sequence, the RESTART DATA command block in our input file will be as follows:

```
BEGIN RESTART DATA RESTART_DATA
  DATABASE NAME = g.rsout
  AT TIME 0.0 INCREMENT = 0.25E-3
END RESTART DATA RESTART_DATA
```

In this block, the DATABASE NAME command line specifies a root file name for the restart file. The AT TIME command line gives the time when we will start to write the restart information and the interval at which the restart information will be written (see Section 9.5.5).

For our first run, the automatic restart option will generate the following restart files:

```
# restart files
g.rsout.2.0
g.rsout.2.1
# results files
rslt.e.2.0
rslt.e.2.1
# history files
hist.h.2.0
hist.h.2.1
```

For the above files, there are extensions on the file names that indicate we have a two-processor run. The 2.0 and 2.1 extensions associate the restart files with the corresponding individual mesh files on each processor. (If our mesh file is mesh.g, then our mesh files on the individual processors will be mesh.g.2.0 and mesh.g.2.1.) All restart information in the above files appears at time intervals of 0.25×10^{-3}, and the last restart information is written at time 2.5×10^{-3}. We have also listed the results and history files that will be generated for this run due to the file definitions in the command blocks for the results and history files.

For the second run in our sequence of restart runs, we want to start at the previous termination time, 2.5×10^{-3}, and terminate at time 5.0×10^{-3}. We leave everything in our input file (including the

START TIME = 0.0 command line in the TIME STEPPING command block, the RESTART command line, and the RESTART DATA command block) the same except for the TERMINATION TIME command line (in the TIME CONTROL command block). The TERMINATION TIME command line will now become:

```
TERMINATION TIME = 5.0E-3
```

It is important to note here that the actual start time for the second run in our analysis is now set by the last time (2.5×10^{-3}) that restart information was written. The command line START TIME = 0.0 in the TIME STEPPING command block is now superseded as the actual starting time for the second run by the restart commands. Any START TIME command line in a TIME STEPPING command block is still valid in terms of defining time stepping blocks (these blocks being used to set activation periods), but the restart process sets the actual start time for our analysis. This pattern of control for setting the actual start time holds for any run in our sequence of restart runs.

For the second run in our sequence of restart runs, the restart files will be from time 2.5×10^{-3} to time 5.0×10^{-3}. The restart files in our current directory after the second run will be as follows:

```
# restart files
g.rsout.2.0
g.rsout.2.1
g.rsout-s0002.2.0
g.rsout-s0002.2.1
# results files
rslt.e.2.0
rslt.e.2.1
rslt.e-s0002.2.0
rslt.e-s0002.2.1
# history files
hist.h.2.0
hist.h.2.1
hist.h-s0002.2.0
hist.h-s0002.2.1
```

Notice that we have generated new restart files with a -s0002 extension in addition to the extension associated with the individual processors. All restart information in the above files with the -s0002 extension appears at time intervals of 0.25×10^{-3}, the restart information is written between time 2.5×10^{-3} and time 5.0×10^{-3}, and the final restart information is written at time 5.0×10^{-3}. The restart files for the first run in our sequence of restart runs, g.rsout.2.0 and g.rsout.2.1, have been preserved. New results and history files have been created using the same extension, -s0002, as that used for the restart files. The original results and history files have been preserved.

Now, we want to do a third run in our sequence of restart runs. For the third run in our sequence of restart runs, we want to start at the previous termination time, 5.0×10^{-3}, and terminate at time 8.5×10^{-3}. We leave everything in our input file (including the START TIME command line, the RESTART command line, and the RESTART DATA command block) the same except for

the TERMINATION TIME command line. The TERMINATION TIME command line (within the TIME CONTROL command block) will now become:

```
TERMINATION TIME = 8.5E-3
```

For the third run in our sequence of restart runs, the restart files will be from time 5.0×10^{-3} to time 8.5×10^{-3}. The restart files in our current directory after the third run will be as follows:

```
# restart files
g.rsout.2.0
g.rsout.2.1
g.rsout-s0002.2.0
g.rsout-s0002.2.1
g.rsout-s0003.2.0
g.rsout-s0003.2.1
# results files
rslt.e.2.0
rslt.e.2.1
rslt.e-s0002.2.0
rslt.e-s0002.2.1
rslt.e-s0003.2.0
rslt.e-s0003.2.1
# history files
hist.h.2.0
hist.h.2.1
hist.h-s0002.2.0
hist.h-s0002.2.1
hist.h-s0003.2.0
hist.h-s0003.2.1
```

Notice that we have generated new restart files with a -s0003 extension in addition to the extension associated with the individual processors. All restart information in the above files with the -s0003 extension appears at time intervals of 0.25×10^{-3}, the restart information is written between time 5.0×10^{-3} and time 8.5×10^{-3}, and the final restart information is written at time 8.5×10^{-3}. The restart files for the first and second runs in our sequence of restart runs have been preserved. New results and history files have been created using the same extension, -s0003, as that used for the restart files. The original results and history files have been preserved.

The process just described can be continued as long as necessary. We will continue the process of generating new restart files with extensions that indicate their place in the sequence of runs.

9.5.1.2 User-Controlled Read and Write of Restart Files

You can use the restart option and select specific restart times and specific restart files to read from and write to by using a combination of the RESTART TIME command line in the SIERRA scope and the INPUT DATABASE NAME and OUTPUT DATABASE NAME command line in the RESTART DATA command block. This “controlled” use of restart can best be explained by an example.

We will use a two-processor example and assume all files will be in our current directory. In this example, we will manage the creation of new restart files so as not to overwrite existing restart files. Unlike the automated option for restart, this controlled use of restart requires that the user manage restart file names so as to prevent overwriting previously generated restart files. The same is true for the results and history files. The user will have to manage the creation of new results and history files so as not to overwrite existing results and history files. Creating new results and history files for each run in the sequence of restart runs requires changing the `DATABASE NAME` command line in the `RESULTS OUTPUT` and `HISTORY OUTPUT` command blocks. We will not show examples for use of the `DATABASE NAME` command line in the `RESULTS OUTPUT` and `HISTORY OUTPUT` command blocks here, as the actual use of the `DATABASE NAME` command line in the results and history command blocks would closely parallel the pattern we see for management of the restart file names.

For the first run in our restart sequence, we will have only a `RESTART DATA` command block in the region; there will be no restart-related command line in the SIERRA scope of our input file. We will, however, have a

```
START TIME = 0.0
```

command line in a `TIME STEPPING` command block (within the `TIME CONTROL` command block) and a

```
TERMINATION TIME = 2.5E-3
```

command line within the `TIME CONTROL` command block to set the limits for the begin and end times. The `RESTART DATA` command block in our input file will be as follows:

```
BEGIN RESTART DATA RESTART_DATA
  OUTPUT DATABASE NAME = RS1.rsout
  AT TIME 0.0 INCREMENT = 0.5E-3
END RESTART DATA RESTART_DATA
```

For our first run, the restart option will generate the following restart files:

```
RS1.rsout.2.0
RS1.rsout.2.1
```

For the above files, the extensions on the file names indicate that we have a two-processor run. The 2.0 and 2.1 extensions associate the restart files with the corresponding individual mesh files on each processor. If our mesh file is `mesh.g`, then our mesh files on the individual processors will be `mesh.g.2.0` and `mesh.g.2.1`. All restart information in the above files appears at time intervals of 0.5×10^{-3}, and the last restart information is written at time 2.5×10^{-3}.

For the second run in our sequence of restart runs, we want to start at the previous termination time, 2.5×10^{-3}, and terminate at time 5.0×10^{-3}. To do this, we must add a

```
RESTART TIME = 2.5E-3
```

command line to the SIERRA scope and set the termination time to 5.0×10^{-3} by using the command line

```
TERMINATION TIME = 5.0E-3 \rm
```

within the TIME CONTROL command block.

It is important to note here that the actual start time for the second run in our analysis is now set by the restart time set on the RESTART TIME command line, 2.5×10^{-3}. The command line START TIME = 0.0 in the TIME STEPPING command block is now superseded as the actual starting time for the second run by the restart commands. Any START TIME command line in a TIME STEPPING command block is still valid in terms of defining time stepping blocks (these blocks being used to set activation periods), but the restart process sets the actual start time for our analysis. This pattern of control for setting the actual start time holds for any run in our sequence of restart runs.

We also must change the RESTART DATA command block to the following:

```
BEGIN RESTART DATA RESTART_DATA
  INPUT DATABASE NAME = RS1.rsout
  OUTPUT DATABASE NAME = RS2.rsout
  AT TIME 0.0 INCREMENT = 0.5E-3
END RESTART DATA RESTART_DATA
```

For this second run, we will read from the following files:

```
RS1.rsout.2.0
RS1.rsout.2.1
```

And we will write to the following files:

```
RS2.rsout.2.0
RS2.rsout.2.1
```

All restart information in the above output files, RS2.rsout.2.0 and RS2.rsout.2.1, appears at time intervals of 0.5×10^{-3}, restart information is written from time 2.5×10^{-3} to time 5.0×10^{-3}, and the last restart information is written at time 5.0×10^{-3}. Notice that we have preserved the restart files from the first run from our restart sequence of runs because we have specifically given the input and output databases distinct names—RS2.rsout for the input file name and RS1.rsout for the output file name.

Now, we want to do a third run in our sequence of restart runs. For this third run, we want to start at time 4.5×10^{-3} and terminate at time 8.5×10^{-3}. We do not want to start at the termination time for the previous restart, which is 5.0×10^{-3}; rather, we want to start at time 4.5×10^{-3}. We change the RESTART TIME command line to

```
RESTART TIME = 4.5E-3
```

and the TERMINATION TIME command line within the TIME CONTROL command block to:

```
TERMINATION TIME = 8.5E-3
```

And we change the RESTART DATA command block to the following:

```
BEGIN RESTART DATA RESTART_DATA
  INPUT DATABASE NAME = RS2.rsout
  OUTPUT DATABASE NAME = RS3.rsout
  AT TIME 0.0, INCREMENT = 0.5E-3
END RESTART DATA RESTART_DATA
```

For this third run, we will read from the following files:

```
RS2.rsout.2.0
RS2.rsout.2.1
```

And we will write to the following files:

```
RS3.rsout.2.0
RS3.rsout.2.1
```

All restart information in the above output files, `RS3.rsout.2.0` and `RS3.rsout.2.1`, appears at time intervals of 0.5×10^{-3}, restart information is written from time 4.5×10^{-3} to time 8.5×10^{-3}, and the last restart information is written at time 8.5×10^{-3}. Notice that we have preserved all restart files from previous runs in our restart sequence of runs because we have specifically given the input and output databases distinct names for this third run.

9.5.1.3 Overwriting Restart Files

If you use the `RESTART TIME` command line in conjunction with the `DATABASE NAME` command line, you will overwrite restart information (unless you have included an `OVERWRITE` command line set to `ON`). As indicated previously, you will probably want to have a restart file (or files in the case of parallel runs) associated with each run in a sequence of restart runs. The example in this section shows how to overwrite restart files if that is an acceptable approach for a particular analysis.

For our first run, we will set a termination time of 1.0×10^{-3} with the command line

```
TERMINATION TIME = 1.0E-3
```

and set the `RESTART DATA` command block as follows:

```
BEGIN RESTART DATA
  DATABASE NAME = RS.out
  AT TIME 0.0 INTERVAL = 0.25E-3
END RESTART DATA
```

Our first run will generate the following restart files:

```
RS.out.2.0
RS.out.2.1
```

All restart information in the above output files, `RS.out.2.0` and `RS.out.2.1`, appears at time intervals of 0.25×10^{-3}, restart information is written from time 0.0 to time 1.0×10^{-3}, and the last restart information is written at time 1.0×10^{-3}.

Suppose for our second run we set the termination time to 2.0×10^{-3} with the command line

```
TERMINATION TIME = 2.0E-3
```

and add the command line

```
RESTART TIME = 1.0E-3
```

to the SIERRA scope. We leave the `RESTART DATA` command block unchanged.

For our second run, restart information is read from the files `RS.out.2.0` and `RS.out.2.1`. These files are then overwritten with new restart information beginning at time 1.0×10^{-3}. The files `RS.out.2.0` and `RS.out.2.1` will have restart information beginning at time 1.0×10^{-3} in intervals of 0.25×10^{-3}. The restart information will terminate at time 2.0×10^{-3}.

Now we want to do a third run with a termination time of 3.0×10^{-3}. We change the termination time by using the command line:

```
TERMINATION TIME = 3.0E-3
```

And we change the `RESTART TIME` command line so that it is now:

```
RESTART TIME = 3.0E-3
```

For our third run, restart information is read from the files `RS.out.2.0` and `RS.out.2.1`. These files are then overwritten with new restart information beginning at time 2.0×10^{-3}. The files `RS.out.2.0` and `RS.out.2.1` will have restart information beginning at time 2.0×10^{-3} in intervals of 0.25×10^{-3}. The restart information will terminate at time 3.0×10^{-3}.

9.5.1.4 Recovering from a Corrupted Restart

Suppose you are using the automated option for restart and a system crash occurs when the restart file is being written. The restart file contains a corrupted entry for one of the restart times. In this case, you can continue using the automated option for restart. Restart will detect the corrupted entry and then find an entry previous to the corrupted entry that can be used for restart. This previous entry should be the entry prior to the corrupted entry unless something unusual has occurred. If the first intact restart entry is not the previous entry, restart continues to back up until an intact restart entry is found.

You could do a manual recovery. The manual recovery requires the use of a `RESTART TIME` command line to select some intact restart entry. You will have to use the `INPUT DATABASE NAME` and `OUTPUT DATABASE NAME` command lines to avoid overwriting previous restart files (see Section 9.5.1.2). You will also have to change file names in the results and history command blocks to avoid overwriting previous results and history files. Once you have done the manual recovery, you could then revert to the automatic restart option.

9.5.2 Overwrite Command in Restart

```
OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
  (ON|TRUE|YES)
```

The OVERWRITE command line can be used to prevent the overwriting of existing restart files. The use of the automatic read and write option for restart files as described in Section 9.5.1.1 does not require the OVERWRITE command line. The automatic read and write option adds extensions to file names and prevents the overwriting of any existing restart files. For the case of a user-controlled read and write of restart files (Section 9.5.1.2), however, the OVERWRITE command line is useful for preventing the overwriting of restart files. If the OVERWRITE command line is set to OFF, FALSE, or NO, then existing restart files will not be overwritten. Execution of the code will terminate before existing restart files are overwritten. The default option is to overwrite existing restart files. If the OVERWRITE command line is not included, or the command line is set to ON, TRUE, or YES, then existing files can be overwritten.

9.5.3 Set Begin Time for Restart Writes

```
START TIME = <real>restart_start_time
```

Using the START TIME command line, you can write restarts to the restart file beginning at time restart_start_time. No restarts will be written before this time. If other commands set times for restarts (AT TIME, ADDITIONAL TIMES) that are less than restart_start_time, those times will be ignored, and restarts will not be written at those times.

9.5.4 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that the restarts will be written at exactly the times specified. To hit the restart times exactly in an explicit transient dynamics code, it is necessary to adjust the time step as the time approaches a restart time. The integer value steps in the TIMESTEP ADJUSTMENT INTERVAL command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, then restarts are written at times closest to the specified restart times.

9.5.5 Restart Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by time_begin, restarts will be written every time increment given by the real value time_increment_dt.

9.5.6 Additional Times for Restart

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any restart times specified by the command line in Section 9.5.5, you can use the ADDITIONAL TIMES command line to specify an arbitrary number of additional restart times.

9.5.7 Restart Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by step_begin, restarts will be written every step increment given by the integer value step_increment.

9.5.8 Additional Steps for Restart

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 9.5.7, you can use the ADDITIONAL STEPS command line to specify an arbitrary number of additional restart steps.

9.5.9 Set End Time for Restart Writes

```
TERMINATION TIME = <real>termination_time_value
```

Restarts will not be written to the restart file after time termination_time_value. If other commands set times for restarts (AT TIME, ADDITIONAL TIMES) that are greater than termination_time_value, those times will be ignored, and restarts will not be written at those times.

9.5.10 Overlay Count

```
OVERLAY COUNT = <integer>overlay_count
```

The OVERLAY COUNT command line specifies the number of restart output times that will be overlaid on top of the current step before advancing to the next step. For example, suppose that we set the overlay_count parameter to 2, and we request that restart information be written every 0.1 second. At time 0.1 second, restart step 1 will be written to the output restart database. At time 0.2 second, restart information will be written over the step 1 information, which originally contained

restart information at 0.1 second. At time 0.3 second, restart information will be written over the step 1 information, which last contained information at 0.2 second. At time 0.4 second, we will now write step 2 to the output restart database (step 1 has already been written over twice). At time 0.5 second, restart information will be written over the step 2 information, which originally contained information at 0.4 second. At time 0.6 second, restart information will be written over the step 2 information, which last contained information at 0.5 second. At time 0.7 second, restart step 3 will be written to the output restart database (step 2 has already been written over twice). This pattern continues so that we would build up a sequence of restart information at times 0.3, 0.6, 0.9, . . . second until we reach the termination time for the problem. If there was a problem during the analysis, the last step on the output restart database would be whatever had last been written to the database. If, for example, we had set our termination time to 1.0 second and a problem occurred after restart information had been written at 0.7 second but before we completed the time step at 0.8 second, then the last information on the output restart database would be at 0.7 second.

You can use the `OVERLAY COUNT` command line in conjunction with a `CYCLE COUNT` command line. For a description of the `CYCLE COUNT` command line and its use with the `OVERLAY COUNT` command line, see Section 9.5.11.

9.5.11 Cycle Count

```
CYCLE COUNT = <integer>cycle_count
```

The `CYCLE COUNT` command line specifies the number of restart steps that will be written to the output restart database before previously written steps are overwritten. For example, suppose we set the `cycle_count` parameter to 5, and we request that restart information be written every 0.1 second. The restart system will write information to the output restart database at times 0.1, 0.2, 0.3, 0.4, and 0.5 second. At time 0.6 second, the information at step 1, originally written at time 0.1 second, will be overwritten with information at time 0.6 second. At time 0.7 second, the information at step 2, originally written at time 0.2 second, will be overwritten with information at time 0.7 second. At time 0.8 second, the output restart database will contain restart information at times 0.6, 0.7, 0.8, 0.4, and 0.5 second. Time will not necessarily be monotonically increasing on a database that uses a `CYCLE COUNT` command line.

If you only want the last step available on the output restart database, set `cycle_count` equal to 1.

The `CYCLE COUNT` and `OVERLAY COUNT` command lines can be used at the same time. For this example, we will combine our example with an overlay count of 2 as given in Section 9.5.10 with our example of a cycle count of 5 as given in this section (Section 9.5.11). Information is written to the output restart database time step every 0.1 second. The output times at which information is written to the output restart database are 0.1, 0.2, 0.3, . . . second. Each of these times corresponds to an output step. Time 0.1 second corresponds to output step 1, time 0.2 second corresponds to output step 2, time 0.3 corresponds to output step 3, and so forth. An output time of $n \times 0.1$ corresponds to output step n. The overlay command will result in information at time 0.3, 0.6, 0.9, 1.2, and 1.5 seconds written as steps 1, 2, 3, 4, and 5 on the output restart database. For times greater than 1.6 seconds, the cycle command will now take effect because we have five steps written

on the output restart database. Information at times 1.6, 1.7, and 1.8 seconds will now overwrite the information at step 1, which had information at time 0.3 second. Information at times 1.9, 2.0, and 2.1 seconds will now overwrite the information at step 2, which had information at time 0.6 second. For any output step n, its position, step number n_s, in the restart output database is as follows:

$$\begin{aligned} & if\, n_s \neq 0 \\ & \quad n_s = int(n/(n_o + 1))\%n_c \\ & else \\ & \quad n_s = n_c \\ & end \end{aligned}$$

In the above equations, n_c is the cycle count, and n_o is the overlay count. The expression $int(n/(n_o + 1))$ produces an integer arithmetic result. For example, if n is 4 and n_o is 2, then we have 4 divided by 3, and the integer arithmetic result is 1 (any fractional remainder is discarded). The operator % is the modulus operator; the modulus operator gives the modulus of its first operand with respect to its second operand, i.e., it produces the remainder of dividing the first operand by the second operand. The result of 1 % 5 is 1, for example.

9.5.12 Synchronize Output

```
SYNCHRONIZE OUTPUT
```

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of restart data between the regions. This can be done by adding the `SYNCHRONIZE OUTPUT` command line to the restart output block. If a restart block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the `USE OUTPUT SCHEDULER` command line can also synchronize output between regions, the `SYNCHRONIZE OUTPUT` will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A restart block with `SYNCHRONIZE OUTPUT` specified will also synchronize its output with the output of the external code.

The `SYNCHRONIZE OUTPUT` command can be used with other output scheduling commands such as time-based or step-based output specifications.

9.5.13 Use Output Scheduler

```
USE OUTPUT SCHEDULER <string>scheduler_name
```

In an analysis with multiple regions, it can be difficult to synchronize output such as restart files. To help synchronize output for analyses with multiple regions, you can define an OUTPUT SCHEDULER command block at the SIERRA scope. The scheduler can then be referenced in the RESTART DATA command block via the USE OUTPUT SCHEDULER command line. The string scheduler_name must match a name used in an RESTART DATA command block. See Section 9.6 for a description of using this command block and the USE OUTPUT SCHEDULER command line.

9.5.14 Write Restart If System Error Encountered

```
OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
  SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
  SIGKILL|SIGILL|SIGSEGV
```

The OUTPUT ON SIGNAL command line is used to initiate the writing of a restart file when the system encounters a type of system error. Only one error type in the list of error types should be entered for this command line. Generally, these system errors cause the code to terminate before the code can add any current restart output (restart output past the last restart output time step) to the restart file. If the code encounters the specified type of error during execution, a restart file will be written before execution is terminated.

This command line can also be used to force the writing of a restart file at some point during execution of the code. Suppose the command line

```
OUTPUT ON SIGNAL = SIGUSR2
```

is included in the input file. While the code is running, a user can execute (from the keyboard) the system command line

kill -s SIGUSR2 *pid*

to terminate execution and force the writing of a results file. In the above system command line, *pid* is the process identifier, which is an integer.

The most useful application of the command line is to send a signal via a system command line to write a restart file. Note that the OUTPUT ON SIGNAL command line is primarily a debugging tool for code developers.

9.6 Output Scheduler

In an analysis with multiple regions, it can be difficult to synchronize output such as results files, history files, and restart files. To help synchronize output for analyses with multiple regions, you can define an OUTPUT SCHEDULER command block at the SIERRA scope. This scheduler can then be referenced in several places:

- The scheduler can be referenced in the RESULTS OUTPUT command block to control the output of results information.
- The scheduler can be referenced in the HISTORY OUTPUT command block to control the output of history information.
- The scheduler can be referenced in the RESTART DATA command block to control the writing of restart files.

In summary, the OUTPUT SCHEDULER command block is defined in the SIERRA scope. The scheduler is referenced by a USE OUTPUT SCHEDULER command line that can appear in a RESULTS OUTPUT, HISTORY OUTPUT, and RESTART DATA command block. Section 9.6.1 describes the OUTPUT SCHEDULER command block, and Section 9.6.2 illustrates how this block is referenced with the USE OUTPUT SCHEDULER command line.

9.6.1 Output Scheduler Command Block

Explicit

```
BEGIN OUTPUT SCHEDULER <string>scheduler_name
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [OUTPUT SCHEDULER <string>scheduler_name]
```

An output scheduler is defined with a command block in the SIERRA scope. The OUTPUT SCHEDULER command block begins with the input line:

```
BEGIN OUTPUT SCHEDULER <string>scheduler_name
```

and is terminated with the line:

```
END OUTPUT SCHEDULER <string>scheduler_name
```

where scheduler_name is a user-defined name for the command block. All the normal scheduling command lines are valid in an OUTPUT SCHEDULER command block.

9.6.1.1 Set Begin Time for Output Scheduler

```
START TIME = <real>output_start_time
```

Using the START TIME command line, you can set the start time for a scheduler beginning at time output_start_time. The scheduler will not take effect before this time. If other commands set times for scheduling (AT TIME, ADDITIONAL TIMES) that are less than output_start_time, those times will be ignored.

9.6.1.2 Adjust Interval for Time Steps

```
TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
```

This command line is used to specify that, when the scheduler is in effect, output will be at exactly the times specified. To hit the output times exactly in an explicit, transient dynamics code, it is necessary to adjust the time step as the time approaches an output time. The integer value steps in the TIMESTEP ADJUSTMENT INTERVAL command line specifies the number of time steps to look ahead in order to adjust the time step.

If this command line does not appear, output occurs at times closest to the specified output times.

9.6.1.3 Output Interval Specified by Time Increment

```
AT TIME <real>time_begin INCREMENT = <real>time_increment_dt
```

At the time specified by time_begin, output will be scheduled at every time increment given by the real value time_increment_dt.

9.6.1.4 Additional Times for Output

```
ADDITIONAL TIMES = <real>output_time1 <real>output_time2 ...
```

In addition to any times specified by the command line in Section 9.6.1.3, you can use the ADDITIONAL TIMES command line to specify an arbitrary number of additional output times.

9.6.1.5 Output Interval Specified by Step Increment

```
AT STEP <integer>step_begin INCREMENT =
  <integer>step_increment
```

At the step specified by step_begin, output will be scheduled at every step increment given by the integer value step_increment.

9.6.1.6 Additional Steps for Output

```
ADDITIONAL STEPS = <integer>output_step1
  <integer>output_step2 ...
```

In addition to any steps specified by the command line in Section 9.6.1.5, you can use the ADDITIONAL STEPS command line to specify an arbitrary number of additional output steps.

9.6.1.7 Set End Time for Output Scheduler

```
TERMINATION TIME = <real>termination_time_value
```

Using the TERMINATION TIME command line, you can set the termination time for a scheduler beginning at time termination_time_value. The scheduler will not be in effect after this time. If other commands set times for scheduling (AT TIME, ADDITIONAL TIMES) that are greater than termination_time_value, those times will be ignored by the scheduler.

9.6.2 Example of Using the Output Scheduler

Once an output scheduler has been defined via the OUTPUT SCHEDULER command block, it can be used by inserting a USE OUTPUT SCHEDULER command line in any of the following command blocks: RESULTS OUTPUT, HISTORY OUTPUT, and RESTART DATA. The following paragraph provides an example of using output schedulers.

In the SIERRA scope, we define two output schedulers, Timer and Every_Step:

```
BEGIN OUTPUT SCHEDULER Timer
  AT TIME 0.0 INCREMENT = 10.0e-6
  TIME STEP ADJUSTMENT INTERVAL = 4
END OUTPUT SCHEDULER Timer
#
BEGIN OUTPUT SCHEDULER Every_Step
  AT STEP 0 INCREMENT = 1
END OUTPUT SCHEDULER Every_Step
```

With the USE OUTPUT SCHEDULER command, we reference the scheduler named Timer for results output:

```
BEGIN RESULTS OUTPUT Out_Region_1
  .
  USE OUTPUT SCHEDULER Timer
  .
END RESULTS OUTPUT Out_Region_1
```

With the USE OUTPUT SCHEDULER command, we reference the scheduler named Every_STEP for history output:

```
BEGIN HISTORY OUTPUT Out_Region_2
  .
  USE OUTPUT SCHEDULER Every_Step
  .
END HISTORY OUTPUT Out_Region_2
```

9.7 Variable Interpolation

```
BEGIN VARIABLE INTERPOLATION [<string>var_interp_name]
  TRANSFER TYPE = INTERPOLATE FROM NEAREST FACE|
    SUM TO NEAREST ELEMENT
  SOURCE VARIABLE = ELEMENT|NODAL|
    SURFACE NORMAL NODAL <string>source_var_name
  SOURCE ELEMENT BLOCK = <string>source_block
  SOURCE SURFACE = <string>source_surface
  TARGET VARIABLE = ELEMENT|FACE|NODAL
    <string>target_var_name
  TARGET DERIVATIVE VARIABLE = ELEMENT|FACE|NODAL
    <string>target_deriv_var_name
  TARGET SURFACE = <string>target_surface
  SEARCH TOLERANCE = <real>search_tol
  PROXIMITY SEARCH TYPE = RAY SEARCH|SPHERE SEARCH
  RAY SEARCH DIRECTION = <real>vecx <real>vecy <real>vecz
  RAY SEARCH DIRECTION = ORTHOGONAL TO LINE <real>p1x
    <real>p1y <real>p1z <real>p2x <real>p2y <real>p2z
  RAY SEARCH DIRECTION = TARGET SURFACE NORMAL

  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [VARIABLE INTERPOLATION <string>var_interp_name]
```

The command block for a variable interpolation begins with `BEGIN VARIABLE INTERPOLATION [<string>var_interp_name]` and is terminated with `END [VARIABLE INTERPOLATION <string>var_interp_name]` where `var_interp_name` is an optional user-selected name for the command block.

The command `TRANSFER TYPE` can be set to either `INTERPOLATE FROM NEAREST FACE` or `SUM TO NEAREST ELEMENT`. The `TRANSFER TYPE` restricts the commands that can be used in conjunction with it.

When `TRANSFER TYPE` is set to `INTERPOLATE FROM NEAREST FACE` the available commands are restricted to:

```
BEGIN VARIABLE INTERPOLATION [<string>var_interp_name]
  TRANSFER TYPE = INTERPOLATE FROM NEAREST FACE
  SOURCE VARIABLE = NODAL|SURFACE NORMAL NODAL
    <string>source_var_name
  SOURCE SURFACE = <string>source_surface
  TARGET VARIABLE = NODAL <string>target_var_name
  TARGET DERIVATIVE VARIABLE = NODAL <string>target_deriv_var_name
  TARGET SURFACE = <string>target_surface
  SEARCH TOLERANCE = <real>search_tol
  PROXIMITY SEARCH TYPE = RAY SEARCH|SPHERE SEARCH
```

```
  RAY SEARCH DIRECTION = <real>vecx <real>vecy <real>vecz
  RAY SEARCH DIRECTION = ORTHOGONAL TO LINE <real>p1x
    <real>p1y <real>p1z <real>p2x <real>p2y <real>p2z
  RAY SEARCH DIRECTION = TARGET SURFACE NORMAL
END [VARIABLE INTERPOLATION <string>var_interp_name]
```

For TRANSFER TYPE = INTERPOLATE FROM NEAREST FACE, a given point on the target surface finds the closest point on the source surface. At that closet point the source variable is interpolated. The interpolated value is then copied to the given point. Optionally if the SURFACE NORMAL NODAL component is being used and the variable being transferred is a vector then only the surface normal component of that variable will be transferred. This is useful for several applications involving transferring solid mesh quantities such as velocity to a reference mesh modeling a fluid (e.g., air).

When TRANSFER TYPE is set to SUM TO NEAREST ELEMENT the available commands are restricted to:

```
BEGIN VARIABLE INTERPOLATION [<string>var_interp_name]
  TRANSFER TYPE = SUM TO NEAREST ELEMENT
  SOURCE VARIABLE = ELEMENT <string>source_var_name
  SOURCE ELEMENT BLOCK = <string>source_block
  TARGET VARIABLE = ELEMENT <string>target_var_name
  TARGET DERIVATIVE VARIABLE = ELEMENT
    <string>target_deriv_var_name
  TARGET SURFACE = <string>target_surface
  SEARCH TOLERANCE = <real>search_tol
  PROXIMITY SEARCH TYPE = RAY SEARCH|SPHERE SEARCH
  RAY SEARCH DIRECTION = <real>vecx <real>vecy <real>vecz
  RAY SEARCH DIRECTION = ORTHOGONAL TO LINE <real>p1x
    <real>p1y <real>p1z <real>p2x <real>p2y <real>p2z
END [VARIABLE INTERPOLATION <string>var_interp_name]
```

For TRANSFER TYPE = SUM TO NEAREST ELEMENT, a given element in the source block computes its centroid. Then the closest face to that centroid on the target surface is found. The source element value is summed to the element associated with the target face.

The command SOURCE VARIABLE specifies the type of variable, ELEMENT, GLOBAL, or NODAL, and the name of the source variable source_var_name.

The command SOURCE SURFACE specifies the name of the source side set source_surface.

The command TARGET VARIABLE specifies the type of variable, ELEMENT, FACE, or NODAL, and the name of the target variable target_var_name. The target variable must exist. If necessary, a target variable can be created using the BEGIN USER VARIABLE command block (Section 11.2.4).

The command TARGET DERIVATIVE VARIABLE can be used to optionally compute the time derivative of the target variable. This command specifies the type of variable, ELEMENT, FACE, or NODAL, and the name of the variable to store the derivative, target_deriv_var_name. The

target derivative variable must exist. If necessary, a target variable can be created using the BEGIN USER VARIABLE command block (Section 11.2.4).

The command TARGET SURFACE specifies the name of the target side set target_surface.

The command SEARCH TOLERANCE specifies a search distance for use in the proximity search.

The command PROXIMITY SEARCH TYPE can be set to either RAY SEARCH or SPHERE SEARCH. When PROXIMITY SEARCH TYPE = SPHERE SEARCH the SEARCH TOLERANCE is used as the radius for closest point searches. When PROXIMITY SEARCH TYPE = RAY SEARCH the SEARCH TOLERANCE and any RAY SEARCH DIRECTION are used together to form rays with lengths equal to the SEARCH TOLERANCE in both the positive and negative directions from an associated RAY SEARCH DIRECTION. Faces penetrated by the ray get included in the set of faces for determining the closest point.

The command RAY SEARCH DIRECTION specifies a search direction for the PROXIMITY SEARCH = RAY SEARCH DIRECTION case. A RAY SEARCH DIRECTION can be specified directly with three values, vecx, vecy, and vecz. Alternatively, RAY SEARCH DIRECTION can be calculated through options ORTHOGONAL TO LINE or TARGET SURFACE NORMAL. For ORTHOGONAL TO LINE the search direction is the vector between a given point and the closest point on the infinite line specified with the two points p1x, p1y, p1z and p2x, p2y, p2z. For TARGET SURFACE NORMAL the search direction is the nodal normal vector calculated from the TARGET SURFACE.

9.8 Global Output Options

The following commands exist at the region scope to control the output of global variables:

```
GLOBAL ENERGY REPORTING = EXACT|APPROXIMATE|OFF (EXACT)
EXTENSIVE RIGID BODY VARS OUTPUT = OFF|HISTORY|RESULTS|ALL (ALL)
```

Through the `GLOBAL ENERGY REPORTING` command line Sierra/SM offers three reporting options for global energy variables: `EXACT`, `APPROXIMATE`, and `OFF`. The `EXACT` and `APPROXIMATE` reporting options use different algorithms for tracking the global values of external energy, internal energy, contact energy, and hourglass energy. In many cases, the `APPROXIMATE` reporting option will provide a modest performance improvement with a negligible effect on the reported energy values. The `OFF` option will result in a further performance improvement and will report energy values of zero.

Note that the `GLOBAL ENERGY REPORTING` command has no effect on the analysis itself; the energy values calculated are used only for reporting purposes.

The line command `EXTENSIVE RIGID BODY VARS OUTPUT` controls the default output of global rigid body variables. Regardless of the option choice here, global rigid body variables may be output by name in the history or results output blocks. See Table 9.2 for a list of available variables. The `EXTENSIVE RIGID BODY VARS OUTPUT` options are: `OFF` to specify no default rigid body global variable output; `HISTORY` to specify default rigid body global variable output to the history file(s) only; `RESULTS` to specify default rigid body global variable output to the results file(s) only; `ALL` to specify default rigid body global variable output to both the history and results files. This option defaults to `ALL` so that if this command is not specified both the history and results files will contain the variables listed in Table 9.2 at every output time.

9.9 Variables

This section lists commonly used variables that the user can select as output to the results file and the history file. The first part of this section lists global, nodal, and element variables. The second part of this section lists variables associated with material models.

9.9.1 Global, Nodal, Face, and Element Variables

This section lists commonly used global, nodal, and element variables. The variables are presented in tables based on use, as follows:

- Table 9.1 Global Variables for All Analyses
- Table 9.2 Global Variables for Rigid Bodies
- Table 9.3 Global Variables for J-Integral
- Table 9.4 Nodal Variables for All Analyses
- Table 9.9.1 Nodal Variables for Implicit Analyses
- Table 9.6 Nodal Variables for Shells
- Table 9.9.1 Nodal Variables for Spot Welds
- Table 9.8 Nodal Variables for Contact
- Table 9.9 Nodal Variables for J-Integral
- Table 9.10 Element Variables for All Elements
- Table 9.11 Element Variables for Solid Elements
- Table 9.12 Element Variables for Membranes
- Table 9.13 Element Variables for Shells
- Table 9.14 Element Variables for Trusses
- Table 9.15 Element Variables for Cohesive Elements
- Table 9.16 Element Variables for Beams
- Table 9.8 Element Variables for Springs
- Table 9.18 Element Variables for J-Integral

The tables provide the following information about each variable:

Variable Name. This is the string that will appear on the GLOBAL, NODE, FACE, or ELEMENT command line.

Type. This is the variable's type. The various types are denoted with the labels Integer, Integer[], Real, Real[], Vector_2D, Vector_3D, SymTen33, and FullTen36. The type Integer indicates the variable is an integer; the type Integer[] is an integer array; the type Real indicates the variable is a real; the type Real[] is a real array. The type Vector_2D indicates the variable type is a two-dimensional vector. The type Vector_3D indicates the variable is a three-dimensional vector. For a three-dimensional vector, the variable quantities will be output with suffixes of _x, _y, and _z. For example, if the variable displacement is requested to be output as displ, the components of the displacement vector on the results file will be displ_x, displ_y, and displ_z. The type SymTen33 indicates the variable is a symmetric 3×3 tensor. For a 3×3 symmetric tensor, the variable quantities will be output with suffixes of _xx, _yy, _zz, _xy, _yz, and _zx. For example, if the variable stress is requested for output as stress, the components of the stress tensor on the results file will be stress_xx, stress_yy, stress_zz, stress_xy, stress_yz, and stress_zx. The type FullTen36 is a full 3×3 tensor with three diagonal terms and six off-diagonal terms.

Derived. Any variable designated with a yes in this column must be included in a BEGIN DERIVED OUTPUT command block if it is to be transferred to another procedure or region as described in Section 6.7.

For multi-integration point elements, quantities from the element integration points will be appended with a numerical suffix indicating the integration point. A suffix ranging from 1 to the number of integration points is attached to the quantity to indicate the corresponding integration point. The suffix is padded with leading zeros. If the number of integration points is less than 10, the suffix has the form _*i*, where *i* ranges from 1 to the number of integration points. If the number of integration points is greater than or equal to 10 and less than 100, the sequence of suffixes takes the form _01, _02, _03, and so forth. Finally, if the number of integration points is greater than or equal to 100, the sequence of suffixes takes the form _001, _002, _003, and so forth. As an example, if von_mises is requested for a shell element with 15 integration points, then the quantities von_mises_01, von_mises_02, ..., von_mises_15 are output for the shell element.

The tables of various types of variables follow.

Table 9.1: Global Variables For All Analyses

Variable Name	**Type**	**Comments**
artificial_energy	Real	
contact_energy	Real	(A component of external energy)
external_energy	Real	
ke_blockblockID	Real	Kinetic energy sum for block blockID
ee_strain_blockblockID	Real	External energy sum for block blockID
ie_strain_blockblockID	Real	Internal energy sum for block blockID
momentum_blockblockID	Vector_3D	Momentum sum for block blockID
hourglass_energy	Real	(A component of internal energy)
hge_blockblockID	Real	Hourglass energy sum for block blockID
internal_energy	Real	
kinetic_energy	Real	
momentum	Vector_3D	Momentum vector
angular_momentum	Vector_3D	Angular momentum vector
timestep	Real	Current time step
timestep_element	Real	Time step from element estimator
timestep_nodal	Real	Time step from nodal estimator
timestep_material	Real	Time step from material model
timestep_lanczos	Real	Time step from Lanczos estimator
timestep_powermethod	Real	Time step from power method estimator
wall_clock_time	Real	Accumulated wall clock time
wall_clock_time_per_ step	Real	Wall clock time for last time step
cpu_time	Real	Accumulated CPU time
cpu_time_per_step	Real	CPU time for last time step

Table 9.2: Global Variables for Rigid Bodies. (See Section 9.8 for default output options.)

Variable Name	Type	Comments
ax,ay,az	Real	Translational acceleration
velx,vely,velz	Real	Translational velocity
displx,disply,displz	Real	Translational displacement
rotax,rotay,rotaz	Real	Rotational acceleration
rotvx,rotvy,rotvz	Real	Rotational velocity
rotdx,rotdy,rotdz	Real	Rotational displacement
reactx,reacty,reactz	Real	Translational reaction
rreactx,rreacty, rreactz	Real	Rotational reaction
qvec1,qvec2,qvec3, qvec4	Real	Unit quaternion

Table 9.3: Global Variables for *J*-Integral (See Section 10.2)

Variable Name	Type	Comments
j_average_<jint_name>	Real[]	Average value of the *J*-integral over the crack. Array sized to number of integration domains and numbered from inner to outer domain. <jint_name> is the name of the J INTEGRAL block.

Table 9.4: Nodal Variables for All Analyses

Variable Name	**Type**	**Comments**
model_coordinates	Vector_3D	Original coordinates of nodes
coordinates	Vector_3D	Current coordinates of nodes
displacement	Vector_3D	Total displacement
velocity	Vector_3D	
acceleration	Vector_3D	
force_internal	Vector_3D	
force_external	Vector_3D	
force external_ transferred	Vector_3D	Force transferred from another physics (coupled problems only)
force_contact	Vector_3D	
reaction	Vector_3D	
mass	Real	
nodal_time_step	Real	Nodal stable time step (explicit control modes, coarse mesh only)
hourglass_energy	Real	Nodal integrated energy due to hourglass forces
quaternion	Real	Current quaternion (rigid body reference nodes only)

Table 9.5: Nodal Variables for Implicit Analyses

Variable Name	**Type**	**Comments**
displacement_increment	Vector_3D	Displacement increment at current time step
residual	Vector_3D	Force imbalance at current time step

Table 9.6: Nodal Variables for Shells and Beams

Variable Name	**Type**	**Comments**
rotational_displacement	Vector_3D	
rotational_velocity	Vector_3D	
rotational_acceleration	Vector_3D	
moment_internal	Vector_3D	
moment_external	Vector_3D	
moment external_ transferred	Vector_3D	Moment transferred from another physics (coupled problems only)
rotational_reaction	Vector_3D	
rotational_mass	Real	

Table 9.7: Nodal Variables for Spot Welds

Variable Name	Type	Comments
spot_weld_parametric_coordinates	Vector_2D	Coordinates of node on face
spot_weld_normal_force_at_death	Real	Value of force normal to face when spot weld breaks
spot_weld_tangential_force_at_death	Real	Value of force tangential to face when spot weld breaks
spot_weld_death_flag	Integer	alive = 0, dead = FAILURE DECAY CYCLES (default is 10), -1 = no spot weld constructed at this node
spot_weld_scale_factor	Real	Nodal influence area of current node
spot_weld_normal_displacement	Real	Current displacement of weld normal to face
spot_weld_tangential_displacement	Real	Current displacement of weld tangential to face
spot_weld_normal_force	Real	Current force of weld normal to face
spot_weld_tangential_force	Real	Current force of weld tangential to face
spot_weld_stiffness	Real	Current stiffness of weld
spot_weld_norm_stiffness	Real	Current stiffness of weld normal to face
spot_weld_tang_stiffness	Real	Current stiffness of weld tangential to face
spot_weld_initial_offset	Vector_3D	The initial offset of the spot weld node from the spot weld surface. Does not change over time, only output if IGNORE INITIAL OFFSET = YES is specified at input.
spot_weld_initial_normal	Vector_3D	The initial normal of the spot weld surface at the point of interaction. Only output if IGNORE INITIAL OFFSET = YES is specified at input.

Table 9.8: Nodal Variables for Contact (See Section 8.7)

Variable	Type	Description
contact_status	Real	Status of the interactions at the node. Possible values are as follows: 0.0 = Node is not a contact node (not in a defined contact surface) 0.5 = Node is not in contact 1.0 = Node is in contact and is slipping -1.0 = Node is in contact and is sticking (celement).
contact_normal_ direction	Vector_3D	Direction of the constraint. This is, in general, the normal of the face in the interaction (cdirnor).
contact_tangential_ direction	Vector_3D	Direction of the contact tangential force (cdirtan).
contact_normal_force_ magnitude	Real	Magnitude of the contact force at the node in the direction normal to the contact face. Magnitude of contact_normal_ direction.
contact_tangential_ force_magnitude	Real	Magnitude of the contact force at the node in the plane of the contact face. Magnitude of contact_tangential_direction.
contact_normal_ traction_magnitude	Real	Traction normal to the contact face. contact_normal_force_magnitude divided by contact_area. If there are multiple interactions for this node, the traction only for the last interaction is given (cfnor).
contact_tangential_ traction_magnitude	Real	Traction in the plane of the contact face. contact_traction_force_ magnitude divided by contact_area. If there are multiple interactions for this node, the traction only for the last interaction is given (cftan).
		Continued on next page

Table 9.8 – Continued from previous page

Variable	Type	Description
contact_incremental_ slip_magnitude	Real	Magnitude of incremental slip over the current time step (cdtan).
contact_incremental_ slip_direction	Vector_3D	Normalized direction of incremental slip over the current time step (cdirislp).
contact_accumulated_ slip	Real	Magnitude of tangential slip accumulated over the entire analysis. This is the distance along the slip path, and not the magnitude of contact_accumulated_slip_ vector (cstan).
contact_accumulated_ slip_vector	Vector_ 3D)	Total accumulated tangential slip over the entire analysis (cdirslp).
contact_frictional_ energy	Real	Accumulated amount of frictional energy dissipated over the entire analysis.
contact_frictional_ energy_density	Real	Accumulated amount of frictional energy dissipated over the entire analysis, divided by the contact area (cetan).
contact_area	Real	Contact area for the node. This is the tributary area around the node for this interaction. If there are multiple interactions, the reported area is the area associated with the last interaction (carea).
contact_normal_gap	Real	Magnitude of gap in the direction normal to the face (cgnor).
contact_tangential_gap	Real	Magnitude of gap in the direction tangent to the face (only applicable for compliant friction models) (cgtan).

Table 9.9: Nodal Variables for J-Integral (See Section 10.2)

Variable Name	**Type**	**Comments**
j_<jint_name>	Real[]	Pointwise value of J-integral along crack. Array sized to number of integration domains and numbered from inner to outer domain. <jint_name> is the name of the J INTEGRAL block.

Table 9.10: Element Variables for All Elements

Variable Name	**Type**	**Derived** (Sec 6.7)	**Comments**
diagonal_ratio	Real		See Section 2.4
element_mass	Real		
perimeter_ratio	Real		See Section 2.4
solid_angle	Real		See Section 2.4
timestep	Real		Critical time step for the element. The element in the model with the smallest time step controls the analysis time step.
von_mises	Real	yes	Von Mises stress norm
hydrostatic_stress	Real	yes	One-third the trace of the stress sensor
fluid_pressure	Real	yes	Negative of hydrostatic_stress
stress_invariant_1	Real	yes	Trace of the stress tensor
stress_invariant_2	Real	yes	Second invariant of the stress tensor
stress_invariant_3	Real	yes	Third invariant of the stress tensor
max_principal_stress	Real	yes	Largest eigenvalue of the stress tensor
intermediate_principal_stress	Real	yes	Middle eigenvalue of the stress tensor
min_principal_stress	Real	yes	Smallest eigenvalue of the stress tensor
max_shear_stress	Real	yes	Maximum shear stress from Mohr's circle
octahedral_shear_stress	Real	yes	Octahedral shear norm of the stress tensor

Table 9.11: Element Variables for Solid Elements

Variable Name	**Type**	**Derived** (Sec 6.7)	**Comments**
aspect_ratio	Real		Tets only. See Section 2.4
dilmod	Real		
left_stretch	SymTen33		
nodal_jacobian_ratio	Real	yes	Hexes only. See Section 2.4
element_shape	Real	yes	Element shape quality metric. See Section 2.4
rate_of_deformation	SymTen33		Hexes and node-based tets only.
rotation	FullTen36		
shrmod	Real		
stress	SymTen33		
unrotated_stress	SymTen33		
log_strain	SymTen33		Log strain tensor
unrotated_log_strain	SymTen33		Log strain tensor in unrotated configuration
effective_log_strain	Real	yes	Effective log strain
log_strain_invariant_1	Real	yes	Trace of the log strain tensor
log_strain_invariant_2	Real	yes	Second invariant of the log strain tensor
log_strain_invariant_3	Real	yes	Third invariant of the log strain tensor
max_principal_log_strain	Real	yes	Largest eigenvalue of the log strain tensor
intermediate_principal_log_strain	Real	yes	Middle eigenvalue of the log strain tensor
min_principal_log_strain	Real	yes	Smallest eigenvalue of the log strain tensor
max_shear_log_strain	Real	yes	Maximum shear log strain from Mohr's circle
octahedral_shear_log_strain	Real	yes	Octahedral strain norm of the log strain tensor
volume	Real		

Table 9.12: Element Variables for Membranes

Variable Name	Type	Comments
memb_stress	SymTen33	
element_area	Real	
element_thickness	Real	

Table 9.13: Element Variables for Shells

Variable Name	Type	Derived (Sec 6.7)	Comments
memb_stress	SymTen33		Stress at midplane in global X, Y, and Z coordinates
bottom_stress	SymTen33		Stress at bottom integration point in global X, Y, and Z coordinates
top_stress	SymTen33		Stress at top integration point in global X, Y, and Z coordinates
unrotated_stress	SymTen33		
transform_shell_stress	SymTen21	yes	In-plane shell stress
strain	SymTen33		Integrated strain at midplane in local shell coordinate system
effective_strain	Real	yes	Effective strain norm
strain_invariant_1	Real	yes	Trace of the strain tensor
strain_invariant_2	Real	yes	Second invariant of the strain tensor
strain_invariant_3	Real	yes	Third invariant of the strain tensor
max_principal_strain	Real	yes	Largest eigenvalue of the strain tensor
intermediate_principal_strain	Real	yes	Middle eigenvalue of the strain tensor
min_principal_strain	Real	yes	Smallest eigenvalue of the strain tensor
max_shear_strain	Real	yes	Maximum shear strain from Mohr's circle
octahedral_shear_strain	Real	yes	Octahedral strain norm of the strain tensor
transform_shell_strain	Real	yes	In-plane shell strain
element_area	Real		
element_thickness	Real		
rate_of_deformation	SymTen33		Rate of deformation (stretching) tensor

Table 9.14: Element Variables for Trusses

Variable Name	Type	Comments
truss_init_length	Real	
truss_stretch	Real	
stress	SymTen33	Axial stress is stored in stress_xx. All other components are zero. See Section 6.2.7 for more details.
truss_strain_incr	Real	
truss_force	Real	

Table 9.15: Element Variables for Cohesive Elements

Variable Name	Type	Comments
cse_traction	Vector_3D	
cse_separation	Vector_3D	
cse_initial_trac	Vector_3D	Available only if traction initialization is used
cse_activated	Integer	For intrinsic elements
cse_fracture_area	Real	Currently not used

Table 9.16: Element Variables for Beams

Variable Name	Type	Comments
beam_strain_inc	Vector_2D	Thirty-two strain increment values are output. Some values may be zero depending on section. Axial strains are 01, 03, 05, . . . Shear strains are 02, 04, 06, . . . See Section 6.2.6 for more details.
stress	SymTen33	Ninety-six stress values are output, although only the first two values per integration point contain actual stress values. Some integration points may not have data depending on the section. Axial stresses are in stress_xx_01, stress_xx_02, . . . , stress_xx_16. Shear stresses are in stress_xy_01, stress_xy_02, . . . , stress_xy_16, where 01, 02, . . . , 16, refer to the integration points. See Section 6.2.6 for more details.
beam_stress_axial	Real	Sixteen axial stress values. Some may be zero depending on section.
beam_stress_shear	Real	Sixteen shear stress values. Some may be zero depending on section.
beam_axial_force	Real	Axial force at midpoint.
beam_transverse_force_s	Real	Transverse shear in s-direction at midpoint.
beam_transverse_force_t	Real	Transverse shear in t-direction at midpoint.
beam_moment_r	Real	Torsion at midpoint.
beam_moment_s	Real	Moment about s-direction at midpoint.
beam_moment_t	Real	Moment about t-direction at midpoint.
beam_avg_rate_of_def	Real	Average rate of deformation over all integration point.

Table 9.17: Element Variables for Springs

Variable Name	Type	Comments
spring_force	Real	Magnitude of the internal spring force.
spring_engineering_strain	Real	Change in length over initial length $\frac{dL}{L_0}$.
spring_init_length	Real	Initial spring length, L_0.

Table 9.18: Element Variables for J-Integral (See Section 10.2)

Variable Name	**Type**	**Comments**
energy_momentum_tensor	FullTen36	Energy momentum tensor
integration_domains_ <jint_name>	Integer[]	Flag indicating elements in integration domains. Set to 1 if in domain, 0 otherwise. Array sized to number of domains and numbered from inner to outer domain. <jint_name> is the name of the J INTEGRAL block.

9.9.2 Variables for Material Models

It is possible to output the state variables from the material models. Most of the materials, with the exception of simple models such as the elastic model, have state variables that can be output. State variables can be accessed by name or index (most of the time they are accessed by name and under special circumstances, by index). The following sections describe the different methods required to output material model variables.

9.9.2.1 State Variable Output for LAME Solid Material Models

The state variables for material models in LAME are accessible directly by name. For instance, the equivalent plastic strain variable is accessible by the name `EQPS` for all elastic-plastic material models.

Section 9.9.2.2 provides tables listing the state variables for all solid material models.

Available LAME state variables for a material will also be listed in the log file from a run that uses the material model.

9.9.2.2 State Variable Tables for Solid Material Models

As explained in the preceding section LAME model variables are obtained by variable name or index. Tables of state variables for commonly used material models are provided in Tables 9.23 through 9.45. These tables contain the indices or names used to access the material state variables.

Table 9.19: State Variables for ELASTIC Model (Section 5.2.1)

This model has no state variables.

Table 9.20: State Variables for ELASTIC FRACTURE Model (Section 5.2.4)

Index	Name	Variable Description
1	DEATH_FLAG	flag for element death
2	CRACK_OPENING_ STRAIN	critical value of opening strain
3	FAILURE_ DIRECTION_X	crack opening direction - x component
4	FAILURE_ DIRECTION_Y	crack opening direction - y component
5	FAILURE_ DIRECTION_Z	crack opening direction - z component
6	PRINCIPAL_ STRESS	value of maximum principal stress

Table 9.21: State Variables for ELASTIC PLASTIC Model (Section 5.2.5)

Index	Name	Variable Description
1	EQPS	equivalent plastic strain
2	BACK_STRESS_XX	back stress - xx component
3	BACK_STRESS_YY	back stress - yy component
4	BACK_STRESS_ZZ	back stress - zz component
5	BACK_STRESS_XY	back stress - xy component
6	BACK_STRESS_YZ	back stress - yz component
7	BACK_STRESS_ZX	back stress - zx component
8	RADIUS	radius of yield surface

Table 9.22: State Variables for EP POWER HARD Model (Section 5.2.6)

Index	Name	Variable Description
1	EQPS	equivalent plastic strain
2	RADIUS	radius of yield surface

Table 9.23: State Variables for DUCTILE FRACTURE Model (Section 5.2.7)

Index	**Name**	**Variable Description**
1	EQPS	equivalent plastic strain
2	RADIUS	radius of yield surface
3	BACK_STRESS_XX	back stress - xx component
4	BACK_STRESS_YY	back stress - yy component
5	BACK_STRESS_ZZ	back stress - zz component
6	BACK_STRESS_XY	back stress - xy component
7	BACK_STRESS_YZ	back stress - yz component
8	BACK_STRESS_ZX	back stress - zx component
9	TEARING_ PARAMETER	tearing parameter
10	CRACK_OPENING_ STRAIN	crack opening strain
11	FAILURE_ DIRECTION_X	crack opening direction - x component
12	FAILURE_ DIRECTION_Y	crack opening direction - y component
13	FAILURE_ DIRECTION_Z	crack opening direction - z component
14	CRACK_FLAG	crack flag for element death
15	DF_STRAIN_XX	
16	DF_STRAIN_YY	
17	DF_STRAIN_ZZ	
18	DF_STRAIN_XY	
19	DF_STRAIN_YZ	
20	DF_STRAIN_ZX	
21	FAILURE_RATIO	
22	DECAY	
23	MAX_RADIUS	maximum radius at failure
24	MAX_PRESS	maximum pressure at failure

Table 9.24: State Variables for MULTILINEAR EP Model (Section 5.2.8)

Index	**Name**	**Variable Description**
1	EQPS	equivalent plastic strain
2	RADIUS	radius of yield surface
3	BACK_STRESS_XX	back stress - xx component
4	BACK_STRESS_YY	back stress - yy component
5	BACK_STRESS_ZZ	back stress - zz component
6	BACK_STRESS_XY	back stress - xy component
7	BACK_STRESS_YZ	back stress - yz component
8	BACK_STRESS_ZX	back stress - zx component
9	YOUNGS_MODULUS	
10	POISSONS_RATIO	
11	YIELD_STRESS	
12	TENSILE_EQPS	Equivalent plastic strain only accumulated in tension
13	ITERATIONS	radial return iterations
14	YIELD_FLAG	

Table 9.25: State Variables for ML EP FAIL Model (Section 5.2.9)

Index	Name	Variable Description
1	EQPS	equivalent plastic strain
2	RADIUS	radius of yield surface
3	BACK_STRESS_XX	back stress - xx component
4	BACK_STRESS_YY	back stress - yy component
5	BACK_STRESS_ZZ	back stress - zz component
6	BACK_STRESS_XY	back stress - xy component
7	BACK_STRESS_YZ	back stress - yz component
8	BACK_STRESS_ZX	back stress - zx component
9	TEARING_ PARAMETER	tearing parameter
10	CRACK_OPENING_ STRAIN	crack opening strain
11	FAILURE_ DIRECTION_X	crack opening direction - x component
12	FAILURE_ DIRECTION_Y	crack opening direction - y component
13	FAILURE_ DIRECTION_Z	crack opening direction - z component
14	CRACK_FLAG	status of the model: 0 for loading, 1 or 2 for initiation of failure, 3 during unloading, 4 for completely unloaded
15	DECAY	
16	FAILURE_ DIRECTION	crack opening direction
17	FAILURE_RATIO	
18	ITERATIONS	radial return iterations
19	MAX_RADIUS	maximum radius at failure
20	ML_STRAIN_XX	
21	ML_STRAIN_XY	
22	ML_STRAIN_YY	
23	ML_STRAIN_YZ	
24	ML_STRAIN_ZX	
25	ML_STRAIN_ZZ	
26	POISSONS_RATIO	
27	TENSILE_EQPS	Equivalent plastic strain only accumulated in tension
28	YIELD_FLAG	
29	YIELD_STRESS	
30	CRITICAL_ CRACK_OPENING_ STRAIN	
31	CRITICAL_ TEARING_ PARAMETER	

Table 9.26: State Variables for FOAM PLASTICITY Model (Section 5.2.15)

Index	**Name**	**Variable Description**
1	ITER	iterations
2	EVOL	volumetric strain
3	PHI	phi
4	EQPS	equivalent plastic strain
5	PA	A
6	PB	B

Table 9.27: State Variables for WIRE MESH Model (Section 5.2.18)

Index	**Name**	**Variable Description**
1	EVOL	engineering volumetric strain
2	PHI	current yield strength in compression

Table 9.28: State Variables for HONEYCOMB Model

Index	**Name**	**Variable Description**
1	CRUSH	minimum volume ratio
2	EQDOT	effective strain rate
3	RMULT	rate multiplier
5	ITER	iterations
6	EVOL	volumetric strain

Table 9.29: State Variables for HYPERFOAM Model

This model has no state variables.

Table 9.30: State Variables for JOHNSON COOK Model (Section 5.2.10)

Index	**Name**	**Variable Description**
1	`RADIUS`	radius of yield surface
2	`EQPS`	equivalent plastic strain
3	`THETA`	temperature
4	`EQDOT`	effective total strain rate
5	`ITER`	

Table 9.31: State Variables for LOW DENSITY FOAM Model (Section 5.2.16)

Index	**Name**	**Variable Description**
	`PAIR`	air pressure

Table 9.32: State Variables for MOONEY RIVLIN Model (Section 5.2.25)

Name	Name	Variable Description
1	C10	
2	C01	
3	K	
4	SFJTH	
5	JTH	
6	VMECH_XX	
7	VMECH_YY	
8	VMECH_ZZ	
9	VMECH_XY	
10	VMECH_YZ	
11	VMECH_ZX	
12	SFJTH_FLAG	
13	WDEV	

Table 9.33: State Variables for NEO HOOKEAN Model (Section 5.2.3)

This model has no state variables.

Table 9.34: State Variables for ORTHOTROPIC CRUSH Model (Section 5.2.19)

Name	Name	Variable Description
CRUSH	CRUSH	minimum volume ratio, crush is unrecoverable

Table 9.35: State Variables for ORTHOTROPIC RATE Model (Section 5.2.20)

Index	Name	Variable Description
1	CRUSH	minimum volume ratio, crush is unrecoverable

Table 9.36: State Variables for PIEZO Model

Index	Name	Variable Description
1	STATE	

Table 9.37: State Variables for POWER LAW CREEP Model (Section 5.2.12)

Index	**Name**	**Variable Description**
1	ECREEP	equivalent creep strain
2	SEQDOT	equivalent stress rate

Table 9.38: State Variables for SHAPE MEMORY Model

Index	**Name**	**Variable Description**
1	STATE	

Table 9.39: State Variables for SOIL FOAM Model (Section 5.2.13)

Index	**Name**	**Variable Description**
1	EVOL_MAX	
2	EVOL_FRAC	
3	EVOL	

Table 9.40: State Variables for SWANSON Model (Section 5.2.28)

Index	Name	Variable Description
1	SFJTH	
2	JTH	
3	VMECHXX	
4	VMECHYY	
5	VMECHZZ	
6	VMECHXY	
7	VMECHYZ	
8	VMECHZX	
9	SFJTH_FLAG	
10	WDEV	

Table 9.41: State Variables for VISCOELASTIC SWANSON Model (Section 5.2.29)

Index	Name	Variable Description
	SFJTH	
	JTH	
	VMECHXX	
	VMECHYY	
	VMECHZZ	
	VMECHXY	
	VMECHYZ	
	VMECHZX	
	VSXXDEV1 – VSXXDEV10	
	VSYYDEV1 – VSYYDEV10	
	VSZZDEV1 – VSZZDEV10	
	VSXYDEV1 – VSXYDEV10	
	VSYZDEV1 – VSYZDEV10	
	VSZXDEV1 – VSZXDEV10	
	SOXXDEV	
	SOYYDEV	
	SOZZDEV	
	SOXYDEV	
	SOYZDEV	
	SOZXDEV	

Table 9.42: State Variables for THERMO EP POWER Model

Index	Name	Variable Description
1	EQPS	equivalent plastic strain
2	RADIUS	radius of yield surface
3	BACK_STRESS_XX	back stress - xx component
4	BACK_STRESS_YY	back stress - yy component
5	BACK_STRESS_ZZ	back stress - zz component
6	BACK_STRESS_XY	back stress - xy component
7	BACK_STRESS_YZ	back stress - yz component
8	BACK_STRESS_ZX	back stress - zx component

Table 9.43: State Variables for THERMO EP POWER WELD Model

Index	Name	Variable Description
	EQPS	equivalent plastic strain
	RADIUS	radius of yield surface
	BACK_STRESS_XX	back stress - xx component
	BACK_STRESS_YY	back stress - yy component
	BACK_STRESS_ZZ	back stress - zz component
	BACK_STRESS_XY	back stress - xy component
	BACK_STRESS_YZ	back stress - yz component
	BACK_STRESS_ZX	back stress - zx component
	WELD_FLAG	

Table 9.44: State Variables for UNIVERSAL POLYMER Model

Index	Name	Variable Description
	AEND	
	IGXX1 - IGXX20	
	IGYY1 - IGYY20	
	IGZZ1 - IGZZ20	
	IGXY1 - IGXY20	
	IGYZ1 - IGYZ20	
	IGZX1 - IGZX20	
	IKI11 - IKI120	
	IKAT1 - IKAT20	
	IF1P1 - IF1P20	
	IF2J1 - IF2J20	
	EPSXX	
	EPSYY	
	EPSZZ	
	EPSXY	
	EPSYZ	
	EPSZX	
	LOGA	

Table 9.45: State Variables for VISCOPLASTIC Model

Index	**Name**	**Variable Description**
	SVBXX	
	SVBYY	
	SVBZZ	
	SVBXY	
	SVBYZ	
	SVBZX	
	EQDOT	
	COUNT	
	SHEAR	
	BULK	
	RATE	
	EXP	
	ALPHA	
	A1	
	A2	
	A4	
	A5	

9.9.2.3 Variables for Shell/Membrane Material Models

Shell and membrane material models also make their state variables available through direct naming of the variables. Tables 9.46 through 9.49 indicate the names of the state variables for the shell material models.

Table 9.46: State Variables for ELASTIC PLASTIC Model for Shells (Section 5.2.5)

Variable Name	Variable Description
eqps	Equivalent plastic strain
back_stress	Back stress
back_stress_xx	Back stress xx component
back_stress_xy	Back stress xy component
back_stress_yy	Back stress yy component
back_stress_yz	Back stress yz component
back_stress_zx	Back stress zx component
back_stress_zz	Back stress zz component
radius	Radius of the yield surface
error	Error in plane stress iterations
iterations	radial return iterations
ps_iter	Plane stress iterations
tensile_eqps	Equivalent plastic strain only accumulated in tension
tstrain	Integrated thickness strain

Table 9.47: State Variables for EP POWER HARD Model for Shells (Section 5.2.6)

Variable Name	Variable Description
eqps	Equivalent plastic strain
radius	Radius of yield surface

Table 9.48: State Variables for MULTILINEAR EP Model for Shells (Section 5.2.8)

Variable Name	Variable Description
eqps	Equivalent plastic strain
tensile_eqps	Equivalent plastic strain only accumulated in tension
back_stress	Back stress
radius	Radius of the yield surface

Table 9.49: State Variables for ML EP FAIL Model for Shells (Section 5.2.9)

Variable Name	**Variable Description**
eqps	Equivalent plastic strain
back_stress	Back stress
radius	Radius of the yield surface
tearing_parameter	The current value of the tearing parameter
crack_opening_strain	The value of the crack opening strain during the failure process
crack_flag	Status of the model: 0 for loading, 1 or 2 for initiation of failure, 3 during unloading, 4 for completely unloaded
error	Error in plane stress iterations
tensile_eqps	Equivalent plastic strain only accumulated in tension
iter	Radial return iterations
ps_iter	Plane stress iterations
poissons_ratio	
youngs_modulus	
yield_stress	

9.9.3 Variables for Surface Models

It is possible to output the state variables from the surface models. The element state variables are output using the variable name by use of following line command:

```
ELEMENT surface_model_state_name
```

Section 9.9.3.1 provides tables listing the state variables for all surface models.

9.9.3.1 State Variable Tables for Surface Models

Table 9.50: State Variables for TRACTION DECAY Surface Model (Section 5.3.1)

Index	Name	Variable Description
0	MAX_SEPARATION_S	maximum separation in the first tangential direction
1	MAX_SEPARATION_T	maximum separation in the second tangential direction
2	MAX_SEPARATION_N	maximum separation in the normal direction

Table 9.51: State Variables for TVERGAARD HUTCHINSON Surface Model (Section 5.3.2)

Index	Name	Variable Description
0	PEAK_TRACTION	maximum traction the model can experience
1	LAMBDA_MAX	maximum lambda the model has experienced
2	TRACTION_AT_LAMBDA_MAX	traction at LAMBDA_MAX

Table 9.52: State Variables for THOULESS PARMIGIANI Surface Model (Section 5.3.3)

Index	Name	Variable Description
0	FRACTION_OF_ FAILURE	current fraction of failure
1	PEAK_TRACTION_ N	maximum normal traction the model can experience
2	PEAK_TRACTION_ T	maximum tangential traction the model can experience
3	LAMBDA_MAX_N	maximum normal lambda the model has experienced
4	TRACTION_AT_ LAMBDA_MAX_N	normal traction at LAMBDA_MAX_N
5	LAMBDA_MAX_T	maximum tangential lambda the model has experienced
6	TRACTION_AT_ LAMBDA_MAX_T	tangential traction at LAMBDA_MAX_T
7	G_AT_LAMBDA_ MAX_N	the area under the normal traction separation curve up to LAMBDA_MAX_N
8	G_AT_LAMBDA_ MAX_T	the area under the tangential traction separation curve up to LAMBDA_MAX_T

9.10 References

1. Larry A. Schoof, Victor R. Yarberry, *EXODUS II: A Finite Element Data Model*, SAND92-2137, Sandia National Laboratories, September 1994. pdf. See also documentation available at EXODUS II sourceforge page. link.

2. The eXtensible Data Model and Format (XDMF). link.

3. Sjaardema, G. D. *Overview of the Sandia National Laboratories Engineering Analysis Code Access System*, SAND92-2292. Albuquerque, NM: Sandia National Laboratories, January 1993. pdf.

Chapter 10

Special Modeling Techniques

This chapter describes techniques useful for performing special types of analyses:

- Section 10.1 describes the Representative Volume Element (RVE) capability, which is a multiscale technique that uses a separate finite element model to represent the material response at a point.
- Section 10.2 describes the capability to compute J-Integrals as a criterion for fracture growth.
- Section 10.2.3 describes the peridynamics functionality available in Sierra/SM. Peridynamics is a nonlocal formulation of continuum mechanics that is well-suited for modeling material discontinuities such as cracks.

10.1 Representative Volume Elements

The use of representative volume elements (RVEs) is a multiscale technique in which the material response at element integration points in a reference mesh is computed using an RVE that is itself discretized with finite elements. RVEs are typically used to represent local, periodic material inhomogeneities such as fibers or random microstructures to avoid the requirement of a global mesh with elements small enough to capture local material phenomena.

This capability is currently implemented only for uniform gradient hex elements in the reference mesh. In the current implementation of RVEs, periodic boundary conditions are applied to each RVE representing the deformation of a parent element and the stresses are computed in the elements of the RVE. These stresses are then volume-averaged over the RVE and the resulting homogenized stresses are passed back to the parent element.

This chapter explains how to use the RVE capability. Section 10.1.1 gives a detailed description of how RVEs are incorporated into an analysis. Details of the mesh requirements are delineated in Section 10.1.2 and the commands needed in an input file are described in Section 10.1.3.

10.1.1 RVE Processing

The use of the RVE capability requires two regions, each with its own mesh file. One region processes the reference mesh and the other processes all the RVEs. The commands used in the input file for the reference mesh region are the same as any other Sierra/SM region with the exception that a special RVE material model is used for any element blocks that use an RVE. The RVE region is also very similar to an ordinary region. The only differences are that an RVE region has a line command for defining the RVEs' relationship to parent elements in the reference region and has restrictions on the use of boundary conditions.

The processing of an RVE essentially replaces the constitutive model of the parent element in the reference mesh. The steps followed at each iteration/time step of the reference mesh during an analysis using RVEs are:

1. Internal force algorithm is called in the reference region to compute rate of deformation.
2. Each RVE gets the rate of deformation from its parent element in the reference region.
3. The rate of deformation is applied to each RVE as a periodic boundary condition using prescribed velocity.
4. The RVE region is solved to obtain the stress in each element of each RVE.
5. The stresses in the elements of an RVE are volume-averaged over the RVE.
6. Each RVE passes its homogenized (i.e. volume-averaged) stress tensor back to its parent element in the reference mesh.
7. The reference region computes internal force again. Element blocks whose elements have associated RVEs do not compute a stress; they simply use the stress passed to them from their RVE.

10.1.2 Mesh Requirements

Two mesh files, one each for the reference region and the RVE region, are required for an RVE analysis. Figure 10.1 shows an example of the two meshes. The reference mesh of a bar with six elements is shown on the upper left. On the lower right is the mesh for the RVE region containing six RVEs, one for each element of the reference region. In this case, the first five RVEs each consist of two element blocks and the last RVE has four.

In general, each RVE should be a cube with any discretization the user desires. All RVEs must be aligned with the global x, y, and z axes. For stress computations, these axes are rotated into a local coordinate system that can be specified on the reference mesh elements. In other words, if a local coordinate system is specified on a reference mesh element, the RVE global axes will be rotated internally in Sierra/SM to align with the local system on the associated parent element. So the global X axis for an RVE is actually the local X' axis in the parent element.

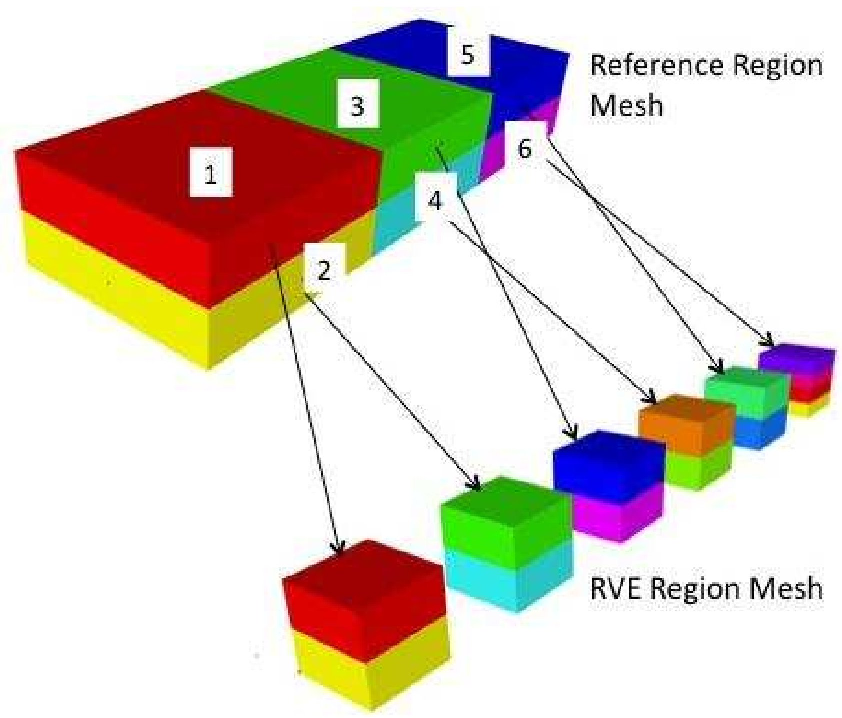


Figure 10.1: Example of meshes for RVE analysis

Additional mesh requirements apply if the mesh does not match across opposing surfaces of the RVE. In this case, the RVE must include a block of membrane elements on the exterior surfaces with matching discretization on opposing surfaces (+x/-x, +y/-y, +z/-z). In order the minimize the effects of this membrane layer on the RVE response, it should be made as thin as possible. This membrane layer then must be tied to the underlying nonmatching RVE surfaces.

The RVE mesh must contain sidesets or nodesets on each surface of every RVE. The RVE may be enclosed with one sideset that spans all six surfaces of the curb, or the user may specify individual sidesets or nodesets on each face. These sidesets/nodesets are used to apply the periodic boundary conditions on the RVE. Sierra/SM generates the boundary conditions internally so the user does not have to include them in the input file. However, this assumes that the sidesets/nodesets exist in the mesh file numbered in a specified order. If individual sidesets/nodesets are used on each face of the RVE, the six sidesets/nodesets must be numbered consecutively, starting with the positive-x face, followed by the negative-x face, positive-y face, negative-y face, positive-z face, and ending with the negative-z face. The beginning sideset id (for the positive-x face) is set by the user in the input file.

10.1.3 Input Commands

There are several input commands that are relevant to RVEs. In the reference region, these commands include a special RVE material model and commands to define and use a local coordinate system along which an associated RVE will be aligned. In addition to the reference region, an RVE region is needed using the `BEGIN RVE REGION` command block. The RVE region com-

mand block uses the same nested commands as any other Sierra/SM region (with some restrictions as explained in this section) and an additional line command that relates the RVEs to their parent elements in the reference region.

10.1.3.1 RVE Material Model

In an RVE analysis, any elements of the reference mesh that use an RVE must use the RVE material model. This model is defined similar to other material models as described in Chapter 5 but uses the RVE keyword on the `BEGIN PARAMETERS FOR MODEL` command line as follows:

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  #
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL RVE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
  END PARAMETERS FOR MODEL RVE
  #
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

Currently, the RVE material model tells the reference element not to perform a constitutive evaluation but to instead accept the stress tensor obtained from computation on an RVE. However, the use of an RVE material model still requires the input of Young's modulus and Poissons ratio. These values may be used for time time step estimation and hourglass computations even though they are not used in a constitutive evaluation.

Element blocks in the RVE region can use any material model that is supported in Sierra/SM other than RVE.

10.1.3.2 Embedded Coordinate System

The finite element model of an element block in the reference mesh that uses RVEs can use an embedded coordinate system to orient the RVE relative to the reference element. A coordinate system is defined in the sierra scope as described in Section 2.1.8. A local coordinate system is then associated with an element block through the use of a `COORDINATE SYSTEM` command line within a `BEGIN SOLID SECTION` command block.

```
BEGIN SOLID SECTION <string>section_name
  #
  COORDINATE SYSTEM = <string>coord_sys_name
  #
END [SOLID SECTION <string>section_name]
```

The string `coord_sys_name` must be a name associated in the input file with a `BEGIN COORDINATE SYSTEM` command block in the sierra scope. This coordinate system will then be used on all elements of a block associated with a `BEGIN PARAMETERS FOR BLOCK` command block that includes the command line specifying this solid section.

10.1.3.3 RVE Region

A representative volume element (RVE) region must be a quasistatic region specified with the RVE keyword in the `BEGIN RVE REGION` command line. The RVE region uses the same block commands and line commands as any other quasistatic region with the addition of line commands that define which element blocks of the reference region are associated with RVEs. There are also some restrictions on boundary conditions as described in Section 10.1.3.6.

```
BEGIN RVE REGION <string>rve_region_name
  #
  # Definition of RVEs
  ELEMENTS <integer>elem_i:<integer>elem_j
    BLOCKS <integer>blk_i:<integer>blk_j
    SURFACE|NODESET <integer>start_id INCREMENT <integer>k
  #
  # Boundary Conditions
  #
  # Results Output Definition
  #
  # Solver Definition
  #
END [RVE REGION <string>rve_region_name]
```

10.1.3.4 Definition of RVEs

One or more `ELEMENTS` command lines are used to associate elements of the reference region mesh with RVEs in the RVE region. In the

```
ELEMENTS <integer>elem_i:<integer>elem_j
  BLOCKS <integer>blk_i:<integer>blk_j
  SURFACE|NODESET <integer>start_id INCREMENT <integer>incr
```

command line, elements numbered `elem_i` through `elem_j` of the reference mesh will be associated with RVEs (for a total number of RVEs equal to `elem_j` - `elem_i` + 1), and each RVE will consist of `blk_i` - `blk_j` + 1 element blocks. The block ids of the first RVE must be `blk_i` through `blk_j` and subsequent RVEs (if `elem_j` is greater than `elem_i`) must have consecutively increasing numbers for their block ids.

Similarly `start_id` gives the `surface_id` of the first RVE if a single, encompassing surface is used, or the first `surface_id` or `nodelist_id` of the first RVE (the positive x surface as

explained in Section 10.1.2) if six individual sidesets/nodeset are used. The remaining surfaces (nodesets) of the first RVE and all the surfaces of the following RVEs must be consecutively numbered following `start_id` in the mesh file as explained in Section 10.1.2.

The increment value `incr` indicates the number of sidesets present on the exterior of the RVEs. This is used to determine how to increment the IDs of the sidesets from one RVE to the next, as well as to determine how to prescribe periodic boundary conditions on the RVE. The increment can have a value of either one or six. A value of one indicates that each RVE has one sideset that encompasses all six faces, while a value of six specifies that six sidesets or nodesets are present, one on each face. Note that nodesets are not allowed for the case where `incr` is one.

The following example shows the use of the `ELEMENTS` command line:

```
elements 1:5 blocks 1:2 surface 7 increment 6
elements 6:6 blocks 11:14 nodeset 15 increment 6
```

These commands generate the RVEs shown in Figure 10.1.

The first `ELEMENTS` command line specifies that elements with element ids 1 through 5 in the parent region mesh each have an RVE with two element blocks. The RVE associated with element 1 of the parent region will have two element blocks starting with `block_id` of 1 and ending with a `block_id` of 2. Subsequent RVEs will have consecutively numbered element blocks. That is, parent element 2 will be associated with an RVE that consists of element blocks 3 and 4 in the RVE region, parent element 3 will be associated with the RVE that has element blocks 5 and 6, etc., for the first five elements of the parent region mesh. The keyword `SURFACE` specifies that all the periodic boundary conditions generated by the code for the RVEs for elements 1 to 5 will use sidesets in the RVE region mesh. These sidesets will start with id 7 for the positive-x face of the RVE associated with parent element 1 and continue consecutively for the other faces of the RVE and the RVEs associated with parent elements 2 through 5 in the order specified in Section 10.1.2. In other words, the positive-x face of the RVE for parent element 1 is sideset 7, negative-x is sideset 8, positive-y is sideset 9, negative-y is sideset 10, positive-z is sideset 11, and negative-z is sideset 12. The sidesets for the RVE for parent element 2 will start with id 13 and continue consecutively in the same face order. The process continues for all five RVEs specified in this command line.

The second `ELEMENTS` line specifies that element 6 of the parent region mesh will be associated with the RVE that consists of element blocks 11, 12, 13, and 14. The `NODESET` keyword says this RVE has a nodeset associated with each face of the RVE, starting with nodeset id 15 on the positive-x face, with id's increasing consecutively for the other five faces in the same order described in the paragraph above.

Note that the six elements specified in these command lines must be in element blocks of the reference region mesh that use the RVE material model.

10.1.3.5 Multi-Point Constraints

In the case in which the RVE has nonmatching surfaces, and therefore includes a block of membrane elements on the exterior surfaces, the user must specify a set of multi-point constraints

(MPCs) to tie the membranes to the surface. This is done in the input file through use of an MPC command block:

```
RESOLVE MULTIPLE MPCS = ERROR
BEGIN MPC
  MASTER SURFACE = <string>membrane_surface_id
  SLAVE SURFACE  = <string>RVE_surface_id
  SEARCH TOLERANCE = <real>tolerance
END
```

In this case, the membrane_surface_id corresponds to the single sideset that encompasses the membrane block is the master and the single sideset that encompasses the exterior surfaces of the RVE is the slave. While the underlying RVE may have nonmatching exterior surfaces, the opposing surfaces of the membrane block must have matching discretizations. For more information on the use of MPCs, see Section 7.12.7.1 and the RESOLVE MULTIPLE MPCS command line is discussed in Section 7.12.7.4.

10.1.3.6 RVE Boundary Conditions

Strain rates computed by elements in the reference region are applied through periodic prescribed velocity boundary conditions on the faces of the associated RVEs. These are generated internally by Sierra/SM so the periodic boundary conditions do not need to be in the user's input file. However, because the RVE region is quasistatic, each of the RVEs must be fixed against rigid body motion. This must be done in the input file through use of the prescribed velocity boundary conditions:

```
BEGIN PRESCRIBED VELOCITY pres_vel_name
  NODE SET = <string>nodelist_name
  FUNCTION = <string>function_name
  SCALE FACTOR = <real>scale_factor
  COMPONENT = <string>X|Y|Z
END [PRESCRIBED VELOCITY pres_vel_name]
```

This type of boundary condition is described in detail in Chapter 7 but the use for RVEs is restricted. First, either the function must always evaluate to 0.0 or the scale_factor must have a value of 0. This is essentially a way of using the prescribed velocity boundary condition to fix the nodes in nodelist_name. However, in order for these conditions to work with the periodic boundary conditions which are used to apply the strain rate, PRESCRIBED VELOCITY must be used rather than FIXED DISPLACEMENT or PRESCRIBED DISPLACEMENT boundary conditions.

Generally, three BEGIN PRESCRIBED VELOCITY command blocks will be needed, one each for X, Y, and Z components. In order to eliminate rigid body motion without over constraining the motion, each BEGIN PRESCRIBED VELOCITY block should constrain exactly one node of an RVE in one component direction. (However, nodelist_name may contain nodes from multiple RVEs. Separate boundary condition blocks are not required for each RVE.). To prevent rigid body rotations, the three constrained nodes on each RVE should not be collinear.

10.2 J-Integrals

Sierra/SM provides a capability to compute the J-integral via a a domain integral.

Known Issue: Currently, the J-Integral evaluation capability is based on assumptions of elastostatics and a stationary crack, and is only implemented for uniform gradient hex elements.

J is analogous to G from linear elastic fracture mechanics ($-\delta\pi/\delta a$) and is the driving force on the crack tip a [1,2]. Crack propagation occurs when $J \geq R$, where R is the material resistance and is often referred to as the critical energy release rate J_{1c}. In the reference configuration, the vectorial form of the J-integral in finite deformation [4] is

$$\boldsymbol{J} = \int_{\Gamma_0} \boldsymbol{\Sigma} \boldsymbol{N} dA \tag{10.1}$$

where $\boldsymbol{\Sigma} = W\boldsymbol{I} - \boldsymbol{F}^T\boldsymbol{P}$ is referred to as the Eshelby energy-momentum tensor [3]. W is the stored energy density in the reference configuration and $\boldsymbol{F}$ and $\boldsymbol{P}$ are the deformation gradient and first Piola-Kirchhoff stress, respectively. Rice [2] realized that because $\boldsymbol{\Sigma}$ is divergence-free in the absence of body forces, one can examine $\boldsymbol{J}$ in the direction of the defect $\boldsymbol{L}$ (unit vector) and obtain a path-independent integral for traction-free crack faces. J can be written as

$$J = \int_{\Gamma_0} \boldsymbol{L} \cdot \boldsymbol{\Sigma} \boldsymbol{N} dA \tag{10.2}$$

and interpreted as a path-independent driving force in the direction of the defect. We note that one can also express $\boldsymbol{\Sigma}$ in terms of $\bar{\boldsymbol{\Sigma}}$, where $\bar{\boldsymbol{\Sigma}} = W\boldsymbol{I} - \boldsymbol{H}^T\boldsymbol{P}$ and $\boldsymbol{H} = \text{Grad}\,\boldsymbol{u}$. Although $\boldsymbol{\Sigma}$ is symmetric and $\bar{\boldsymbol{\Sigma}}$ is not symmetric, they are equivalent when integrated over the body ($\text{Div}\boldsymbol{P} = \boldsymbol{0}$). In fact, differences in the energy-momentum tensor stem from the functional dependence of the stored energy function W. $\boldsymbol{\Sigma}$ and $\bar{\boldsymbol{\Sigma}}$ derive from $W(\boldsymbol{F})$ and $W(\boldsymbol{H})$, respectively. When integrated, both collapse to the familiar 2-D relation for infinitesimal deformations.

$$J = \int_{\Gamma} \boldsymbol{e}_1 \cdot \boldsymbol{\Sigma} \boldsymbol{n} ds = \int_{\Gamma} (W n_1 - u_{i,1}\sigma_{ij}n_j)\text{d}s \tag{10.3}$$

10.2.1 Technique for Computing J

J is often expressed as a line (2D) or surface (3D) integral on a ring surrounding the crack tip. Defining a smooth ring over which to compute this surface integral and performing projections of the required field values onto that ring presents a number of difficulties in the context of a finite element code.

To compute the J-integral in a finite element code, it is more convenient to perform a volume integral over a domain surrounding the crack tip. We can then leverage the information at integration points rather than rely on less accurate projections. To do this, we follow the method described in

[5]. We replace $\boldsymbol{L}$ with a smooth function $\boldsymbol{q}$. On the inner contour of the domain Γ_0, $\boldsymbol{q} = \boldsymbol{L}$. On the outer contour of the domain C_0, $\boldsymbol{q} = \boldsymbol{0}$. Because the outer normal of the domain $\boldsymbol{M}$ is equal and opposite of the normal $\boldsymbol{N}$ on Γ_0, there is a change of sign. For traction-free surfaces, we can apply the divergence theorem, enforce $\text{Div}\boldsymbol{\Sigma} = \boldsymbol{0}$, and find that the energy per unit length $\bar{J}$ is

$$\bar{J} = -\int_{\Omega_0} (\boldsymbol{\Sigma} : \text{Grad}\, \boldsymbol{q}) \text{d}V. \tag{10.4}$$

We note that the all the field quantities are given via simulation and we choose to define $\boldsymbol{q}$ on the nodes of the domain $\boldsymbol{q}^I$ and employ the standard finite element shape functions to calculate the gradient. We can specify the crack direction $\boldsymbol{L}$ or assume that the crack will propagate in the direction normal to the crack front $-\boldsymbol{M}$. For a "straight" crack front, $\boldsymbol{L} = -\boldsymbol{M}$. If $\boldsymbol{S}$ is is tangent to the crack front and $\boldsymbol{T}$ is normal to the lower crack surface, $\boldsymbol{S} \times \boldsymbol{M} = \boldsymbol{T}$. We note that for non-planar, curving cracks, $\boldsymbol{M}$, $\boldsymbol{S}$, and $\boldsymbol{T}$ are functions of the arc length S. For ease, we employ the notation $\boldsymbol{N}$ rather than $-\boldsymbol{M}$. For a crack front S_0, we can define the average driving force J_{avg} as

$$J_{avg} = \frac{\bar{J}}{\int_{S_0} \boldsymbol{L} \cdot \boldsymbol{N} \text{d}S}. \tag{10.5}$$

While the average driving force is useful for interpreting experimental findings and obtaining a macroscopic representation of the driving force, we also seek to examine the local driving force $J(S)$. Using the finite element interpolation functions to discretize $\boldsymbol{L}$ through the smooth function $\boldsymbol{q}$, we find $\boldsymbol{q} = \lambda^I \boldsymbol{q}^I$. For a specific node K, we can define $|\boldsymbol{q}^K| = 1$ and $\boldsymbol{q}^I = \boldsymbol{0}$ for all other $I \neq K$ on S_0. Note that we still need to specify the function $\boldsymbol{q}$ in the $\boldsymbol{S} - \boldsymbol{T}$ plane from the inner contour Γ_0 to the outer contour C_0. The resulting expression for the approximate, pointwise driving force at node K on the crack front is

$$J^K = \frac{\bar{J}}{\int_{S_0} \lambda^K \boldsymbol{q}^K \cdot \boldsymbol{N} \text{d}S}. \tag{10.6}$$

Again, we note that if the direction of propagation $\boldsymbol{L}$ is taken in the direction of the normal $\boldsymbol{N}$, the denominator is $\int_{S_0} \lambda^K \text{d}S$. More information regarding the pointwise approximation of J^K can be found in [6,7].

10.2.2 Input Commands

A user can request that J-integrals be computed during the analysis by including one or more `J INTEGRAL` command blocks in the `REGION` scope. This block can contain the following commands:

```
BEGIN J INTEGRAL <jint_name>
  #
  # integral parameter specification commands
  CRACK DIRECTION = <real>dir_x <real>dir_y <real>dir_z
  CRACK PLANE SIDE SET = <string list>side_sets
  CRACK TIP NODE SET = <string list>node_sets
  INTEGRATION RADIUS = <real>int_radius
```

```
    NUMBER OF DOMAINS = <integer>num_domains
    FUNCTION = PLATEAU|PLATEAU_RAMP|LINEAR(PLATEAU)
    SYMMETRY = OFF|ON(OFF)
    DEBUG OUTPUT = OFF|ON(OFF) WITH <integer>num_nodes
      NODES ON THE CRACK FRONT
    #
    # time period selection commands
    ACTIVE PERIODS = <string list>period_names
    INACTIVE PERIODS = <string list>period_names
  END J INTEGRAL <jint_name>
```

A set of parameters must be provided to define the crack geometry and the integration domains used in the calculation of the *J*-integral. The model must be set up so that there is a sideset on one surface of the crack plane behind the crack tip and a nodeset containing the nodes on the crack tip. Both the `CRACK PLANE SIDE SET` and `CRACK TIP NODE SET` commands must be used to specify the names of the sideset behind the crack tip and the nodeset on the crack tip, respectively.

By default, the direction of crack propagation is computed from the geometry of the crack plane and tip as provided in the crack plane sideset and crack tip nodeset ($\boldsymbol{L} = \boldsymbol{N}$). The `CRACK DIRECTION` command can optionally be used to override the direction of crack propagation ($\boldsymbol{L}$) computed from the geometry. This command takes three real numbers that define the three components of the crack direction vector as arguments.

To fully define the domains used for the domain integrals, the radius of the domains and the number of domains must also be specified. A series of disc-shaped integration domains are formed with varying radii going out from the crack tip. The `INTEGRATION RADIUS` command is used to specify the radius of the outermost domain. The number of integration domains is specified using the `NUMBER OF DOMAINS` command. The radii of the domains increase linearly going from the innermost to the outermost domain.

The weight function $\boldsymbol{q}$ used to calculate the *J*-integral is specified by use of the option `FUNCTION` command line. The `LINEAR` function set the weight function to 1.0 on the crack front Γ_0 and 0.0 at the edge of the domain C_0, `int_radius` away from the crack tip. The `PLATEAU` function, which is the default behavior, sets all values of the weight function to 1.0 that lie within the domain of integration and all values outside of the domain are set to 0.0. This allows for integration over a single ring of elements at the edge of the domain. The third option for the `FUNCTION` command is `PLATEAU_RAMP`, which for a single domain will take on the same values as the `LINEAR` function. However, when there are multiple domains over the radius `int_radius`, the nth domain will have weight function values of 1.0 over the inner (n-1) domains and will vary from 1.0 to 0.0 over the outer n^{th} ring of the domain. These functions can be seen graphically in Figure 10.2.

We note that in employing both the `PLATEAU` and the `PLATEAU_RAMP` functions, one is effectively taking a line integral at finite radius (albeit different radii). In contrast, the `LINEAR` option can be viewed as taking the $lim\ \Gamma_0 \rightarrow 0^+$. If the model is a half symmetry model with the symmetry plane on the plane of the crack, the optional `SYMMETRY` command can be used to specify that the symmetry conditions be accounted for in the formation of the integration domains and in the evaluation of the integral. The default behavior is for symmetry to not be used.

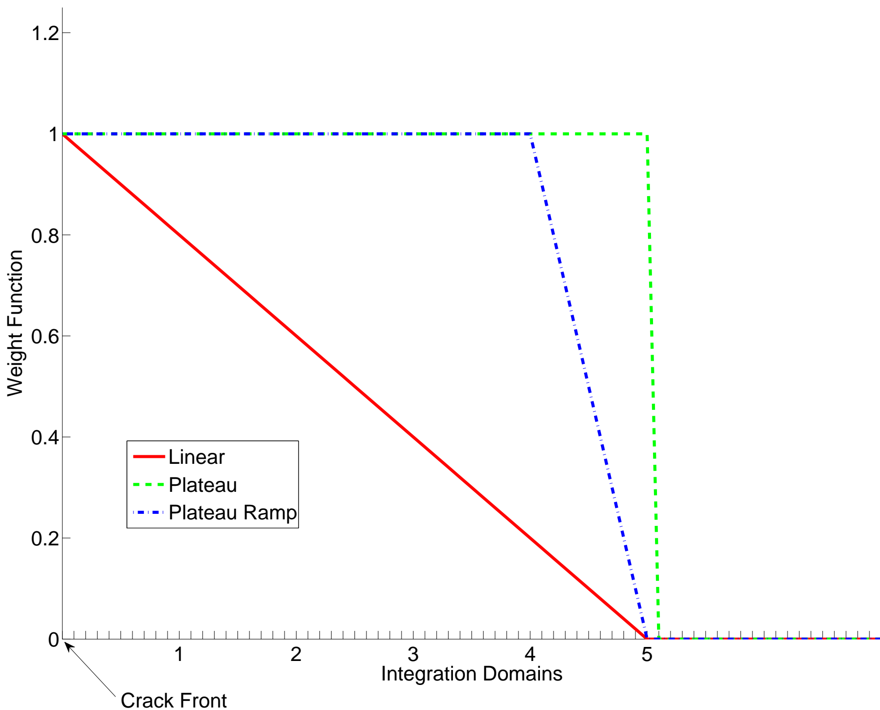


Figure 10.2: Example weight functions for a J-integral integration domain. Weight functions shown for domain 5.

The user may optionally specify the time periods during which the J-integral is computed. The `ACTIVE PERIODS` and `INACTIVE PERIODS` command lines are used for this purpose. See Section 2.5 for more information about these command lines.

10.2.3 Output

A number of variables are generated for output when the computation of the J-integral is requested. The average value of J for each integration domain is available as a global variables, as described in Table 9.3. The pointwise value of J at nodes along the crack for each integration domain is available as a nodal variable, as shown in Table 9.9. Element variables such as the Eshelby energy-momentum tensor and fields defining the integration domains are also available, as listed in Table 9.18.

The `DEBUG OUTPUT` command can be used to generate output data for debugging the J-integral. If the `DEBUG OUTPUT = ON|OFF(OFF) WITH <integer>num_nodes NODES ON THE CRACK FRONT` line command is set to `ON`, the weight functions, q, will be output for each node-based J value that is calculated. The user must specify `num_nodes`, which represents the number of nodes along the crack front. An internal check is performed during problem initialization that will verify that the number of nodes specified by the user on the command line matches the number of nodes associated with the crack front.

Warning: Using the `DEBUG OUTPUT` command line can potentially result in an extremely large output file due to the fact that every node in the integration domain will now compute and store $(NumNodeOnCrackFront + 1) * NumDomains$ weight function vectors. This can also potentially exhaust the available memory on the machine.

10.3 Peridynamics

Peridynamics is an extension of classical solid mechanics that allows for the modeling of bodies in which discontinuities occur spontaneously [8, 9, 10, 11, 12]. Unlike the classical continuum formulation, the peridynamic expression for the balance of linear momentum does not contain spatial derivatives and instead is based on an integral equation. For this reason, peridynamics is well suited for modeling phenomena involving spatial discontinuities such as cracking. This section presents a basic overview of the peridynamics formulation implemented in Sierra/SM and suggested guidelines for setting up peridynamic analyses. For a comprehensive review of peridynamics theory, the reader is referred to [12]

10.3.1 Overview

Peridynamics is a nonlocal theory in which any point in the body $\mathbf{x}$ is acted on by forces due to the deformation of all points $\mathbf{x}'$ within a neighborhood of radius δ centered at $\mathbf{x}$. The radius δ is referred to as the horizon and the vector $\mathbf{x}' - \mathbf{x}$ is called a bond.

The peridynamic equation of motion is

$$\begin{aligned} \rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) &= \mathbf{L}_{\mathbf{u}}(\mathbf{x},t) + \mathbf{b}(\mathbf{x},t) \qquad \forall\, \mathbf{x} \in \mathscr{B}, \quad t \geq 0, \\ \mathbf{L}_{\mathbf{u}}(\mathbf{x},t) &= \int_{\mathscr{B}} \left\{ \underline{\mathbf{T}}[\mathbf{x},t]\langle \mathbf{x}' - \mathbf{x}\rangle - \underline{\mathbf{T}}[\mathbf{x}',t]\langle \mathbf{x} - \mathbf{x}'\rangle \right\} \, \mathrm{d}V_{\mathbf{x}'}. \end{aligned} \tag{10.7}$$

Here, $\mathscr{B}$ is the reference configuration of the body, ρ is the density in the reference configuration, $\mathbf{u}$ is the displacement, and $\mathbf{b}$ is the body force density.

The relationship between $\mathbf{x}$ and $\mathbf{x}'$ is expressed in terms of the *force state* at $\mathbf{x}$ at time t, $\underline{\mathbf{T}}[\mathbf{x},t]$. The force state is a function that associates with any bond $\mathbf{x}' - \mathbf{x}$ a force density per unit volume $\underline{\mathbf{T}}[\mathbf{x},t]\langle \mathbf{x}' - \mathbf{x}\rangle$ acting on $\mathbf{x}$. This force density per unit volume is referred to as a *pairwise force*.

The body $\mathscr{B}$ may be discretized in the reference configuration into a finite number of cells with each cell containing a single node at its center. The integral in Equation (10.7) may then be replaced by a summation,

$$\rho(\mathbf{x})\ddot{\mathbf{u}}_h(\mathbf{x},t) = \sum_{i=0}^{N} \left\{ \underline{\mathbf{T}}[\mathbf{x},t]\left\langle \mathbf{x}'_i - \mathbf{x}\right\rangle - \underline{\mathbf{T}}'[\mathbf{x}'_i,t]\left\langle \mathbf{x} - \mathbf{x}'_i\right\rangle \right\} \Delta V_{\mathbf{x}'_i} + \mathbf{b}(\mathbf{x},t), \tag{10.8}$$

where N is the number of cells in the neighborhood of $\mathbf{x}$, $\mathbf{x}'_i$ is the position of the node centered in cell i, and $\Delta V_{\mathbf{x}'_i}$ is the volume of cell i. The length of the bond between two cells is determined as the distance between the nodes at the centers of the cells.

Equation (10.8) requires evaluation of the pairwise forces $\underline{\mathbf{T}}[\mathbf{x},t]\left\langle \mathbf{x}'_i - \mathbf{x}\right\rangle$ and $\underline{\mathbf{T}}'[\mathbf{x}'_i,t]\left\langle \mathbf{x} - \mathbf{x}'_i\right\rangle$, where the force states $\underline{\mathbf{T}}[\mathbf{x},t]$ and $\underline{\mathbf{T}}'[\mathbf{x}'_i,t]$ are, in general, functions of the deformations of all cells within the neighborhoods of $\mathbf{x}$ and $\mathbf{x}'_i$, respectively. A constitutive model is required to compute the

force state, $\underline{\mathbf{T}}[\mathbf{x}, t]$, in terms of the deformation state in the neighborhood of $\mathbf{x}$ and possibly other variables as well.

The constitutive models available for peridynamics in Sierra/SM fall into two categories:

1. Constitutive models that are specific to peridynamics and make use of the complete deformation state when computing internal forces.
2. Classical constitutive models within the LAME library that have been made available from within peridynamics via a wrapper.

Constitutive models developed specifically for peridynamics are generally more robust than classical constitutive models that have been adapted for use within peridynamics, particularly in analyses involving extreme deformation and pervasive fracture. The *Linear Peridynamic Solid* material model developed by Silling, et al. [10], described below, is the only model of the first type currently available in Sierra/SM. The peridynamics interface to classical material models offers a means for accessing the large number of material models within LAME. This interface is based on the computation of an approximate deformation gradient, $\bar{\mathbf{F}}$, described in Section 10.3.3.

10.3.2 Linear Peridynamic Solid Material Model

The linear peridynamic solid material model computes the pairwise forces $\underline{\mathbf{T}}[\mathbf{x}, t]\left\langle \mathbf{x}'_i - \mathbf{x} \right\rangle$ acting on $\mathbf{x}$ based on the deformation of all cells $\mathbf{x}'$ within the neighborhood of $\mathbf{x}$,

$$\underline{\mathbf{T}}[\mathbf{x}, t]\left\langle \mathbf{x}'_i - \mathbf{x} \right\rangle = \underline{t}\ \underline{\mathbf{M}}[\mathbf{x}, t]\left\langle \mathbf{x}'_i - \mathbf{x} \right\rangle, \tag{10.9}$$

where $\underline{t}$ is the magnitude of the pairwise force and $\underline{\mathbf{M}}[\mathbf{x}, t]\left\langle \mathbf{x}'_i - \mathbf{x} \right\rangle$ is the unit vector pointing from the deformed position of $\mathbf{x}$ to the deformed position of $\mathbf{x}'$.

The magnitude of the pairwise force in a linear peridynamic solid is given by

$$\underline{t} = \frac{-3p}{m}\underline{\omega}\,\underline{x} + \frac{15\mu}{m}\underline{\omega}\,\underline{e}^d, \qquad p = -k\theta, \tag{10.10}$$

where the p is the peridynamic pressure, θ is the peridynamic dilatation, m is the weighted volume at $\mathbf{x}$, $\underline{\omega}$ is the value of the influence function between cells $\mathbf{x}$ and $\mathbf{x}'$, $\underline{x}$ is the length of the bond $\mathbf{x}' - \mathbf{x}$ in the undeformed configuration, $\underline{e}^d$ is the deviatoric part of the extension $\underline{e}$, and μ and k are material constants (the shear modulus and bulk modulus, respectively). For a complete description of the linear peridynamic solid material model, see [10].

10.3.3 Interface to Classical Material Models

A means for adapting classical material models for use with peridynamics has been developed by Silling, et al. [10]. The approach is based on the evaluation of an approximate deformation gradient, $\bar{\mathbf{F}}$, at $\mathbf{x}$,

$$\bar{\mathbf{F}} = \left(\sum_{i=0}^{N} \underline{\omega}_i\, \underline{\mathbf{Y}}_i \otimes \underline{\mathbf{X}}_i\, \Delta V_{\mathbf{x}_i} \right) \mathbf{K}^{-1}, \tag{10.11}$$

where $\mathbf{K}$ is the *shape tensor*, defined as

$$\mathbf{K} = \sum_{i=0}^{N} \underline{\omega}_i \, \underline{\mathbf{X}}_i \otimes \underline{\mathbf{X}}_i \, \Delta V_{\mathbf{x}_i}. \tag{10.12}$$

Here, $\mathbf{X}$ denotes a vector directed from $\mathbf{x}$ to $\mathbf{x}'$ in the reference configuration, $\mathbf{Y}$ denotes a vector directed from $\mathbf{x}$ to $\mathbf{x}'$ in the deformed configuration, $\underline{\omega}$ is the value of the influence function between cells $\mathbf{x}$ and $\mathbf{x}'$, and $\Delta V_{\mathbf{x}}$ is the volume of cell $\mathbf{x}$. The operator $\otimes$ denotes a dyadic product.

The approximate deformation gradient, as defined in Equation (10.11), allows for the computation of a strain measure or, alternatively, a strain increment, that can be passed to a classical material model. The classical material model is expected to return stress which is then transformed into a pairwise force as follows,

$$\underline{\mathbf{T}} \left\langle \mathbf{x}' - \mathbf{x} \right\rangle = \underline{\omega} \, \boldsymbol{\sigma} \mathbf{K}^{-1} \left\langle \mathbf{x}' - \mathbf{x} \right\rangle, \tag{10.13}$$

where $\boldsymbol{\sigma}$ is the Piola stress.

The accuracy of the peridynamic interface to classical material models depends strongly on the calculation of the approximate deformation gradient, $\bar{\mathbf{F}}$. The approximate deformation gradient calculation may become unreliable in cases of extreme deformation, or in cases of pervasive material failure in which a large percentage of bonds at a given point are broken. This may result in nonphysical strain values being passed to the classical material model, and a subsequent nonphysical internal force calculation. In addition, the inversion of the shape tensor in Equation (10.13) will fail if the number of bonds for a given cell is less than three, or if all bonds for a given cell are coplanar. For these reasons, Sierra/SM sets the stress to zero for peridynamic cells with more than 85% of their bonds broken, or for cells with less than three intact bonds that are not coplanar. Under these conditions, a cell will remain active in the simulation and may be acted on by other cells but will not contribute to the internal force. Cells for which all bonds have been broken will behave as mass particles.

Zero-energy modes are possible in the calculation of the approximate deformation gradient, $\bar{\mathbf{F}}$. Zero energy modes may result in nonphysical motion of cells within a simulation, typically manifesting as rapid oscillations. Sierra/SM provides a means for suppressing zero-energy modes through peridynamic hourglass control. Peridynamic hourglass control is based on the comparison of cell positions with the positions of those cells as predicted by direct application of the approximate deformation gradient computed at a cell $\mathbf{x}$. The suppression of zero-energy modes is achieved by applying a penalty force proportional to the difference between these values.

10.3.4 Modeling Fracture

Peridynamics includes a natural mechanism for modeling fracture through the breaking of peridynamic bonds. In an undamaged material, bonds connect a given cell $\mathbf{x}$ to each cell $\mathbf{x}'$ within its neighborhood in the reference configuration. A damage law dictates the conditions under which individual bonds are broken. Material discontinuities, such as cracks, form as a result of the accumulation of broken bonds.

The critical stretch damage model has been implemented in Sierra/SM to allow for the modeling of fracture within a peridynamics simulation. The critical stretch model assigns a damage value of

zero (unbroken) or one (broken) as a function of the maximum stretch obtained by a bond and a critical stretch material parameter, s_{crit}. The damage value, ϕ, is given by the relation

$$\begin{aligned} \phi &= 0 \qquad s < s_{\text{crit}} \\ \phi &= 1 \qquad s \geq s_{\text{crit}}, \end{aligned} \tag{10.14}$$

where s is the stretch,

$$s = \frac{y_{\text{max}} - x}{x}. \tag{10.15}$$

Here, y_{max} is the maximum distance between the bonded cells $\mathbf{x}$ and $\mathbf{x}'$ achieved over the duration of the simulation, and x is the distances between $\mathbf{x}$ and $\mathbf{x}'$ in the reference configuration. The use of the maximum distance, y_{max}, ensures that the breaking of bonds is irreversible.

Sierra/SM also allows for the breaking of peridynamic bonds based on the value of an element variable. A common use case for this option is the breaking of bonds based on the value of a material model variable, such as equivalent plastic strain. In the case where a bond connects two peridynamic cells for which the specified element variable is defined, the bond is broken when the average value of the element variable exceeds the given threshold. In the case where the element variable is defined on only one of the cells, the single available value is compared against the threshold value.

10.3.5 Peridynamics and Contact

Peridynamics elements are compatible with the Dash contact algorithm as described in Chapter 8. The Dash contact algorithm allows for general and self contact for peridynamics blocks as well as general contact between peridynamics blocks and blocks modeled with classical finite elements. Contact for peridynamics is enabled by the use of lofted contact geometry, which provides the planar facets required by the contact algorithm (see Section 8.5.1.2). In the case of self contact, the contact algorithm has been specialized for peridynamics. Self contact is enabled between peridynamic elements that are not bonded to each other, but is disabled between any pair of elements that is bonded. In this way, peridynamic elements may interact via their constitutive laws, or via contact, but not via both their constitutive law and contact simultaneously. Self contact between a pair of bonded elements is enabled in the event that the bond is broken due to material damage.

10.3.6 Usage Guidelines

This section provides basic usage guidelines for carrying out peridynamic analyses with Sierra/SM. Suggestions are provided for constructing input mesh files, applying boundary conditions, and selecting appropriate values for peridynamic parameters. For a description of the `PERIDYNAMICS SECTION` input syntax, see Section 6.2.12.1.

Model generation. A set of mesh generation tools is available for the construction of input meshes compatible with the peridynamics capabilities of Sierra/SM. The suggested workflow for creating a peridynamics-compatible input mesh is as follows:

1. Construct a hexahedral mesh containing the peridynamics blocks using Cubit.

2. Convert the elements in the resulting mesh file from hexahedron elements to sphere elements using the SEACAS tool `sphgen3d`.

3. Convert the Exodus I database produced by `sphgen3d` to the Exodus II format with the SEACAS tool `ex1ex2v2`.

4. In the case of analyses using both peridynamics and classical finite element analysis, or in the case where bond cutting blocks are used to control the creation of peridynamic bonds, combine an input mesh containing standard finite elements with an input mesh discretized with sphere elements using the SEACAS tool `gjoin`.

Warning: Nodesets defined on a hexahedral mesh are lost when the mesh is converted to sphere elements with `sphgen3d`. In addition to eliminating the nodesets defined in the original hexahedral mesh file, `sphgen3d` automatically creates a new nodeset corresponding to each block in the model. Name conflicts will occur if the resulting sphere-element mesh is combined with a standard finite element mesh containing nodesets with names identical to those created automatically by `sphgen3d`.

Horizon. The horizon defines the region of nonlocality for a given cell $\mathbf{x}$ in a peridynamics analysis. Analysis results may be strongly dependent on the choice of the horizon. For example, the choice of horizon affects the localization of forces in the neighborhood of a discontinuity, as well as the dispersion of waves traveling through blocks modeled with peridynamics. In addition to affecting material behavior, the choice of horizon has a strong effect on the computational cost of a peridynamic analysis. In extreme cases, a large horizon may render an analysis intractable. A recommend value of the horizon is three times the mesh spacing. When using the `SCALE BY ELEMENT RADIUS` option, this corresponds to a horizon value of approximately five.

Boundary conditions. Boundary conditions for a nonlocal approach such as peridynamics differ fundamentally from boundary conditions in classical mechanics. To capture exactly the material response in the vicinity of a boundary condition, the boundary condition in a nonlocal model must be applied over a volume. For example, displacement boundary conditions applied at the surface of a body should be applied to a volume that reaches into the body a distance equal to the peridynamic horizon. This is not always practical (as with contact), and in many cases satisfactory results can be obtained by applying boundary conditions at the surface only. Users should be aware, however, that nonphysical edge effects will be present if boundary conditions are not applied in a way that is consistent with the nonlocal nature of peridynamics.

The application of boundary conditions for peridynamics analyses in Sierra/SM is also affected by the restrictions placed on nodeset definitions by `sphgen3d`. Nodesets defined on a hexahedral mesh are lost when the mesh is converted to sphere elements by `sphgen3d`. For this reason, boundary conditions must be applied on blocks. This requires careful construction of the initial hexahedral mesh to include blocks that correspond to regions over which boundary conditions will be applied.

Timestep. The timestep computed by peridynamic elements in Sierra/SM is an approximate value that may require modification to ensure a stable analysis. Peridynamic elements in Sierra/SM compute a stable time step under the assumptions of linear material behavior and the approximate equivalence of the bulk modulus in *state-based* peridynamics with the micromodulus in *bond-based* peridynamics. For details regarding the timestep computed by peridynamic elements in Sierra/SM, see [9]. Because the timestep computed by peridynamic elements is not guaranteed to be stable, it is recommended that the timestep be reduced using a scale factor of approximately 0.75. This can be achieved using the `TIME STEP SCALE FACTOR` command described in Chapter 2.7.

Hourglass stiffness. The use of classical material models with peridynamics in Sierra/SM may require the application of hourglass stiffness to reduce the effects of zero-energy modes in the computation of the approximate deformation gradient, $\bar{\mathbf{F}}$. Recommended values for hourglass stiffness are in the range 0.01 - 0.05. The value of hourglass stiffness is multiplied by a material's bulk modulus when computing the stiffening forces for peridynamics hourglass control. For a description of the corresponding input syntax, see Section 6.2.12.1.

10.4 References

1. Eshelby, J. D., “The Force on an Elastic Singularity.” *Philosophical Transactions of the Royal Society of London* A244(1951): 87–112. doi.

2. Rice, J. R., “A Path Independent Integral and the Approximate Analysis of Stress Concentration by Notches and Cracks.” *Journal of Applied Mechanics* 35(1968): 379-386.

3. Eshelby, J. D., “Energy Relations and the Energy-Momentum Tensor in Continuum Mechanics.” *Inelastic Behavior of Solids* New York: McGraw-Hill, 1970.

4. Maugin, G. A., *Material Inhomogeneities in Elasticity* New York: Chapman & Hall/CRC, 1993.

5. Li, F. Z., C. F. Shih, and A. Needleman. “A Comparison of Methods for Calculating Energy Release Rates.” *Engineering Fracture Mechanics* 21(1985): 405–421. doi.

6. Shih, C. F., B. Moran, and T. Nakamura. “Energy release rate along a three-dimensional crack front in a thermally stressed body.” *International Journal of Fracture* 30 (1986): 79–102.

7. HKS. *ABAQUS Version 6.7, Theory Manual*. Providence, RI: Hibbitt, Karlsson and Sorensen, 2007.

8. Silling, S. A., “Reformulation of elasticity theory for discontinuities and long-range forces.” *Journal of the Mechanics and Physics of Solids* 42(2000): 175–209. doi.

9. Silling, S. A., and Askari, E. “A meshfree method based on the peridynamic model of solid mechanics.” *Computers and Structures* 83(2005): 1526–1535. doi.

10. Silling, S. A., “Peridynamic States and Constitutive Modeling” *Journal of Elasticity* 88(2007): 151–184. doi.

11. Silling, S. A. and Lehoucq, R. B., “Convergence of peridynamics to classical elasticity theory” *Journal of Elasticity* 93(2008): 13–37. doi.

12. Silling, S. A. and Lehoucq, R. B., “Peridynamic Theory of Solid Mechanics” *Advances in Applied Mechanics* 44(2010): 73–168. doi. Also available as SAND2010-1233J. pdf.

Chapter 11

User Subroutines

User-defined subroutines is a functionality shared by Adagio and Presto. This chapter discusses when and how to use user-defined subroutines. There are examples of user-defined subroutines in the latter part of this chapter. Some of the examples are code specific, i.e., they are applicable to Presto rather than Adagio or vice versa. All examples, regardless of their applicability, do provide important information about how to use the command options available for user-defined subroutines.

In the introductory part of Chapter 11, we first describe, in general, possible applications for the user subroutine functionality in Sierra/SM. Then, again in general, we describe the various pieces and steps that are required by the user to implement a user subroutine. Subsequently, we focus on various aspects of implementing the user subroutine functionality. Section 11.1 describes the details of the user subroutine. Section 11.2 describes the command lines associated with user subroutines that appear in the Sierra/SM input file. In Section 11.3, we explain how to build and use a version of Sierra/SM that incorporates your user subroutine. Finally, Section 11.4 provides examples of actual user subroutines, and Section 11.5 lists some subroutines that are now in the standard user library.

Applications. User subroutines are primarily intended as complex function evaluators that are to be used in conjunction with existing Sierra/SM capability (boundary conditions, element death (only for explicit dynamics), user output, etc.). For example, suppose we want to have a prescribed displacement boundary condition applied to a set of nodes, and we want the displacement at each node to vary with both time and spatial location of the node. The standard function option associated with the prescribed direction displacement boundary condition in Sierra/SM only allows for time variation; i.e., at any given time, the direction and the magnitude of the displacement at each node, regardless of the spatial location of the node, are the same. If we wanted to have a spatial variation of the displacement field in addition to the time variation, it would be necessary to implement a user subroutine for the prescribed direction displacement boundary condition. Other examples of possible uses of user subroutines are as follows:

- Element death is determined by a complex function based on a set of physical parameters and element stress.
- The user wants to compute the total contact force acting on a given surface.

- Element stress information must be transformed to a local coordinate system so that the stress values will be meaningful.
- An aerodynamic pressure based on velocity and surface normal is applied to a specified surface.

Some capability exists for using mesh connectivity. It is possible to compute an element quantity based on values at the element nodes.

Some difficulties might occur in parallel applications. If computations for element A depend on quantities in element B and elements A and B are on different processors, then the computations for A may not have access to quantities in element B. For most computations in user subroutines, however, this should not be a problem.

Implementing completely new capabilities, particularly if these capabilities involve parallel computing, may be difficult or impossible with user subroutines.

General Pieces and Steps. A number of pieces and steps are required to make use of user subroutines. Here, we present a brief description of the pieces and steps that a user will need for user subroutines without going into detail. The details are discussed in later parts of this chapter.

1. You must first determine whether your application fits in the user subroutine format. This can be done by considering the above requirements and examining the description of commands for functionality in Sierra/SM. For example, the basic kinematic boundary conditions and force conditions allow for the use of user subroutines. The description of these commands includes a discussion of how a user subroutine could be applied and what command line will invoke a user subroutine.

2. If you determine that your application can make use of the user subroutine functionality in Sierra/SM, you will then need to write the subroutine. The parts of the subroutine that interface to Sierra/SM have specified formats. The details of these interfaces are described in later sections. One part of the subroutine with a specified format is the call list. Other parts of the subroutine with a specified format are code that will do the following:

   - Read parameters from the Sierra/SM input file
   - Access a variety of information—field variables, analysis time, etc.—from Sierra/SM
   - Store computed quantities

   Parameters are values they may be passed from the Sierra/SM input file to the user subroutine. Suppose that the spatial variation for some quantity in the user subroutine uses some characteristic length and the user wishes to examine results generated by using several different values of the characteristic length. By setting up the characteristic length as a parameter, the value for the parameter in the user subroutine can easily be changed by changing the value for the parameter in the input file. This lets the user change the value for a variable inside the user subroutine without having to recompile the user subroutine.

The portion of your subroutine not built on the Sierra/SM specifications will reflect your specific application. The code to implement your application may include a loop over nodes that prescribes a displacement based on the current time for the analysis and the spatial location of the node.

3. After you write the user subroutine, you will need to have a command line in your input file that tells Sierra/SM you want to use the user subroutine you have written. For example, if your user subroutine is a specialized prescribed displacement boundary condition, then inside a `PRESCRIBED DISPLACEMENT` command block, you will have a command line of the form

   ```
   NODE SET SUBROUTINE = <string>subroutine_name
   ```

   that provides the name of your user subroutine.

4. Following the invocation of the user subroutine, there may be command lines for various parameters associated with the user subroutine. There may also be some additional command lines in other sections of the code required for your application. For example, you may have to add command lines in the region scope that will create an internal variable associated with a computed quantity so that the computed quantity can be written to the results file.

5. Once you have constructed the user subroutine, which is a FORTRAN file, and the Sierra/SM input file, you can build an executable version of Sierra/SM that will run your user subroutine. Your Sierra/SM run will then incorporate the functionality you have created in your user subroutine.

Figure 11.1 presents a very high-level overview of the various components that work together to implement the user subroutine functionality. The two main components needed for user subroutines, which are commands in the Sierra/SM input file and the actual user subroutine, are represented by the two columns in Figure 11.1.

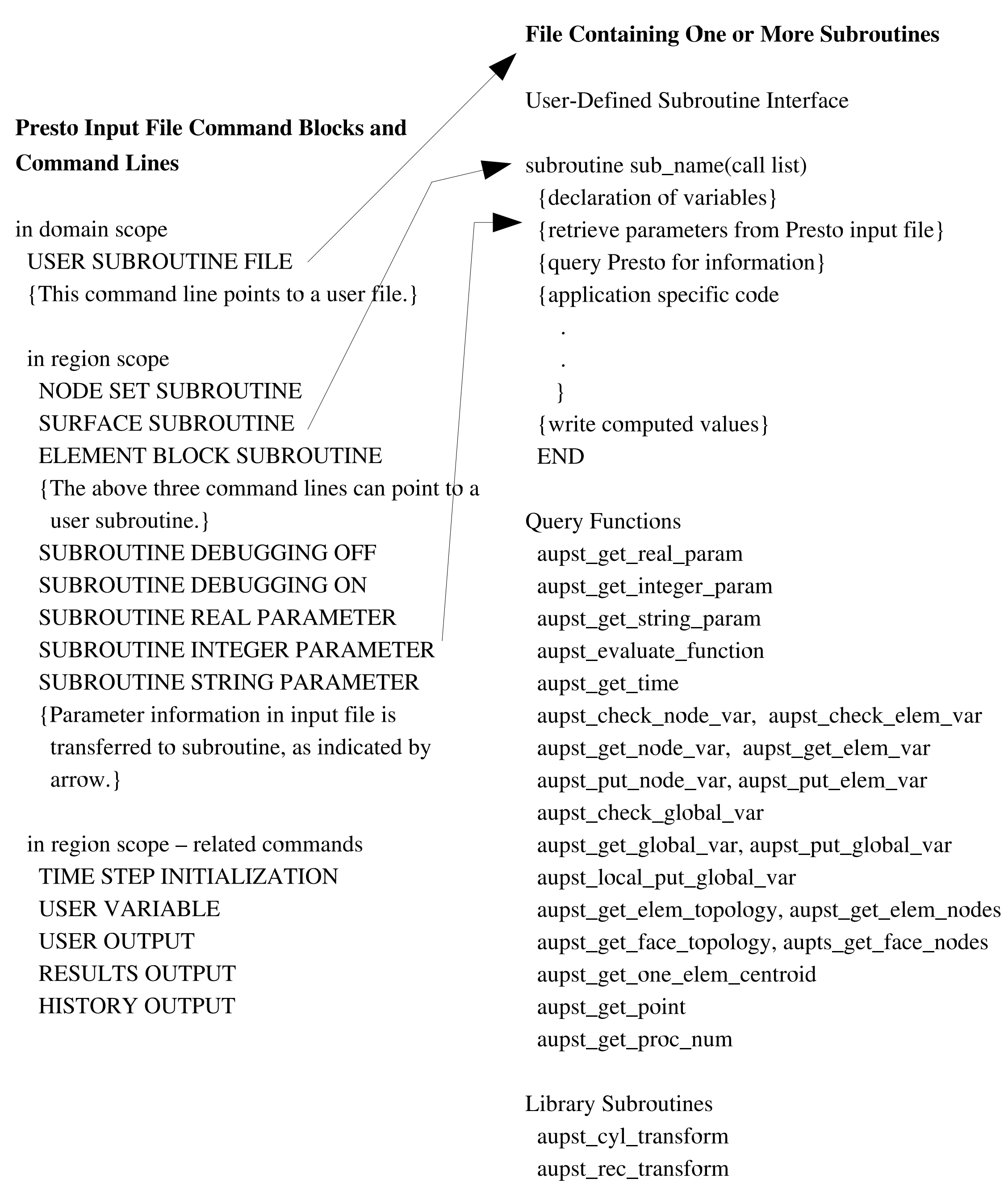


Figure 11.1: Overview of components required to implement user subroutine functionality, excluding compilation and execution commands.

11.1 User Subroutines: Programming

Currently, user subroutines are only supported in FORTRAN 77. Any subroutine that can be compiled with a FORTRAN 77 compiler on the target execution machine can be used. The user should be aware that some computers support different FORTRAN language extensions than others. (In the future, other languages such as FORTRAN 90, C, and C++ may be supported.)

User subroutine variable types must interface directly with the matching variable types used in the main Sierra/SM code. Thus, the FORTRAN 77 subroutines should use only integer, double precision, or character types for any data used in the interface or in any query function. Using the wrong data type may yield unpredictable results. The methods used to pass character types from Sierra/SM to FORTRAN user subroutines can be machine-dependent, but generally this functionality works quite well.

The basic structure for the user subroutine is as follows:

```
subroutine sub_name(call list)
{declaration of variables}
{retrieve parameters from Sierra/SM input file}
{query Sierra/SM for information}
{application-specific code
   .
   .
}
{write computed values}
END
```

In general, the user will begin the subroutine with variable declarations. After the variable declarations, the user can then query the Sierra/SM input file for parameters. Additional Sierra/SM information such as field variables or element topology can then be retrieved from Sierra/SM. Once the user has collected all the information for the application, the application-specific portion of the code can be written. After the application-specific code is complete, the user may store computed values.

Section 11.1.1 through Section 11.1.3 describe in detail the format for the interfaces to Sierra/SM that will allow the user to make the subroutine call, retrieve information from Sierra/SM, and write computed values. In these sections, mesh entities can be a node, an element face, or an element.

11.1.1 Subroutine Interface

The following interface is used for all user subroutines:

```
subroutine sub_name(int  num_objects,
           int  num_values,
           real evaluation_time,
           int  object_ids[],
           real output_values[],
           int  output_flags[],
           int  error_code)
```

The name of the user subroutine, `sub_name`, is selected by the user. Avoid names for the subroutine that are longer than 10 characters. This may cause build problems on some systems.

A detailed description of the input and output parameters is provided in Table 11.1 and Table 11.2.

Table 11.1: Subroutine Input Parameters

Input Parameter	**Data Type**	**Parameter Description**
num_objects	Integer	Number of input mesh entities. For example, if the subroutine is a node set subroutine, this would be the number of nodes on which the subroutine will operate.
num_values	Integer	Number of return values. This is the number of values per mesh entity.
evaluation_time	Real	Time at which the subroutine should be evaluated. This may vary slightly from the current analysis time. For example, in explicit dynamics analyses, velocities are evaluated one-half time step ahead.
object_ids (num_objects)	Integer	Array of mesh-entity identification numbers. The array has a length of num_objects. The input numbers are the global numbers of the input objects. The object identification numbers can be used to query information about a mesh entity.

11.1.2 Query Functions

Sierra/SM follows a design philosophy for user subroutines that a minimal amount of information should be passed through the call list. Additional information may be queried from within the subroutine. A user subroutine may query a wide variety of information from Sierra/SM.

Table 11.2: Subroutine Output Parameters

Output Parameter	Data Type	Parameter Description
output_values (num_values, num_objects)	Integer	Array of output values computed by the subroutine. The number of output values will be either the number of mesh entities or some multiple of the number of mesh entities. For example, if there were six nodes (num_objects equals 6) and one value was to be computed per node, the length of output_values would be 6. Similarly, if there were six nodes (num_objects equals 6) and three values were to be computed for each node (as for acceleration, which has X-, Y-, and Z-components), the length of output_values would be 18.
output_flags (num_objects)	Integer	Array of returned flags for each set of data values. When used, this array will generally have a length of num_objects. The usage of the flags depends on subroutine type; the flags are currently used only for element death and for kinematic boundary conditions. For the kinematic boundary conditions (displacement, velocity, acceleration) a flag of –1 means ignore the constraint, a flag of 0 means set the absolute constraint value, and a flag of 1 means set the constraint with direction and distance.
error_code	Integer	Error code returned by the user subroutine. A value of 0 indicates no errors. Any value other than zero is an error. If the return value is nonzero, Sierra/SM will report the error code and terminate the analysis.

11.1.2.1 Parameter Query

A number of user subroutine parameters may be set as described in Section 11.2.2.3. These subroutine parameters can be obtained from the Sierra/SM input file via the query functions listed below.

```
aupst_get_real_param(string var_name, real var_value,
                     int error_code)

aupst_get_integer_param(string var_name, int var_value,
                        int error_code)

aupst_get_string_param(string var_name, string var_value,
                     int error_code)
```

All three of these subroutine calls are tied to a corresponding "parameter" command line that will appear in the Sierra/SM input file. The parameter command lines are described in Section 11.2.2.3. These command lines are named based on the type of value they store, i.e., `SUBROUTINE REAL PARAMETER`, `SUBROUTINE INTEGER PARAMETER`, and `SUBROUTINE STRING PARAMETER`.

We will use the example of a real parameter to show how the subroutine call works in conjunction with the `SUBROUTINE REAL PARAMETER` command line. Suppose we have a real parameter radius that is set to a value of 2.75 on the `SUBROUTINE REAL PARAMETER` command line:

```
SUBROUTINE REAL PARAMETER: radius = 2.75
```

Also suppose we have a call to `aupst_get_real_parameter` in the user subroutine:

```
call aupst_get_real_parameter("radius",cyl_radius,error_code)
```

In the call to `aupst_get_real_parameter`, we have `var_name` set to `radius` and `var_value` defined as the real FORTRAN variable `cyl_radius`. The call to `aupst_get_real_parameter` will assign the value 2.75 to the FORTRAN variable `cyl_radius`. A similar pattern is followed for integer and string parameters.

The arguments for the parameter-related query functions are described in Table 11.3, Table 11.4, and Table 11.5. The function is repeated prior to each table for easy reference.

```
aupst_get_real_param(string var_name, real var_value,
                     int error_code)
```

Table 11.3: aupst_get_real_param Arguments

Parameter	**Usage**	**Data Type**	**Description**
var_name	Input	String	Name of a real-valued subroutine parameter, as defined in the Sierra/SM input file via the `SUBROUTINE REAL PARAMETER` command line.
var_value	Output	Real	Name of a real variable to be used in the FORTRAN subroutine. The FORTRAN variable var_value will be set to the value specified by the `SUBROUTINE REAL PARAMETER` command line.
error_code	Output	Integer	Error code indicating status of retrieving the parameter value from the input file. If the retrieval is successful, error_code is set to 0. If the parameter is not found or is the wrong type, error_code is set to a value other than 0.

```
aupst_get_integer_param(string var_name, int var_value,
                        int error_code)
```

Table 11.4: aupst_get_integer_param Arguments

Parameter	**Usage**	**Data Type**	**Description**
var_name	Input	String	Name of an integer-valued subroutine parameter, as defined in the Sierra/SM input file via the `SUBROUTINE INTEGER PARAMETER` command line.
var_value	Output	Integer	Name of an integer variable to be used in the FORTRAN subroutine. The FORTRAN variable var_value will be set to the value specified by the `SUBROUTINE INTEGER PARAMETER` command line.
error_code	Output	Integer	Error code indicating status of retrieving the parameter value from the input file. If the retrieval is successful, error_code is set to 0. If the parameter is not found or is the wrong type, error_code is set to a value other than 0.

```
aupst_get_string_param(string var_name, string var_value,
                       int error_code)
```

Table 11.5: aupst_get_string_param Arguments

Parameter	**Usage**	**Data Type**	**Description**
var_name	Input	String	Name of a string-valued subroutine parameter, as defined in the Sierra/SM input file via the `SUBROUTINE STRING PARAMETER` command line.
var_value	Output	String	Name of a string variable to be used in the FORTRAN subroutine. The FORTRAN variable var_value will be set to the value specified by the `SUBROUTINE STRING PARAMETER` command line.
error_code	Output	Integer	Error code indicating status of retrieving the parameter value from the input file. If the retrieval is successful, error_code is set to 0. If the parameter is not found or is the wrong type, error_code is set to a value other than 0.

11.1.2.2 Function Data Query

The function data query routine listed below may be used for extracting data from a function that is defined in a `DEFINITION FOR FUNCTION` command block in the Sierra/SM input file. This query allows the user to directly access information stored in a function defined in the Sierra/SM input file.

```
aupst_evaluate_function(string func_name, real input_times[],
                        int num_times, real output_data[])
```

The arguments for this function are described in Table 11.6.

Table 11.6: aupst_evaluate_function Arguments

Parameter	Usage	Data Type	Description
func_name	Input	String	Name of the function to look up.
input_times (num_times)	Input	Real	Array of times used to extract values of the function.
num_times	Input	Integer	Length of the array input_times.
output_data (num_times)	Output	Real	Array of output values of the named function at the specified times.

11.1.2.3 Time Query

The time query function can be used to determine the current analysis time. This is the time associated with the new time step. This time may not be equivalent to the `evaluation_time` argument passed into the subroutine (see Section 11.1.1, Table 11.1) as some boundary conditions need to be evaluated at different times than others. The parameter of the time query function listed below is given in Table 11.7.

```
aupst_get_time(real time)
```

Table 11.7: aupst_get_time Argument

Parameter	Usage	Data Type	Description
time	Output	Real	Current analysis time.

11.1.2.4 Field Variables

Field variables (displacements, stresses, etc.) may be defined on groups of mesh entities. A number of queries are available for getting and putting field variables. These queries involve passing in a

set of mesh-entity identification numbers to receive field values on the mesh entities. There are query functions to check for the existence and size of a field, functions to retrieve the field values, and functions to store new variables in a field. The field query functions listed below can be used to extract any nodal or element variable field.

```
aupst_check_node_var(int num_nodes, int num_components,
                     int node_ids[], string var_name,
                     int error_code)

aupst_check_elem_var(int num_elems, int num_components,
                     int elem_ids[], string var_name,
                     int error_code)

aupst_get_node_var(int num_nodes, int num_components,
                   int node_ids[], real return_data[],
                   string var_name, int error_code)

aupst_get_elem_var(int num_elems, int num_components,
                   int elem_ids[], real return_data[],
                   string var_name, int error_code)

aupst_get_elem_var_offset(int num_elems, int num_components,
                          int offset, int elem_ids[],
                          real return_data[], string var_name,
                          int error_code)

aupst_put_node_var(int num_nodes, int num_components,
                   int node_ids[], real new_data[],
                   string var_name, int error_code)

aupst_put_elem_var(int num_elems, int num_components,
                   int elem_ids[], real new_data[],
                   string var_name, int error_code)

aupst_put_elem_var_offset(int num_elems, int num_components,
                          int offset, int elem_ids[],
                          real new_data[], string var_name,
                          int error_code)
```

The arrays where data are stored are static arrays. These arrays of a set size will be declared at the beginning of a user subroutine. The query functions to check for the existence and size of a field can be used to ensure that the size of the array of information being returned from Sierra/SM does not exceed the size of the array allocated by the user.

The arguments to field query functions are defined in Table 11.8 through Table 11.15. The function is repeated before each table for easy reference.

```
aupst_check_node_var(int num_nodes, int num_components,
                     int node_ids[], string var_name,
                     int error_code)
```

Table 11.8: aupst_check_node_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_nodes	Input	Integer	Number of nodes used to extract field information.
num_components	Output	Integer	Number of components in the field information. A displacement field at a node has three components, for example.
node_ids (num_nodes)	Input	Integer	Array of size num_nodes listing the node identification number for each node where field information will be retrieved.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_check_elem_var(int num_elems, int num_components,
                     int elem_ids[], string var_name,
                     int error_code)
```

Table 11.9: aupst_check_elem_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements used to extract field information.
num_components	Output	Integer	Number of components in the field information. A stress field for a an eight-node hexahedron element has six components, for example.
elem_ids (num_elems)	Input	Integer	Array of size num_elems listing the element identification number for each element where field information will be retrieved.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_get_node_var(int num_nodes, int num_components,
                   int node_ids[], real return_data[],
                   string var_name, int error_code)
```

Table 11.10: aupst_get_node_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_nodes	Input	Integer	Number of nodes used to extract field information.
num_components	Input	Integer	Number of components in the field information. A displacement field at a node has three components, for example.
node_ids (num_nodes)	Input	Integer	Array of size num_nodes listing the node identification number for each node where field information will be retrieved.
return_data (num_components, num_nodes)	Output	Real	Array of size num_components $\times$ num_nodes containing the field data at each node.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_get_elem_var(int num_elems, int num_components,
                   int elem_ids[], real return_data[],
                   string var_name, int error_code)
```

Table 11.11: aupst_get_elem_var Arguments

Parameter	Usage	Data Type	Description
num_elems	Input	Integer	Number of elements used to extract field information.
num_components	Input	Integer	Number of components in the field information. A stress field for a an eight-node hexahedron element has six components, for example.
elem_ids (num_elems)	Input	Integer	Array of size num_elems listing the element identification number for each element where field information will be retrieved.
return_data (num_components, num_elems)	Output	Real	Array of size num_components $\times$ num_elems containing the field data for each element.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_get_elem_var_offset(int num_elems, int num_components,
                          int offset, int elem_ids[],
                          real return_data[], string var_name,
                          int error_code)
```

Table 11.12: aupst_get_elem_var_offset Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements used to extract field information.
num_components	Input	Integer	Number of components in the field information. A stress field for an eight-node hexahedron element has six components, for example.
offset	Input	Integer	Offset into var_name field variable at which to get data.
elem_ids (num_elems)	Input	Integer	Array of size num_elems listing the element identification number for each element where field information will be retrieved.
return_data (num_components, num_elems)	Output	Real	Array of size num_components × num_elems containing the field data for each element.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_put_node_var(int num_nodes, int num_components,
                   int node_ids[], real new_data[],
                   string var_name, int error_code)
```

Table 11.13: aupst_put_node_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_nodes	Input	Integer	Number of nodes for which the user will specify the field data.
num_components	Input	Integer	Number of components in the field information. A displacement field at a node has three components, for example.
node_ids (num_nodes)	Input	Integer	Array of size num_nodes listing the node identification number for each node where field information will be retrieved.
new_data (num_components, num_nodes)	Input	Real	Array of size num_components × num_nodes containing the new data for the field.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_put_elem_var(int num_elems, int num_components,
                   int elem_ids[], real new_data[],
                   string var_name, int error_code)
```

Table 11.14: aupst_put_elem_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements for which the user will specify the field data.
num_components	Input	Integer	Number of components in the field information. A stress field for a an eight-node hexahedron element has six components, for example.
elem_ids (num_elems)	Input	Integer	Array of size num_elems listing the element identification number for each element where field information will be retrieved.
new_data (num_components, num_elems)	Input	Real	Array of size num_components × num_elems containing the new data for the field.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

```
aupst_put_elem_var_offset(int num_elems, int num_components,
                          int offset, int elem_ids[],
                          real new_data[], string var_name,
                          int error_code)
```

Table 11.15: aupst_put_elem_var_offset Arguments

Parameter	Usage	Data Type	Description
num_elems	Input	Integer	Number of elements for which the user will specify the field data.
num_components	Input	Integer	Number of components in the field information. A stress field for an eight-node hexahedron element has six components, for example.
offset	Input	Integer	Offset into var_name field variable at which to put data.
elem_ids (num_elems)	Input	Integer	Array of size num_elems listing the element identification number for each element where field information will be retrieved.
new_data (num_components, num_elems)	Input	Real	Array of size num_components × num_elems containing the new data for the field.
var_name	Input	String	Name of the field variable. The field variable must be a defined Sierra/SM variable.
error_code	Output	Integer	Error code indicating status of retrieving the field. If the retrieval is successful, error_code is set to 0. If a nonzero value is returned for error_code, the field variable does not exist or is not defined on one or more of the input nodes.

11.1.2.5 Global Variables

Global variables may be extracted or set from user subroutines. A global variable has a single value for a given region.

Global variables have limited support for parallel operations. There are two subroutines to perform parallel modification of global variables: `aupst_put_global_var` and `aupst_local_put_global_var`.

- The subroutine aupst_local_put_global_var only modifies a temporary local copy of the global variable. The local copies on the various processors are reduced to create the true global value at the end of the time step. Global variables set with aupst_local_put_global_var do not have the single processor value available immediately. The true global variable will not be available through the aupst_get_global_var routine until the next time step.

- The subroutine aupst_put_global_var attempts to immediately modify and perform a parallel reduction of the value of a global variable. Care must be taken to call this routine on all processors at the same time with the same arguments. Failure to call the routine from all processors will result in the code hanging. For some types of subroutines this is not possible or reliable. For example, a boundary condition subroutine may not be called at all on a processor that contains no nodes in the set of nodes assigned to the boundary condition. It is recommended that aupst_local_put_global_var only be used in conjunction with a user subroutine referenced in a USER OUTPUT command block (Section 9.2.2).

Only user-defined global variables may be modified by the user subroutine (see Section 11.2.4). However, any global variable that exists on the region may be checked or extracted. The following subroutine calls pertain to global variables:

```
aupst_get_global_var(int num_comp, real return_data,
                     string var_name, int error_code)

aupst_put_global_var(int num_comp, real input_data,
                     string reduction_type,
                     string var_name, int error_code)

aupst_local_put_global_var(int num_comp, real input_data,
                     string var_name, string reduction_type,
                     int error_code)
```

The arguments for subroutine calls pertaining to global variables are defined in Table 11.16 through Table 11.19. The call is repeated before each table for easy reference.

```
aupst_check_global_var(int num_comp, string var_name
                       int error_code)
```

Table 11.16: aupst_check_global_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_comp	Output	Integer	Number of components of the global variable.
var_name	Input	String	Name of the global variable.
error_code	Output	Integer	Error code indicating status of accessing the global variable. If there is no error in accessing this variable, error_code is set to 0. A nonzero value of error_code means the global variable does not exist or in some way cannot be accessed.

```
aupst_get_global_var(int num_comp, real return_data,
                     string var_name, int error_code)
```

Table 11.17: aupst_get_global_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_comp	Input	Integer	Number of components of the global variable.
return_data	Output	Real	Value of the global variable.
var_name	Input	String	Name of the global variable.
error_code	Output	Integer	Error code indicating status of accessing the global variable. If there is no error in accessing this variable, error_code is set to 0. A nonzero value of error_code means the global variable does not exist or in some way cannot be accessed.

```
aupst_put_global_var(int num_comp, real input_data,
                     string reduction_type,
                     string var_name, int error_code)
```

Table 11.18: aupst_put_global_var Arguments

Parameter	Usage	Data Type	Description
num_comp	Input	Integer	Number of components of the global variable.
input_data	Input	Real	New value of the global variable.
reduction_type	Input	String	Type of parallel reduction to perform on the variable. Options are “sum”, “min”, “max”, and “none”.
var_name	Input	String	Name of the global variable.
error_code	Output	Integer	Error code indicating status of accessing the global variable. If there is no error in accessing this variable, error_code is set to 0. A nonzero value of error_code means the global variable does not exist, in some way cannot be accessed, or may not be overwritten.

```
aupst_local_put_global_var(int num_comp, real input_data,
                           string var_name,
                           string reduction_type,
                           int error_code)
```

Table 11.19: aupst_local_put_global_var Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_comp	Input	Integer	Number of components of the global variable.
input_data	Input	Real	New value of the global variable.
reduction_type	Input	String	Type of parallel reduction to perform on the variable. Options are “sum”, “min”, and “max”. The operation type specified here must match the operation type given to the user-defined global variable when it is defined in the Sierra/SM input file.
var_name	Input	String	Name of the global variable.
error_code	Output	Integer	Error code indicating status of accessing the global variable. If there is no error in accessing this variable, error_code is set to 0. A nonzero value of error_code means the global variable does not exist, in some way cannot be accessed, or may not be overwritten.

11.1.2.6 Topology Extraction

The element and surface subroutines operate on groups of elements or faces. The elements and faces may have a variety of topologies. Topology queries can be used to get topological data about elements and faces. The topology of an object is represented by an integer. The integer is formed from a function of the number of dimensions, vertices, and nodes of an object. The topology of an object is given by:

```
topology = num_node + 100 * num_vert + 10000 * num_dim
```

In a FORTRAN routine, the number of nodes can easily be extracted with the mod function:

```
num_node = mod(topo,100)
num_vert = mod(topo / 100, 100)
num_dim  = mod(topo / 10000, 100)
```

Table 11.20: Topologies Used by Sierra/SM

Topology	Element / Face Type
00101	One-node particle
10202	Two-node beam, truss, or damper
20404	Four-node quadrilateral
20303	Three-node triangle
20304	Four-node triangle
20306	Six-node triangle
30404	Four-node tetrahedron
30408	Eight-node tetrahedron
30410	Ten-node tetrahedron
30808	Eight-node hexahedron

Table 11.20 lists the topologies currently in use by Sierra/SM.

The following topology query functions are available in Sierra/SM:

```
aupst_get_elem_topology(int num_elems, int elem_ids[],
                        int topology[], int error_code)

aupst_get_elem_nodes(int num_elems, int elem_ids[],
                     int elem_node_ids[], int error_code)

aupst_get_face_topology(int num_faces, int face_ids[],
                        int topology[], int error_code)

aupst_get_face_nodes(int num_faces, int face_ids[],
                     int face_node_ids[], int error_code)
```

The arguments for the topology extraction functions are defined in Table 11.21 through Table 11.24. The function is repeated before each table for easy reference.

```
aupst_get_elem_topology(int num_elems, int elem_ids[],
                        int topology[], int error_code)
```

Table 11.21: aupst_get_elem_topology Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements from which the topology will be extracted.
elem_ids (num_elems)	Input	Integer	Array of length num_elems listing the element identification for each element from which the topology will be extracted.
topology (num_elems)	Output	Integer	Array of length num_elems that has the topology for each element. See Table 8.18.
error_code	Output	Integer	Error code indicating status of retrieving the element identification numbers. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if one of the element identification numbers is not valid.

```
aupst_get_elem_nodes(int num_elems, int elem_ids[],
                     int elem_node_ids[], int error_code)
```

Table 11.22: aupst_get_elem_nodes Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements from which the topology will be extracted.
elem_ids (num_elems)	Input	Integer	Array of length num_elems listing the element identification for each element from which the topology will be extracted.
elem_node_ids (number of nodes for element type × num_elems)	Output	Integer	Array containing the node identification numbers for each element requested. The length of the array is the total number of nodes contained in all elements. If the elements are eight-node hexahedra, then the number of nodes will be 8 × num_elems. The first set of eight entries in the array will be the eight nodes defining the first element. The second set of eight entries will be the eight nodes defining the second element, and so on.
error_code	Output	Integer	Error code indicating status of retrieving the element identification numbers. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if one of the element identification numbers is not valid.

```
aupst_get_face_topology(int num_faces, int face_ids[],
                        int topology[], int error_code)
```

Table 11.23: aupst_get_face_topology Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_faces	Input	Integer	Number of faces from which the topology will be extracted.
face_ids (num_faces)	Input	Integer	Array of length num_faces listing the face identification for each face from which the topology will be extracted.
topology (num_faces)	Output	Integer	Array of length num_faces containing the output topologies of each face.
error_code	Output	Integer	Error code indicating status of retrieving the face identification numbers. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if one of the face identification numbers is not valid.

```
aupst_get_face_nodes(int num_faces, int face_ids[],
                     int face_node_ids[], int error_code)
```

Table 11.24: aupst_get_face_nodes Arguments

Parameter	Usage	Data Type	Description
num_faces	Input	Integer	Number of faces from which the topology will be extracted.
face_ids (num_faces)	Input	Integer	Array of length num_faces listing the face identification for each face from which the topology will be extracted.
face_node_ids (number of nodes for face type × num_faces)	Output	Integer	Array containing the node identification numbers for each face requested. The length of the array is the total number of nodes contained in all faces. If the faces are four-node quadrilaterals, then the number of nodes will be 4 × num_faces. The first set of four entries in the array will be the four nodes defining the first face. The second set of four entries will be the four nodes defining the second face, and so on.
error_code	Output	Integer	Error code indicating status of retrieving the face identification numbers. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if one of the face identification numbers is not valid.

11.1.3 Miscellaneous Query Functions

A number of miscellaneous query functions are available for computing some commonly used quantities.

```
aupst_get_elem_centroid(int num_elems, int elem_ids[],
                        real centroids, int error_code)

aupst_get_point(string point_name, real point_coords,
                        int error_code)

aupst_get_proc_num(proc_num)
```

The arguments for the miscellaneous query functions are defined in Table 11.25 through Table 11.27. The function is repeated before each table for easy reference.

```
aupst_get_elem_centroid(int num_elems, int elem_ids[],
                        real centroids[], int error_code)
```

Table 11.25: aupst_get_elem_centroid Arguments

Parameter	**Usage**	**Data Type**	**Description**
num_elems	Input	Integer	Number of elements for which to extract the topology.
elem_ids (num_elems)	Input	Integer	Array of length num_elems listing the element identification for each element for which the centroid will be computed.
centroids (3, num_elems)	Output	Real	Array of length 3 × num_elems containing the centroid of each element.
error_code	Output	Integer	Error code indicating status of retrieving the element identification numbers. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if one of the element identification numbers is not valid.

```
aupst_get_point(string point_name, real point_coords,
                     int error_code)
```

Table 11.26: aupst_get_point Arguments

Parameter	**Usage**	**Data Type**	**Description**
point_name	Input	String	SIERRA name for a given point.
point_coords (3)	Output	Real	Array of length 3 containing the x, y, and z coordinates of the point.
error_code	Output	Integer	Error code indicating status of retrieving the point. If the retrieval is successful, error_code is set to 0. A nonzero value is returned for error_code if the point cannot be found

```
aupst_get_proc_num(proc_num)
```

Table 11.27: aupst_get_proc_num Arguments

Parameter	**Usage**	**Data Type**	**Description**
proc_num	Output	Integer	Processor number of the calling process. This number can be used for informational purposes. A common example is that output could only be written by a single processor, e.g., processor 0, rather than by all processors.

11.2 User Subroutines: Command File

In addition to the actual user subroutine, you will need to add command lines to your input file to make use of your user subroutine. This section describes the command lines that are used in conjunction with user subroutines. This section also describes two additional command blocks, `TIME STEP INITIALIZATION` and `USER VARIABLE`. The `TIME STEP INITIALIZATION` command block lets you execute a user subroutine at the beginning of a time step as opposed to some later time. The `USER VARIABLE` command block can be used in conjunction with user subroutines or for user-defined output.

11.2.1 Subroutine Identification

As described in Section 2.1.4, there is one command line associated with the user subroutine functionality that must be provided in the SIERRA scope:

```
USER SUBROUTINE FILE = <string>file_name
```

The named file may contain one or more user subroutines. The file must have an extension of ".F", as in blast.F.

11.2.2 User Subroutine Command Lines

```
{begin command block}
  NODE SET SUBROUTINE = <string>subroutine name |
    SURFACE SUBROUTINE = <string>subroutine_name |
    ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
      = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
      = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
      = <string>param_value
{end command block}
```

A number of user subroutine command lines will appear in some Sierra/SM command block. User subroutine commands can appear in boundary condition, element death (only for explicit dynamics), user output, and state initialization command blocks. The possible command lines are shown above. The following sections describe the command lines related to user subroutines.

11.2.2.1 Type

User subroutines are currently available in three general types: node set, surface, and element.

Node set subroutines operate on groups of nodes. The command line for defining a node set subroutine is:

```
NODE SET SUBROUTINE = <string>subroutine_name
```

where `subroutine_name` is the name of the user subroutine. The name is case sensitive. A node set subroutine will operate on all nodes contained in an associated mechanics instance.

Surface subroutines work on groups of surfaces. A surface may be an external face of a solid element or the face of a shell element associated with either the positive or negative normal for the surface of the shell. The command line for defining a surface subroutine is:

```
SURFACE SUBROUTINE = <string>subroutine_name
```

Element block subroutines work on groups of elements. The command line for defining an element block subroutine is:

```
ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
```

An element may be a solid element such as a hexahedron or a two-dimensional element such as a shell.

Different Sierra/SM features may accept one or more types of user subroutines. Only one subroutine is allowed per command block.

11.2.2.2 Debugging

Subroutines may be run in a special debugging mode to help catch memory errors. For example, there is a potential for a user subroutine to write outside of its allotted data space by writing beyond the bounds of an input or output array. Generally, this causes Sierra/SM to crash, but it also has the potential to introduce other very hard-to-trace bugs into the Sierra/SM analysis. Subroutines run in debug mode require more memory and more processing time than subroutines not run in debug mode.

Subroutine debugging is on by default in debug executables. It can be turned off with the following command line:

```
SUBROUTINE DEBUGGING OFF
```

Subroutine debugging is off by default in optimized executables. It can be turned on with the following command line:

```
SUBROUTINE DEBUGGING ON
```

11.2.2.3 Parameters

All user subroutines have the ability to use parameters. Parameters are defined in the input file and are quickly accessible by the user subroutine during run time. Parameters are a way of making a single user subroutine much more versatile. For example, a user subroutine could be written to define a periodic loading on a structure. A parameter for the subroutine could be defined specifying the frequency of the function. In this way, the same subroutine can be used in different parts of

the model, and the subroutine behavior can be modified without recompiling the program. These command lines are placed within the scope of the command block in which the user subroutine is specified.

Real-valued parameters can be stored with the following command line:

```
SUBROUTINE REAL PARAMETER: <string>param_name
      = <real>param_value
```

Integer-valued parameters can be stored with the following command line:

```
SUBROUTINE INTEGER PARAMETER: <string>param_name
      = <integer>param_value
```

String-valued parameters can be stored with the following command line:

```
SUBROUTINE STRING PARAMETER: <string>param_name
      = <string>param_value
```

Any number of subroutine parameters may be defined. The subroutine parameters may be defined in any order within the command block. The user subroutine may request the values of the parameters but is not required to use them or even have any knowledge of their existence. An example of subroutine usage with parameters is as follows:

```
BEGIN PRESSURE
  SURFACE = surface_1
  SURFACE SUBROUTINE = blast_pressure
  SUBROUTINE REAL PARAMETER: blast_time = 1.2
  SUBROUTINE REAL PARAMETER: blast_power = 1.3e+07
  SUBROUTINE STRING PARAMETER: formulation = alpha
  SUBROUTINE INTEGER PARAMETER: decay_exponent = 2
  SUBROUTINE DEBUGGING ON
END PRESSURE
```

In the above example, four parameters are associated with the subroutine `blast_pressure`. Two of the parameters are real (`blast_time` and `blast_power`), one of the parameters is a string (`formulation`), and one of the parameters is an integer (`decay_exponent`). To access the parameters in the user subroutine, the user will need to include interface calls described in previous sections.

11.2.3 Time Step Initialization

```
BEGIN TIME STEP INITIALIZATION
  # mesh-entity set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list> surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>sub_name |
    ELEMENT BLOCK SUBROUTINE = <string>sub_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional command
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names

END TIME STEP INITIALIZATION
```

The TIME STEP INITIALIZATION command block, which appears in the region scope, is used to flag a user subroutine to run at the beginning of every time step. This subroutine can be used to compute quantities used by other command types. For example, if the traction on a surface was dependent on the area, the time step initialization subroutine could be used to calculate the area, and that area could be stored and later read when calculating the traction. The user initialization subroutine will pass the specified mesh objects to the subroutine for use in calculating some value.

The TIME STEP INITIALIZATION command block contains two groups of commands—mesh entity set and user subroutine. In addition to the command lines in the these command groups, there is an additional command line: ACTIVE PERIODS or INACTIVE PERIODS. Following are descriptions of the different command groups and the ACTIVE PERIODS or INACTIVE PERIODS command line.

11.2.3.1 Mesh-Entity Set Commands

The mesh-entity set commands portion of the TIME STEP INITIALIZATION command block specifies the nodes, element faces, or elements associated with the particular subroutine that

will be run at the beginning of the applicable time steps. This portion of the command block can include some combination of the following command lines:

```
NODE SET = <string list>nodelist_names
SURFACE = <string list>surface_names
BLOCK = <string list>block_names
INCLUDE ALL BLOCKS
REMOVE NODE SET = <string list>nodelist_names
REMOVE SURFACE = <string list> surface_names
REMOVE BLOCK = <string list>block_names
```

These command lines, taken collectively, constitute a set of Boolean operators for constructing a set of nodes, element faces, or elements. See Section 6.1 for more information about the use of these command lines for mesh entities. There must be at least one NODE SET, SURFACE, BLOCK, or INCLUDE ALL BLOCKS command line in the command block.

11.2.3.2 User Subroutine Commands

The following command lines are related to the user subroutine specification:

```
NODE SET SUBROUTINE = <string>subroutine_name |
  SURFACE SUBROUTINE = <string>subroutine_name |
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
SUBROUTINE REAL PARAMETER: <string>param_name
  = <real>param_value
SUBROUTINE INTEGER PARAMETER: <string>param_name
  = <integer>param_value
SUBROUTINE STRING PARAMETER: <string>param_name
  = <string>param_value
```

Only one of the first three command lines listed above can be specified in the command block. The particular command line selected depends on the mesh-entity type of the variable being initialized. For example, variables associated with nodes would be initialized if you are using the NODE SET SUBROUTINE command line, variables associated with faces if you are using the SURFACE SUBROUTINE command line, and variables associated with elements if you are using the ELEMENT BLOCK SUBROUTINE command line. The string subroutine_name is the name of a FORTRAN subroutine that is written by the user.

Following the selected subroutine command line are other command lines that may be used to implement the user subroutine option. These command lines are described in Section 11.2.2 and consist of SUBROUTINE DEBUGGING OFF, SUBROUTINE DEBUGGING ON, SUBROUTINE REAL PARAMETER, SUBROUTINE INTEGER PARAMETER, and SUBROUTINE STRING PARAMETER. Examples of using these command lines are provided throughout Chapter 11.

11.2.3.3 Additional Command

The ACTIVE PERIODS or INACTIVE PERIODS command line can optionally appear in the TIME STEP INITIALIZATION command block:

```
ACTIVE PERIODS = <string list>period_names
INACTIVE PERIODS = <string list>period_names
```

The ACTIVE PERIODS or INACTIVE PERIODS command line is used to activate or deactivate the running of the user subroutine at the beginning of every time step for certain time periods. See Section 2.5 for more information about this optional command line.

11.2.4 User Variables

```
BEGIN USER VARIABLE <string>var_name
  TYPE = <string>NODE|ELEMENT|GLOBAL
    [<string>REAL|INTEGER LENGTH = <integer>length]|
    [<string>SYM_TENSOR|FULL_TENSOR|VECTOR]
  GLOBAL OPERATOR = <string>SUM|MIN|MAX
  INITIAL VALUE = <real list>values
  INITIAL VARIATION = <real list>values LINEAR DISTRIBUTION
  USE WITH RESTART
END [USER VARIABLE <string>var_name]
```

The `USER VARIABLE` command block is used to create a user-defined variable. This kind of variable may be used for scratch space in a user subroutine or for some user-defined output. A user-defined variable may be output to the results file or the history file just like any other defined variable; i.e., a user-defined variable once defined by the `USER VARIABLE` command block can be specified in a `USER OUTPUT` command block, a `RESULTS OUTPUT` command block, and a `HISTORY OUTPUT` command block.

User-defined variables are associated with mesh entities. For example, a node variable will exist at every node of the model. An element variable will exist on every element of the model. A global variable will have a single value for the entire model.

If the user-defined variable functionality is used in conjunction with restart, the `USE WITH RESTART` command line must be included.

Known Issue: User defined variables are not currently supported with heartbeat output (see Section 9.4).

The `USER VARIABLE` command block is placed within a Sierra/SM region. The command block begins with the input line:

```
BEGIN USER VARIABLE <string>var_name
```

and ends with the input line:

```
END [USER VARIABLE <string>var_name]
```

where `var_name` is a user-selected name for the variable.

In the above command block:

- A user-defined variable has an associated type that is specified by the `TYPE` command line, which itself contains several parameters. The `TYPE` command line is required.

  1. The variable must be a nodal quantity, an element quantity, or a global quantity. The options `NODE`, `ELEMENT`, and `GLOBAL` determine whether the variable will be a nodal, element, or global quantity. One of these options must appear on the `TYPE` command line.

2. The user-defined variable can be either an integer or a real, as specified by the INTEGER or REAL option.
3. The length of the variable must be set by using one of the options SYM_TENSOR, FULL_TENSOR, VECTOR, or LENGTH = <integer>length. If the LENGTH option is used, the user must specify whether the variable is an integer number or a real number by using the INTEGER or REAL option. If the SYM_TENSOR option is used, the variable has six real components. If the FULL_TENSOR is used, the variable has nine real components. If the VECTOR option is used, the variable has three real components. The three options SYM_TENSOR, FULL_TENSOR, and VECTOR all imply real numbers, and thus the REAL option need not be included in the command line when one of these three options is specified.

Some examples of the TYPE command line follow:

```
type = global real length = 1
type = element tensor
type = element real length = 3
type = node sym_tensor
type = node vector
```

- If you use the GLOBAL option on the TYPE command line, a global variable is created, and this global variable must be given an associated reduction type, which is specified by the GLOBAL OPERATOR command line. The reduction type tells Sierra/SM how to reduce the individual values stored on each processor to a mesh global value. Global reductions are performed at the end of each time step. Any modifications to a global variable made by an aupst_local_put_global_var call (see Section 11.1.2.5) will not be seen until the next time step after the user-defined global variables have been updated and reduced. The SUM operator sums all processor variable contributions. The MAX operator takes the maximum value of the aupst_local_put_global_var calls. The MIN operator takes the minimum value of the aupst_local_put_global_var calls.

- One or more initial values may be specified for the user-defined variable in the INITIAL VALUE command line. The number of initial values specified should be the same as the length of the variable, as specified in the TYPE command line either explicitly via the LENGTH option or implicitly via the SYM_TENSOR, FULL_TENSOR, or VECTOR option. The initial values will be copied to the variable space on every mesh object on which the variable is defined. Only real type variables may be given initial values at this time.

- The initial value of the user variable can be given some random distribution by use of the the INITIAL VARIATION command line. If the INITIAL VARIATION command is used the INITIAL VALUE command must also be used. In addition the number of values specified in the initial variation command must exactly match the number of values specified in the initial value command. When the initial variation command is used the initial values of the variable will be set as initial value plus or minus some random factor times the initial variation. Currently the only random distribution supported is the linear distribution. With

linear distribution the random values will be distributed evenly from initial value minus variation to initial value plus variation.

- All intrinsic type options such as REAL, INTEGER, SYM_TENSOR, FULL_TENSOR, VECTOR and the LENGTH option can be used with any of the mesh entity options (NODE, ELEMENT, GLOBAL).

- As indicated previously, if the user-defined variable functionality is used in conjunction with restart, the USE WITH RESTART command line must be included.

11.3 User Subroutines: Compilation and Execution

Running a code with user subroutines is a two-step process. First, you must create an executable version of Sierra/SM that recognizes the user subroutines. Next, you must use this version of Sierra/SM for an actual Sierra/SM run with an input file that incorporates the proper user subroutine command lines.

How the above two steps are carried out is site-specific. The actual process will depend on how Sierra/SM is set up at your installation. We will give an example that shows how the process is carried out on various systems at Sandia using SIERRA command lines. SIERRA is a general code framework and code management system at Sandia.

For the first step, you will need the user subroutine, in a FORTRAN file, and a Sierra/SM input file that makes use of the user subroutine. You will use a system command line of the general form shown below. Note that the examples shown here use the presto executable. Either the `presto` or `adagio` executable could be used as they are completely interchangeable.

```
% sierra presto -i <string>input_file_name --make
```

Suppose that you have a subdirectory in your area called `test` and you wish to incorporate a user subroutine called `blast_load`. The actual user subroutine will be in a file called `blast_load.F`, and the associated input file will be called `blast_load_1.i`. Both of these files will be in the directory `test`. In the input file, you will have the following command line in the SIERRA scope:

```
USER SUBROUTINE FILE = blast_load.F
```

You will also have some subset of the command lines described in the previous section in your Sierra/SM input file. The specific form of the system command line to execute the first step of the user subroutine process is shown below.

```
% sierra presto -i blast_load_1.i --make
```

The above command will create a local version of Sierra/SM in a local directory named `UserSubsProject`. The system command line to run the local version of Presto is shown below.

```
% sierra presto -i <string>input_file_name
        -x UserSubsProject
```

The specific form of the system command line you will execute in the subdirectory test is shown below.

```
% sierra presto -i blast_load_1.i -x UserSubsProject
```

The second command line runs Sierra/SM using `blast_load_1.i` as an input file and utilizes the user subroutines in the process. Again, all of this is a site-specific example. You must determine how Sierra/SM is set up at your installation to determine what system command lines are necessary to build Sierra/SM with user subroutines and then use this version of Sierra/SM.

11.4 User Subroutines: Examples

11.4.1 Pressure as a Function of Space and Time

(The following example provides functionality—a blast load on a surface—more applicable to Presto than Adagio. It is included in both manuals as it is instructive in the general use of a user-defined subroutine.)

The following code is an example of a user subroutine to compute blast pressures on a group of faces. The blast pressure simulates a blast occurring at a specified position and time. The blast wave radiates out from the center of the blast and dissipates over time. This subroutine is included in the input file as follows:

```
#In the SIERRA scope:
user subroutine file = blast_load.F

#In the region scope:
begin pressure
  surface = surface_1
  surface subroutine = blast_load
  subroutine real parameter: pos_x = 5.0
  subroutine real parameter: pos_y = 5.0
  subroutine real parameter: pos_z = 1.6
  subroutine real parameter: wave_speed = 1.5e+02
  subroutine real parameter: blast_time = 0.0
  subroutine real parameter: blast_energy = 1.0e+09
  subroutine real parameter: blast_wave_width = 0.75
end pressure
```

The FORTRAN 77 subroutine listing follows. Note that it would be possible to increase the speed of this subroutine by calling the topology functions (see Section 11.1.2.6) on groups of elements, though this would increase subroutine complexity.

```
c
c  Subroutine to simulate a blast load on a surface
c
      subroutine blast_load(num_faces, num_vals,
     &  eval_time, faceID, pressure, flags, err_code)

      implicit none
c
c  Subroutine input arguments
c
      integer num_faces
      double precision eval_time
      integer faceID(num_faces)
```

```
      integer num_vals

c
c  Subroutine output arguments
c
      double precision pressure(num_vals, num_faces)
      integer flags(num_faces)
      integer err_code
c
c  Variables to hold the subroutine parameters
c
      double precision pos_x, pos_y, pos_z, wave_speed,
     &                 blast_time, blast_energy,
     &                 blast_wave_width
c
c  Local variables
c
      integer iface, inode
      integer cur_face_id, face_topo, num_nodes
      integer num_comp_check
      double precision dist, blast_o_rad, blast_i_rad
      double precision blast_volume, blast_pressure
      integer query_error
      double precision face_center(3)
c
c  Create some static variables to hold queried
c  information.  Assume no face has more than 10
c  nodes
c
      double precision face_nodes(10)
      double precision face_coords(3, 10)
c
c  Extract the subroutine parameters
c
      call aupst_get_real_param("pos_x",pos_x,query_error)
      call aupst_get_real_param("pos_y",pos_y,query_error)
      call aupst_get_real_param("pos_z",pos_z,query_error)
      call aupst_get_real_param("wave_speed",wave_speed,
     &                          query_error)
      call aupst_get_real_param("blast_energy",
     &                          blast_energy,query_error)
      call aupst_get_real_param("blast_time",
     &                          blast_time,query_error)
      call aupst_get_real_param("blast_wave_width",
     &                  blast_wave_width, query_error)
c
c  Determine the outer radius of the blast wave
```

```
c
      blast_o_rad = (eval_time - blast_time) * wave_speed
      if(blast_o_rad .le. 0.0) return;
c
c  Determine the inner radius of the blast wave
c
      blast_i_rad = blast_o_rad - blast_wave_width
      if(blast_i_rad .le. 0.0) blast_i_rad = 0.0
c
c  Determine the total volume the blast wave occupies
c
      blast_volume = 3.1415 * (4.0/3.0) *
     &               (blast_o_rad**2 - blast_i_rad**2)
c
c  Determine the total pressure on faces inside the
c  blast wave
c
      blast_pressure = blast_energy / blast_volume
c
c  Loop over all faces in the set
c
      do iface = 1, num_faces
c
c  Extract the topology of the current face
c
        cur_face_id = faceID(iface)
        call aupst_get_face_topology(1, cur_face_id,
     &                           face_topo, query_error)
c
c  Determine the number of nodes of the current face
c
        num_nodes = mod(face_topo,100)
c
c  Extract the node ids for nodes contained in the current
c  face
c
        call aupst_get_face_nodes(1, cur_face_id,
     &                            face_nodes, query_error)
c
c  Extract the nodal coordinates of the face nodes
c
        call aupst_get_node_var(num_nodes, 3, face_nodes,
     &          face_coords, "coordinates", query_error)
c
c  Compute the centroid of the face
c
        face_center(1) = 0.0
```

```
          face_center(2) = 0.0
          face_center(3) = 0.0
          do inode = 1, num_nodes
            face_center(1) = face_center(1) +
     &                       face_coords(1,inode)
            face_center(2) = face_center(2) +
     &                       face_coords(2,inode)
            face_center(3) = face_center(3) +
     &                       face_coords(3,inode)
          enddo
          face_center(1) = face_center(1)/num_nodes
          face_center(2) = face_center(2)/num_nodes
          face_center(3) = face_center(3)/num_nodes
c
c  Determine the distance from the current face
c  to the blast center
c
          dist = sqrt((face_center(1) - pos_x)**2 +
     &                (face_center(2) - pos_y)**2 +
     &                (face_center(3) - pos_z)**2)
c
c  Apply pressure to the current face if it falls within
c  the blast wave
c
          if(dist .ge. blast_i_rad .and.
     &       dist .le. blast_o_rad) then
            pressure(1,iface) = blast_pressure
          else
            pressure(1,iface) = 0.0
          endif
        enddo
        err_code = 0
        end
```

11.4.2 Error Between a Computed and an Analytic Solution

The following code is a user subroutine to compute the error between Sierra/SM-computed results and results from an analytic manufactured solution. This subroutine is called by a `USER OUTPUT` command block immediately prior to producing an output Exodus file. The error for the mesh is computed by taking the squared difference between the computed and analytic displacements at every node. Finally, a global sum of the error is produced along with the square root norm of the error.

This user subroutine requires a user variable, which is defined in the Sierra/SM input file. The command block for the user variable specified in this user subroutine is as follows:

```
begin user variable conv_error
```

```
  type = global real length = 1
   global operator = sum
   initial value = 0.0
 end user variable conv_error
```

The subroutine is called in the Sierra/SM input file as follows:

```
begin user output
  node set = nodelist_10
  node set subroutine = conv0_error
  subroutine real parameter: char_length = 1.0
  subroutine real parameter: char_time   = 1.0e-3
  subroutine real parameter: x_offset    = 0.0
  subroutine real parameter: y_offset    = 0.0
  subroutine real parameter: z_offset    = 0.0
  subroutine real parameter: t_offset    = 0.0
  subroutine real parameter: u0          = 0.01
  subroutine real parameter: v0          = 0.02
  subroutine real parameter: w0          = 0.03
  subroutine real parameter: alpha       = 1.0
  subroutine real parameter: youngs_modulus = 10.0e6
  subroutine real parameter: poissons_ratio = 0.3
  subroutine real parameter: density        = 0.0002588
  subroutine real parameter: num_nodes      = 125.0
end user output
```

The FORTRAN listing for the subroutine is as follows:

```
      subroutine conv0_error(num_nodes, num_vals,
     &  eval_time, nodeID, values, flags, ierror)
      implicit none

      integer num_nodes
      integer num_vals
      double precision eval_time
      integer nodeID(num_nodes)
      double precision values(1)
      integer flags(1)
      integer ierror
c
c     Local vars
c
      integer inode
      integer error_code
      double precision clength, ctime, xoff, yoff, zoff, toff
      double precision zero, one, two, three, four, nine
```

```
      double precision mod_coords(3,3000)
      double precision cdispl(3,3000)
      integer num_comp_check
      double precision expat
      double precision x, y, z, t
      double precision u0, v0, w0, alpha
      double precision pi
      double precision half
      double precision mdisplx, mdisply, mdisplz
      double precision xdiff, ydiff, zdiff
      double precision conv_error
      double precision numnod

      pi    = 3.141592654
      half  = 0.5
      zero  = 0.0
      one   = 1.0
      two   = 2.0
      three = 3.0
      four  = 4.0
      nine  = 9.0
c
c  Check that the nodal coordinates will fit into the
c  statically allocated array
c
      if(num_nodes .gt. 3000) then
        write(6,*) ŠERROR in sphere disp, Ś,
     &  num_nodes exceeds static array sizeŠ
        ierror = 1
        return
      endif
c
c  Extract the model coordinates for all nodes
c
      call aupst_check_node_var(num_nodes, num_comp_check,
     &                          nodeID, "model_coordinates",
     &                          ierror)
      if(ierror .ne. 0) return
      if(num_comp_check .ne. 3) return
      call aupst_get_node_var(num_nodes, num_comp_check,
     &       nodeID, mod_coords, "model_coordinates",
     &       ierror)
c
c  Extract the computed displacements for all nodes
c
      call aupst_check_node_var(num_nodes, num_comp_check,
     &                          nodeID, "displacement",
```

```
     &                            ierror)
      if(ierror .ne. 0) return
      if(num_comp_check .ne. 3) return
      call aupst_get_node_var(num_nodes, num_comp_check,
     &       nodeID, cdispl, "displacement",
     &       ierror)
c
c  Extract the subroutine parameters.
c
      call aupst_get_real_param("char_length",
     &                            clength,error_code)
      call aupst_get_real_param("char_time",
     &                            ctime,error_code)
      call aupst_get_real_param("x_offset",xoff,error_code)
      call aupst_get_real_param("y_offset",yoff,error_code)
      call aupst_get_real_param("z_offset",zoff,error_code)
      call aupst_get_real_param("t_offset",toff,error_code)
      call aupst_get_real_param("u0",u0,error_code)
      call aupst_get_real_param("v0",v0,error_code)
      call aupst_get_real_param("w0",w0,error_code)
      call aupst_get_real_param("alpha",alpha,error_code)
      call aupst_get_real_param("num_nodes",
     &                            numnod,error_code)
c
c  Calculate a solution scaling factor
c
      expat = half * ( one - cos( pi * eval_time / ctime ) )
c
c  Compute the expected solution at each node and do a
c  sum of the differences from the analytic solution
c
      conv_error = zero
      do inode = 1, num_nodes
c
c  Set the displacement value from the manufactured solution
c
        x = ( mod_coords(1,inode) - xoff ) / clength
        y = ( mod_coords(2,inode) - yoff ) / clength
        z = ( mod_coords(3,inode) - zoff ) / clength
c
        mdisplx = u0 * sin(x) * cos(two*y) * cos(three*z)
     *               * expat
        mdisply = v0 * cos(three*x) * sin(y) * cos(two*z)
     *               * expat
        mdisplz = w0 * cos(two*x) * cos(three*y) * sin(z)
     *               * expat
c
```

```
        xdiff = mdisplx - cdispl(1,inode)
        ydiff = mdisply - cdispl(2,inode)
        zdiff = mdisplz - cdispl(3,inode)
        conv_error = conv_error + xdiff*xdiff
     *                          + ydiff*ydiff
     *                          + zdiff*zdiff
c
      enddo
c
      ierror = 0
c
c  Do a parallel sum of the squared errors and extract
c  the total summed value on all processors
c
      call aupst_put_global_var(1,conv_error,
     &                          "conv_error","sum",ierror)
      call aupst_get_global_var(1,conv_error,
     &                          "conv_error",ierror)
c
c  Take the square root of the errors and store that as
c  the net error norm
c
      conv_error = sqrt(conv_error) / sqrt(numnod)
      call aupst_put_global_var(1,conv_error,
     &                          "conv_error","none",ierror)
c
      return
      end
```

11.4.3 Transform Output Stresses to a Cylindrical Coordinate System

The following code is a user subroutine to transform element stresses in global *x*, *y*, and *z* coordinates to a global cylindrical coordinate system. This subroutine could be used to transform the relatively meaningless shell stress in *x*, *y*, and *z* coordinates to more meaningful tangential, hoop, and radial stresses. The subroutine is called from a `USER OUTPUT` command block. It reads in the old stresses, transforms them, and writes them back out to a user-created scratch variable, defined via a `USER VARIABLE` command block, for output.

```
begin user variable cyl_stress
  type = element sym_tensor length = 1
  initial value = 0.0 0.0 0.0 0.0 0.0 0.0
end user variable

begin user output
  block = block_1
  element block subroutine = aupst_cyl_transform
```

```
  subroutine string parameter: origin_point = Point_O
  subroutine string parameter: z_point      = Point_Z
  subroutine string parameter: xz_point     = Point_XZ
  subroutine string parameter: input_stress = memb_stress
  subroutine string parameter: output_stress = cyl_stress
end user output
```

The FORTRAN listing for the subroutine is as follows:

```
      subroutine aupst_cyl_transform(num_elems, num_vals,
     *  eval_time, elemID, values, flags, ierror)
      implicit none
#include<framewk/Fmwk_type_sizes_decl.par>
#include<framewk/Fmwk_type_sizes.par>
c
c  Subroutine Arguments
c
c  num_elems: Input: Number of elements to calculate on
c  num_vals : Input: Ignored
c  eval_time: Input: Time at which to evaluate the stress.
c  elemID   : Input: Global sierra IDs of the input elements
c  values   : I/O  : Ignored, stress will be stored manually
c  flags    : I/O  : Ignored
c  ierror   :Output: Returns non-zero if an error occurs
c
      integer num_elems
      integer num_vals
      double precision eval_time
      integer elemID(num_elems)
      double precision values(1)
      integer flags(1)
      integer ierror
c
c  Fortran cannot dynamically allocate memory, thus worksets
c  will be iterated over by  chucks each of size chunk_size.
c
      integer chunk_size
      parameter (chunk_size = 100)
      integer chunk_ids(chunk_size)
c
c  Subroutine parameter data
c
      character*80     origin_point_name
      double precision origin_point(3)
      character*80     z_point_name
      double precision z_point(3)
      character*80     xz_point_name
```

```
      double precision xz_point(3)
      character*80     input_stress_name
      character*80     output_stress_name
c
c  Local element data for centroids and rotation vectors
c
      double precision cent(3)
      double precision centerline_pos(3)
      double precision dot_prod
      double precision z_vec(3)
      double precision r_vec(3)
      double precision theta_vec(3)
      double precision rotation_tensor(9)
c
c  Chunk data storage
c
      double precision elem_centroid(3, chunk_size)
      double precision input_stress_val(6, chunk_size)
      double precision output_stress_val(6, chunk_size)
c
c  Simple iteration variables
c
      integer error_code
      integer ichunk, ielem
      integer zero_elem, nel
c
c  Extract the current subroutine parameters.  origin_point
c  is the origin of the coordinate system
c  z_point is a point on the z axis of the coordinate system
c  xz_point is a point on the xz plane
c
      call aupst_get_string_param("origin_point",
     &                            origin_point_name,
     &                            error_code)
      call aupst_get_string_param("z_point",
     &                            z_point_name,
     &                            error_code)
      call aupst_get_string_param("xz_point",
     &                            xz_point_name,
     &                            error_code)
      call aupst_get_string_param("input_stress",
     &                            input_stress_name,
     &                            error_code)
      call aupst_get_string_param("output_stress",
     &                            output_stress_name,
     &                            error_code)
c
```

```
c  Use the point names to look up the coordinates of each
c  relevant point
c
      call aupst_get_point(origin_point_name, origin_point,
     &                     error_code)
      call aupst_get_point(z_point_name, z_point,
     &                     error_code)
      call aupst_get_point(xz_point_name, xz_point,
     &                     error_code)
c
c  Compute the z axis vector
c
      z_vec(1) = z_point(1) - origin_point(1)
      z_vec(2) = z_point(2) - origin_point(2)
      z_vec(3) = z_point(3) - origin_point(3)
c
c  Transform z_vec into a unit vector, abort if it is invalid
c
      call aupst_unitize_vector(z_vec, ierror)
      if(ierror .ne. 0) return
c
c  Loop over chunks of the data arrays
c
      do ichunk = 1, (num_elems/chunk_size + 1)
c
c  Determine the first and last element number for the
c  current chunk of elements
c
       zero_elem = (ichunk-1) * chunk_size
       if((zero_elem + chunk_size) .gt. num_elems) then
        nel = num_elems - zero_elem
       else
        nel = chunk_size
       endif
c
c  Copy the elemIDs for all elems in the current chunk to a
c  temporary array
c
       do ielem = 1, nel
       chunk_ids(ielem) = elemID(zero_elem + ielem)
       enddo
c
c  Extract the element centroids and stresses
c
       call aupst_get_elem_centroid(nel, chunk_ids,
     &                              elem_centroid,
     &                              ierror)
```

```
      call aupst_get_elem_var(nel, 6, chunk_ids,
     &                         input_stress_val,
     &                         input_stress_name, ierror)
c
c  Loop over each element in the current chunk
c
      do ielem = 1, nel
c
c  Find the closest point on the cylinder centerline axis
c  to the element centroid
c
        cent(1) = elem_centroid(1, ielem) - origin_point(1)
        cent(2) = elem_centroid(2, ielem) - origin_point(2)
        cent(3) = elem_centroid(3, ielem) - origin_point(3)
        dot_prod = cent(1) * z_vec(1) +
     &             cent(2) * z_vec(2) +
     &             cent(3) * z_vec(3)
        centerline_pos(1) = z_vec(1) * dot_prod
        centerline_pos(2) = z_vec(2) * dot_prod
        centerline_pos(3) = z_vec(3) * dot_prod
c
c  Compute the current normal radial vector
c
        r_vec(1) = cent(1) - centerline_pos(1)
        r_vec(2) = cent(2) - centerline_pos(2)
        r_vec(3) = cent(3) - centerline_pos(3)
        call aupst_unitize_vector(r_vec, ierror)
        if(ierror .ne. 0) return
c
c  Compute the current hoop vector
c
        theta_vec(1) = z_vec(2)*r_vec(3) - r_vec(2)*z_vec(3)
        theta_vec(2) = z_vec(3)*r_vec(1) - r_vec(3)*z_vec(1)
        theta_vec(3) = z_vec(1)*r_vec(2) - r_vec(1)*z_vec(2)
c
c  The r, theta, and z vectors describe the new stress
c  coordinate system, Transform the input stress tensor
c  in x,y,z coords to the output stress tensor in r, theta,
c  and z coords use the unit vectors to create a rotation
c  tensor
c
        rotation_tensor(k_f36xx) = r_vec(1)
        rotation_tensor(k_f36yx) = r_vec(2)
        rotation_tensor(k_f36zx) = r_vec(3)
        rotation_tensor(k_f36xy) = theta_vec(1)
        rotation_tensor(k_f36yy) = theta_vec(2)
        rotation_tensor(k_f36zy) = theta_vec(3)
```

```
         rotation_tensor(k_f36xz) = z_vec(1)
         rotation_tensor(k_f36yz) = z_vec(2)
         rotation_tensor(k_f36zz) = z_vec(3)
c
c  Rotate the current stress tensor to the new configuration
c
         call fmth_rotate_symten33(1, 1, 0, rotation_tensor,
     &                            input_stress_val(1,ielem),
     &                           output_stress_val(1,ielem))
        enddo
c
c  Store the new stress
c
         call aupst_put_elem_var(nel, 6, chunk_ids,
     &                           output_stress_val,
     &                           output_stress_name, ierror)
       enddo
       ierror = 0
       end
```

11.5 User Subroutines: Library

A number of user subroutines are used commonly and have been permanently incorporated into the code. These subroutines are used just like any other subroutines, but they do not need to be compiled into the code. (The user need be concerned only about the Sierra/SM command lines.) This section describes the usage of each of these subroutines.

11.5.1 aupst_cyl_transform

Author: Nathan Crane

Purpose:

The purpose of this subroutine is to transform element stresses from a global rectangular coordinate system to a local cylindrical coordinate system. This subroutine is generally called by a `USER OUTPUT` command block. For example:

```
begin user output
  block = block_1
  element block subroutine = aupst_cyl_transform
  subroutine string parameter: origin_point = Point_O
  subroutine string parameter: z_point      = Point_Z
  subroutine string parameter: xz_point     = Point_XZ
  subroutine string parameter: input_stress = memb_stress
  subroutine string parameter: output_stress = cyl_stress
end user output
```

Requirements:

This subroutine requires a tensor variable to store the cylindrical stress into a variable for each element. The variable is created by the following command block in the Sierra/SM region:

```
begin user variable cyl_stress
   type = element sym_tensor length = 1
   initial value = 0.0 0.0 0.0 0.0 0.0 0.0
end user variable
```

Parameters:

Parameter Name	Usage	Description
origin_point	String	Name of the point at the cylinder origin.
z_point	String	Point on the cylinder axis.
xz_point	String	Point on the line that passes through theta = 0 on the cylinder.
input_stress	String	Name of the Sierra/SM internal input stress tensor variable.
output_stress	String	Name of the Sierra/SM internal output stress tensor variable.

11.5.2 aupst_rec_transform

Author: Daniel Hammerand

Purpose:

The purpose of this subroutine is to transform element stresses from a global rectangular coordinate system to a different local rectangular coordinate system. This subroutine is generally called by a `USER OUTPUT` command block. For example:

```
begin user output
  block = block_1
  element block subroutine = aupst_rec_transform
  subroutine string parameter: origin_point = Point_O
  subroutine string parameter: z_point      = Point_Z
  subroutine string parameter: xz_point     = Point_XZ
  subroutine string parameter: input_stress = memb_stress
  subroutine string parameter: output_stress = new_stress
end user output
```

Requirements:

This subroutine requires a tensor variable to store the new stress into a variable for each element. The variable is created by the following command block in the Sierra/SM region:

```
begin user variable new_stress
  type = element sym_tensor length = 1
  initial value = 0.0 0.0 0.0 0.0 0.0 0.0
end user variable
```

Parameters:

Parameter Name	Usage	Description
origin_point	String	Name of the point at the cylinder origin.
z_point	String	Point on the cylinder axis.
xz_point	String	Point on the line that passes through theta = 0 on the cylinder.
input_stress	String	Name of the Sierra/SM internal input stress tensor variable.
output_stress	String	Name of the Sierra/SM internal output stress tensor variable.

11.5.3 copy_data

Author: Jason Hales

Purpose:

The purpose of this subroutine is to copy data from one variable to another with offsets given for both variables. This subroutine is generally called by a `USER OUTPUT` command block. For example:

```
begin user output
  block = block_1
  element block subroutine = copy_data
  subroutine integer parameter: source_offset = 4
  subroutine string parameter:  source_name = stress
  subroutine integer parameter: destination_offset = 1
  subroutine string parameter:  destination_name = uservarxy
end user output
```

Requirements:

This subroutine requires that the source and destination fields exist and have lengths at least as great as the values supplied as offsets. The fields used may be defined by the user as variables. In this example, the variable is created by the following command block in the Sierra/SM region:

```
begin user variable uservarxy
  type = element real length = 1
  initial value = 0.0
end user variable
```

Parameters:

Parameter Name	Usage	Description
source_offset	Integer	The offset into the source variable.
source_name	String	The name of the source variable.
destination_offset	Integer	The offset into the destination variable.
destination_name	String	The name of the destination variable.

11.5.4 trace

Author: Jason Hales

Purpose:

The purpose of this subroutine is to compute the trace of a tensor. This subroutine is generally called by a `USER OUTPUT` command block. For example:

```
begin user output
  block = block_1
  element block subroutine = trace
  subroutine string parameter:  source_name = log_strain
  subroutine string parameter:  destination_name = uvarbulkstrain
end user output
```

Requirements:

This subroutine requires that the source and destination fields exist. The source field should have a length of six. The destination field should have a length of one. The destination field will typically be defined by the user as a variable. In this example, the variable is created by the following command block in the Sierra/SM region:

```
begin user variable uvarbulkstrain
  type = element real length = 1
  initial value = 0.0
end user variable
```

Parameters:

Parameter Name	**Usage**	**Description**
source_name	String	The name of the source variable.
destination_name	String	The name of the destination variable.

Chapter 12

Transfers

It is sometimes desirable to chain two or more analyses (procedures) together. A common example of this is the need to preload an assembly quasistatically and then subject that assembly to a loading environment best suited to an explicit transient dynamics analysis. The displacements and stresses produced by the quasistatic preload are initial conditions for the transient dynamics event. These displacements and stresses must be transferred from the initial analysis to the subsequent one.

This chapter reviews the concept of transfers in SIERRA and outlines the syntax required to perform a transfer of information between procedures.

12.1 SIERRA Transfers

Applications built on the SIERRA computational framework share underlying data structures. This makes it convenient to couple applications together using transfers.

The coupling available through SIERRA is of two types. The first is what is called intra-procedural coupling. In this case, multiple regions within a single procedure share data. This enables multi-physics analysis such as thermal-mechanical coupling. The details of this type of coupling, along with the syntax to support it, will not be covered here. Coupled codes such as Calagio and Arpeggio use this type of coupling.

The second type of coupling is inter-procedural coupling. Here, the result from one procedure is handed to the next procedure. This is the type of coupling used when moving from one analysis stage to another in Adagio and Presto. When using this type of coupling, the two procedures generally have only one region each.

12.2 Inter-procedural Transfers

The inter-procedural transfers used by Adagio and Presto can transfer data from one or all blocks of the preceding or sending procedure to the subsequent or receiving procedure. The commands to control the transfers should appear at the procedure scope in the second procedure.

When using the inter-procedural transfers, all of the appropriate element data will be transferred from the sending to the receiving elements. Nodal data will be transferred based on the type of analysis done in the two procedures. For example, if the first procedure is a quasistatic analysis and the second is an explicit transient dynamics analysis, the displacements of the nodes will be transferred but not their velocities.

Warning: Transfer of data for node-based tetrahedra is currently not fully supported. A coupled analysis with node-based tetrahedra will only be correct if the tet elements are not deformed in the first (sending) procedure, in which case initialization in the second (receiving) procedure is appropriate.

The set of available commands are below.

```
BEGIN PROCEDURAL TRANSFER <string>name

  BLOCK = <string list>block_name
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_name

  BEGIN INTERPOLATION TRANSFER <string>name

    BLOCK BY BLOCK
    NEAREST ELEMENT COPY
```

```
    SEND BLOCKS = <string list>block_name
    SEND COORDINATES = ORIGINAL|CURRENT

    RECEIVE BLOCKS = <string list>block_name
    RECEIVE COORDINATES = original|current

    TRANSFORMATION TYPE = NONE|RIGIDBODY

  END [INTERPOLATION TRANSFER <string>name]
END [PROCEDURAL TRANSFER <string>name]
```

The inter-procedural transfers can be invoked in one of two ways. If the sending and receiving regions use the same finite element model, data can be copied from the sending to the receiving region. In this case, the first three lines of syntax in the transfer block can be used to copy data for all blocks except those that are rigid bodies.

If different finite element models are used by the sending and receiving meshes, or if it is desired to copy data for rigid bodies, the `INTERPOLATION TRANSFER` block must be used, and the first three lines in the `PROCEDURAL TRANSFER` block should not be used. If the lines appropriate for the copy of data are used when they do not apply, their behavior is undefined.

12.2.1 Copying Data with Inter-procedural Transfers

To copy data from one or more blocks in a sending region to the matching blocks in the receiving region, use these line commands:

```
  BLOCK = <string list>block_name
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_name
```

The first of these, the `BLOCK` line command, is used to list blocks that should be copied. If it is desired to copy data for all blocks, use the `INCLUDE ALL BLOCKS` line command. Finally, one or more blocks may be removed from the set of all blocks with the `REMOVE BLOCK` line command. Either the `BLOCK` or `INCLUDE ALL BLOCKS` line command must appear.

12.2.2 Interpolating Data with Interpolation Transfers

When different finite element models are used by the sending and receiving regions, or when it is desired to transfer data for rigid bodies, the `INTERPOLATION TRANSFER` must be used.

If data for a rigid body is to be transferred, use the `TRANSFORMATION TYPE = RIGIDBODY` line command along with the `SEND BLOCKS` and `RECEIVE BLOCKS` line commands. Transfers of rigid

body information should be entered for each rigid body separately, which is done simply by listing only one block on each of the SEND BLOCKS and RECEIVE BLOCKS lines.

When transferring data for non-rigid body blocks, the only required line commands are the SEND BLOCKS and RECEIVE BLOCKS line commands. These each list one or more blocks to be included in the transfer.

The interpolation transfers move data from sending to receiving meshes by performing searches and interpolating using shape functions. For nodal data, a node in the receiving mesh is located in the sending mesh. The node will be found in (or near) an element in the sending mesh. The parametric coordinates of the point associated with the receiving node will be calculated based on the sending element, and those parametric coordinates will be used in conjunction with the shape functions and the data on the nodes of the sending element to calculate values for the receiving node.

For the transfer of element data, the location of the center of the receiving element will be located in the sending mesh. This point will appear in (or near) an element in the sending mesh. A patch of elements in the sending mesh will be created centered around the element that contains the point associated with the receiving element. The data in the patch of sending elements will be interpolated using a least squares approach to the point associated with the receiving element, and the result will be given to the receiving element.

The interpolation transfer will give the best results when the sending and receiving meshes are very similar. If the meshes do not represent the same volumes in space, for example, the interpolated values will be suspect at best.

On rare occasions, it may be desired to use the current coordinates instead of the original coordinates in performing the search used by the transfers. Use the SEND COORDINATES = CURRENT and/or RECEIVE COORDINATES = CURRENT line commands for this purpose.

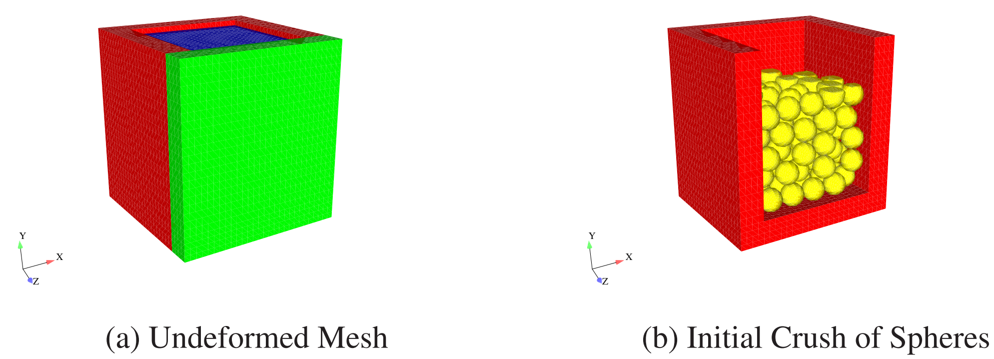
In some instances, the sending and receiving meshes are very similar such that there is a one-to-one correspondence between the list of sending blocks and the list of receiving blocks. In other words, it may be desired to send data from a given block to a corresponding block, from another block to its pair, and so forth. If this is the case, use of the BLOCK BY BLOCK line command will cause a separate transfer to be created for each pair of sending and receiving blocks. This is useful to ensure that data from a single block will be sent to one and only one receiving block. This could also be accomplished by listing multiple transfer blocks in the input file.

The NEAREST ELEMENT COPY line command changes the behavior of the transfer of element variables. Instead of a least squares approach, the use of this line command will cause data to be sent directly from the nearest sending element to the receiving element.

Appendix A

Explicit Dynamic Example Problem

This chapter provides an example problem to illustrate the construction of an input file for an analysis. The example problem consists of 124 spheres made of lead enclosed in a steel box. The steel box has an open top into which a steel plate is placed (see Figure A.1). A prescribed velocity is then applied on the steel plate, pushing it into the box and crushing the spheres contained within using frictionless contact. This problem is a severe test for the contact algorithms as the spheres crush into a nearly solid block. See Figure A.2 for results of this problem.



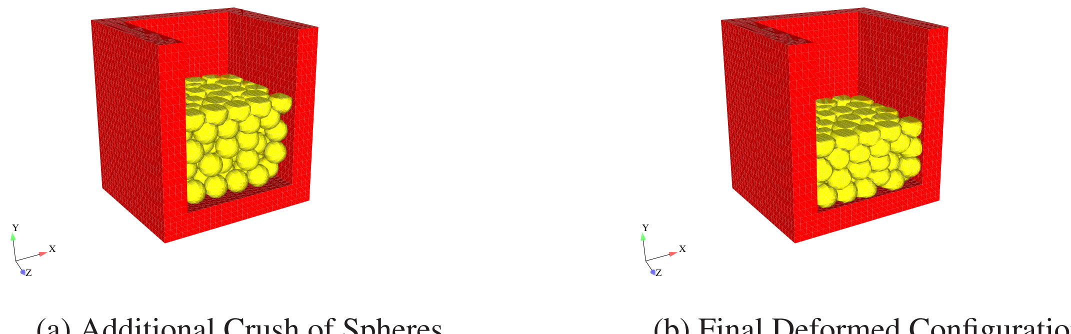
(a) Undeformed Mesh (b) Initial Crush of Spheres

Figure A.1: Mesh for example problem: (a) box (red and green surfaces) with plate in top (blue surface) and (b) mesh with blue and green surfaces removed to show internal spheres (yellow) with initial crush.

The input file is described below, with comments to explain every few lines. Following the description, the full input file is listed again. Most of the key words in this example are all lowercase, which is different from the convention we have used to describe the command lines in this document. However, all the lowercase usage in the following example is an acceptable format in Presto.

The input file starts with a begin sierra statement (i.e., the first line of the SIERRA command block), as is required for all input files:

```
begin sierra crush_124_spheres
```



(a) Additional Crush of Spheres (b) Final Deformed Configuration

Figure A.2: Mesh with blue and green surfaces removed to show internal spheres (yellow) after initial crush shown in Figure A.1 (b).

We now need to define the functions used with this problem. The boundary conditions require a function for the initial velocity, as follows:

```
begin definition for function constant_velocity
  type is piecewise linear
  ordinate is velocity
  abscissa is time
  begin values
    0.0 30.0
    1.0 30.0
  end values
end definition for function constant_velocity
```

To define the boundary conditions, we need to define the direction for the initial velocity—this is in the *y*-direction. We could also choose to simply specify the Y component for the initial condition, but this input file uses directions.

```
define direction y_axis with vector 0.0 1.0 0.0
```

Next we define the material models that will be used for this analysis. There are two materials in this problem: steel for the box, and lead for the spheres. Both materials use the elastic-plastic material model (denoted as `elastic_plastic`).

```
begin property specification for material steel
  density = 7871.966988

  begin parameters for model elastic_plastic
    youngs modulus = 1.999479615e+11
    poissons ratio = 0.33333
    yield stress = 275790291.7
    hardening modulus = 275790291.7
```

```
    beta = 1.0
  end parameters for model elastic_plastic

end property specification for material steel

begin property specification for material lead
  density = 11253.30062

  begin parameters for model elastic_plastic
    youngs modulus = 1.378951459e+10
    poissons ratio = 0.44
    yield stress = 13789514.59
    hardening modulus = 0.0
    beta = 1.0
  end parameters for model elastic_plastic

end property specification for material lead
```

Now we define the finite element mesh. This includes specification of the file that contains the mesh, as well as a list of all the element blocks we will use from the mesh and the material associated with each block. The name of the file is `crush_124_spheres.g`. The specification of the database type is optional—ExodusII is the default. Currently, each element block must be defined individually. For this particular problem, all the spheres are the same element block. Each sphere is a distinct geometry entity, but all spheres constitute one element block in the Exodus II database. Note that the three element blocks that make up the box and lid all reference the same material description. The material description is *not* repeated three times. The material description for steel appears once and is then referenced three times.

```
begin finite element model mesh1
  Database Name = crush_124_spheres.g
  Database Type = exodusII

  begin parameters for block block_1
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_1

  begin parameters for block block_2
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_2

  begin parameters for block block_3
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_3
```

```
  begin parameters for block block_4
    material linear_elastic_lead
    solid mechanics use model elastic_plastic
  end parameters for block block_4

end finite element model mesh1
```

As an alternative to referencing the material description for steel three times as done above, you could define multiple element blocks simultaneously on the same command line. Thus, the three element block specifications with the material `linear_elastic_steel` could be consolidated into one, as follows:

```
begin parameters for block block_1 block 2 block 3
  material linear_elastic_steel
  solid mechanics use model elastic_plastic
end parameters for block block_1 block 2 block 3
```

At this point we have finished specifying physics-independent quantities. We now want to set up the Presto procedure and region, along with the `time control` command block. We start by defining the beginning of the procedure scope, the `time control` command block, and the beginning of the region scope. Only one `time stepping block` command block is needed for this analysis. The termination time is set to 7×10^{-4}.

```
begin presto procedure Apst_Procedure

begin time control
  begin time stepping block p1
    start time = 0.0
    begin parameters for presto region presto
      time step scale factor = 1.0
      time step increase factor = 2.0
      step interval = 25
    end parameters for presto region presto
  end time stepping block p1

  termination time = 7.0e-4
end time control

  begin presto region presto
```

Next we associate the finite element model we defined above (mesh1) with this presto region.

```
    use finite element model mesh1
```

Now we define the boundary conditions on the problem. We prescribe the velocity on the top surface of the box (nodelist_100) to crush the spheres, and we confine the bottom surface of the box (nodelist_200) not to move. Note that although we use node sets to define these boundary conditions, we could have used the corresponding side sets.

```
begin prescribed velocity
  node set = nodelist_100
  direction = y_axis
  function = constant_velocity
  scale factor = -1.0
end

begin fixed displacement
  node set = nodelist_200
  components = Y
end
```

Now we define the contact for this problem. For this problem, we want all four element blocks to be able to contact each other, with a normal tolerance of 0.0001 and a tangential tolerance of 0.0005. In this case, we simply define the same contact characteristics for all interactions. However, we could also specify tolerances and kinematic partition factors for individual interactions. Since no friction model is defined in the block below, the contact defaults to frictionless contact. (There are numerous options you can use to control the contact algorithm. The options you choose will affect contact algorithm efficiency and solution accuracy. See Chapter 8 to determine how to set input for the CONTACT DEFINITION command block to obtain the best level of efficiency and accuracy for your particular problem.)

```
begin contact definition
  skin all blocks = on
   begin search options
     normal tolerance = 0.0001
     tangential tolerance = 0.00005
   end
   begin interaction defaults
     general contact = on
     self contact = on
   end
 end
```

Now we define what variables we want in the results file, as well as how often we want this file to be written. Here we request files written every 7×10^{-6} sec of analysis time. This will result in results output at one hundred time steps (plus the zero time step) since the termination time is set to 7×10^{-4} sec. The output file will be called crush_124_spheres.e, and it will be an Exodus II file (the database type command is optional; it defaults to ExodusII). The variables we are requesting are the displacements and external forces at the nodes, the rotated stresses for the elements, the time-step increment, and the kinetic energy.

```
    begin Results Output output_presto
      Database Name = crush_124_spheres.e
      Database Type = exodusII
      At Time 0.0, Increment = 7.0e-6
      nodal displacement as displ
      nodal force_external as fext
      element stress as stress
      global KineticEnergy as KE
      global timestep
    end
```

Now we end the presto region, presto procedure, and sierra blocks to complete the input file.

```
      end presto region presto
    end presto procedure Apst_Procedure
  end sierra crush_124_spheres
```

Here is the resulting full input file for this problem:

```
begin sierra crush_124_spheres
  begin definition for function constant_velocity
    type is piecewise linear
    ordinate is velocity
    abscissa is time
    begin values
      0.0 30.0
      1.0 30.0
    end values
  end definition for function constant_velocity
  define direction y_axis with vector 0.0 1.0 0.0

  begin property specification for material steel
    density = 7871.966988

    begin parameters for model elastic_plastic
      youngs modulus = 1.999479615e+11
      poissons ratio = 0.33333
      yield stress = 275790291.7
      hardening modulus = 275790291.7
      beta = 1.0
    end parameters for model elastic_plastic

  end property specification for material steel

  begin property specification for material lead
    density = 11253.30062

    begin parameters for model elastic_plastic
      youngs modulus = 1.378951459e+10
      poissons ratio = 0.44
      yield stress = 13789514.59
      hardening modulus = 0.0
      beta = 1.0
    end parameters for model elastic_plastic

  end property specification for material lead

  begin finite element model mesh1
    Database Name = crush_124_spheres.g
    Database Type = exodusII

    begin parameters for block block_1
      material linear_elastic_steel
      solid mechanics use model elastic_plastic
    end parameters for block block_1
```

```
  begin parameters for block block_2
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_2

  begin parameters for block block_3
    material linear_elastic_steel
    solid mechanics use model elastic_plastic
  end parameters for block block_3

  begin parameters for block block_4
    material linear_elastic_lead
    solid mechanics use model elastic_plastic
  end parameters for block block_4

end finite element model mesh1

begin presto procedure Apst_Procedure

  begin time control
    begin time stepping block p1
      start time = 0.0
      begin parameters for presto region presto
        time step scale factor = 1.0
        time step increase factor = 2.0
        step interval = 25
      end parameters for presto region presto
    end time stepping block p1

    termination time = 7.0e-4
  end time control

  begin presto region presto

    use finite element model mesh1

    begin prescribed velocity
      node set = nodelist_100
      direction = y_axis
      function = constant_velocity
      scale factor = -1.0
    end prescribed velocity

    begin fixed displacement
      node set = nodelist_200
      components = Y
    end fixed displacement
```

```
      begin contact definition
        skin all blocks = on
         begin search options
           normal tolerance = 0.0001
           tangential tolerance = 0.00005
         end
         begin interaction defaults
           general contact = on
           self contact = on
         end
       end

       begin Results Output output_presto
         Database Name = crush_124_spheres.e
         Database Type = exodusII
         At Time 0.0, Increment = 7.0e-6
         nodal displacement as displ
         nodal force_external as fext
         element stress as stress
         global KineticEnergy as KE
         global timestep
       end results output output_presto

     end presto region presto
   end presto procedure Apst_Procedure
 end sierra crush_124_spheres
```

Appendix B

Implicit Quasistatic Example Problem

This appendix provides an example problem to illustrate the construction of an input file for an analysis. The problem is modeled after a pencil/eraser that is pressed against and rubbed across a tablet. The pencil, eraser and tablet are represented by blocks 1, 2 and 3 respectively. Block 4 is used to apply kinematics directly to the eraser via tied contact between block 4 and block 2. Block 1 is really superfluous except that when viewed together, blocks 1, 2 and 3 closely resemble a pencil with an eraser on a tablet. The problem demonstrates the overall structure of an input file and includes the use of both tied and frictional contact simultaneously, the multilevel solver, and a linear solver for preconditioning. A schematic of the problem is shown in Figure B.1 and the mesh is shown in Figure B.2.

The problem kinematics and loading are described briefly here. There are two phases of the loading: preload and sliding. In the preload phase, the tablet (block 3) is kinematically fixed in the x-y plane and a vertical load (in the z-direction) is applied to the tablet which presses the tablet against the eraser (block 2). The eraser is kinematically fixed to block 4 via tied contact (see Figure B.2). Block 4 is held fixed in all three coordinate directions for the compression phase. The initial con-

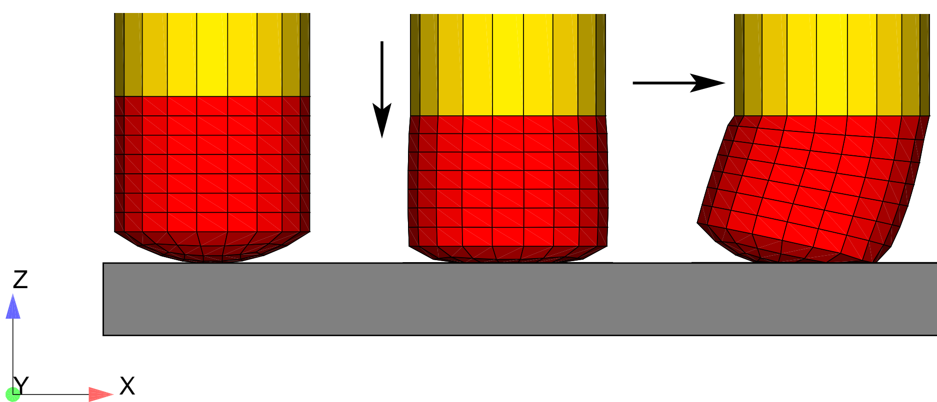


Figure B.1: Eraser schematic; Block 1 (yellow) is pencil, block 2 (red) is eraser, block 3 (gray) is tablet, block 4 (not shown) is tied to eraser and has the kinematics applied to it.

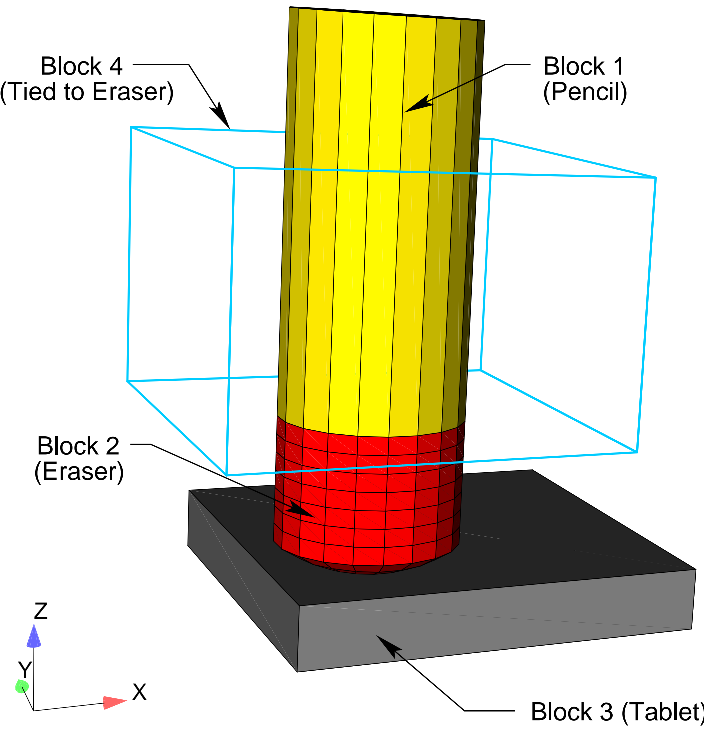


Figure B.2: Complete eraser mesh

figuration and deformations of the preload phase are shown in the first two snapshots on the left in Figure B.1. In the sliding phase, block 4 is kinematically prescribed to move along the *x*-direction, thus sliding or dragging the eraser along the tablet (see Figure B.1) while the tablet force is held constant.

The input file is described below, with comments to explain every few lines. Following the description, the full input file is listed again. Note that all character strings in the input file are presented in lowercase, which is an acceptable format in Adagio.

The input file starts with a begin sierra statement, as is required for all input files:

```
begin sierra eraser
```

We begin by defining vectors corresponding to the coordinate axes. These vectors/directions will be used to define the input for boundary condition blocks that come later.

```
define direction X with vector 1.0 0.0 0.0
define direction Y with vector 0.0 1.0 0.0
define direction Z with vector 0.0 0.0 1.0
```

We now need to define the functions used for this problem. The boundary conditions require a function for the tablet force as well as the sliding motion. Note that both the tablet force and sliding motion are prescribed as functions of time.

```
begin definition for function slide
  type is piecewise linear
  begin values
    0.0      0.0
    1.0      0.0
    2.0      1.0
    3.0      1.0
  end values
end definition for function slide

begin definition for function tablet_force
  type is piecewise linear
  begin values
    0.0      0.0
    1.0      1.0
    3.0      1.0
  end values
end definition for function tablet_force

begin definition for function zero
  type is constant
  begin values
    0.0      0.0
  end values
end definition for function zero
```

Note that the tablet force ramps up over the time interval (0.0,1.0) and then is held constant over the interval (1.0,3.0). The sliding function is zero over the time interval (0.0,1.0) and then it linearly increases over the interval (1.0,2.0). Next, we define the material properties and models that are used in this problem. In this example, we define two sets of material properties: one is called stiff (pencil, tablet, and block 4), and the second is called soft (eraser). We use a linear elastic constitutive model for both cases.

```
begin property specification for material stiff
  density = 1.0
  begin parameters for model elastic
    youngs modulus = 1.e5
    poissons ratio = 0.3
  end parameters for model elastic
end property specification for material stiff

begin property specification for material soft
  density = 1.0
  begin parameters for model elastic
    youngs modulus = 1000.
    poissons ratio = 0.3
  end parameters for model elastic
```

```
end property specification for material soft
```

Now, we define the finite element mesh. This includes specification of the file that contains the mesh, as well as a list of all the element blocks we will use from the mesh and the material associated with each block. The name of the file is `eraser.g`. The specification of the database type is optional-ExodusII is the default. Currently, each element block must be defined individually. Note that the tablet, pencil and block 4 all reference the same material description (stiff). The material description is not repeated three times. The material description for stiff appears once and is then referenced three times.

```
begin finite element model mesh1
  database name = eraser.g
  database type = exodusII
  begin parameters for block block_1 #Pencil
    material stiff
    solid mechanics use model elastic
  end parameters for block block_1 #Pencil
  begin parameters for block block_2 #Eraser
    material soft
    solid mechanics use model elastic
  end parameters for block block_2 #Eraser
  begin parameters for block block_3 #Tablet
    material stiff
    solid mechanics use model elastic
  end parameters for block block_3 #Tablet
  begin parameters for block block_4 #dummy block
    material stiff
    solid mechanics use model elastic
  end parameters for block block_4 #dummy block
end finite element model mesh1
```

At this point we have finished specifying physics-independent quantities. We now want to set up the Adagio procedure and region, along with the time control command block. We start by defining the beginning of the procedure scope, the time control command block, and the beginning of the region scope. Three time stepping command blocks are used in this analysis. The termination time is set to 1.8. Having multiple time blocks is useful because we can make some solver options a function of the time block.

```
begin adagio procedure agio_procedure
  begin time control
    begin time stepping block preload
      start time = 0.0
      begin parameters for adagio region adagio
        number of time steps = 5
      end parameters for adagio region adagio
    end time stepping block preload
```

```
    begin time stepping block slide_1
      start time = 1.0
      begin parameters for adagio region adagio
        number of time steps = 1
      end parameters for adagio region adagio
    end time stepping block slide_1
    begin time stepping block slide_2
      start time = 1.1
      begin parameters for adagio region adagio
        number of time steps = 7
      end parameters for adagio region adagio
    end time stepping block slide_2
    termination time = 1.8
  end time control

  begin adagio region agio_region
```

Next we associate the finite element model we defined above (`mesh1`) with this Adagio region.

```
    use finite element model mesh1
```

Now we define the kinematic boundary conditions. First, we prescribe the displacement of surface_200 which is part of block 4. We fix this surface in both the x and z directions and prescribe the horizontal displacement along the X-direction using the previously defined functions `zero` and `slide`.

```
    ### movement of pencil prescribed by block 4
    begin prescribed displacement
      surface = surface_200
      direction = X
      function = slide
      scale factor = 3.0
    end prescribed displacement
    begin fixed displacement
      surface = surface_200
      components = Y Z
    end fixed displacement
```

Next, we fix the tablet and prevent it from moving in the X-Y plane. We do this on all tablet nodes (nodelist_111 and nodelist_112).

```
    ### Constraints on tablet
    begin fixed displacement
      node set = nodelist_111
      components = X Y
```

```
end fixed displacement
begin fixed displacement
  node set = nodelist_112
  components = X Y
end fixed displacement
```

Finally, we prescribe preload force on the tablet which compresses the tablet against the eraser.

```
### Tablet force
begin prescribed force
  node set = nodelist_111
  direction = Z
  function = tablet_force
  scale factor = 100.0
end prescribed force
```

We now define the contact for this problem. Here we need to define tied contact between block 4 and the eraser. In addition, we have frictional sliding contact between the tablet and the eraser. Two contact block definitions are required; one for the tied contact and one for the frictional contact. The first contact block definition is used for the tied contact between block 4 and the eraser, and the second block is used to define the frictional sliding contact between the eraser and the tablet.

```
### block 4 tied to eraser ###
begin contact definition
  #
  # Set the contact enforcement algorithm.
  enforcement = kinematic
  #
  # define contact surfaces.
  contact surface surf_200 contains surface_200
  contact surface surf_110 contains surface_110
  #
  begin interaction
    master = surf_200
    slave = surf_110
    normal tolerance = 1.0
    tangential tolerance = 0.5e-3
    friction model = tied
  end interaction
end contact definition

begin contact definition
  #
  # Set the contact enforcement algorithm.
  enforcement = kinematic
  #
```

```
  # Define the contact surfaces.
  contact surface surf_11 contains surface_11
  contact surface surf_10 contains surface_10
  #
  begin interaction
    master = surf_11
    slave = surf_10
    normal tolerance = 1.0
    tangential tolerance = 0.5e-3
    capture tolerance = 1.0e-2
    friction model = frict
  end interaction
  begin constant friction model frict
    friction coefficient = 0.8
  end
end contact definition
```

Now we define what variables we want in the output file, as well as how often we want the output file to be written. The output file will be called eraser.e, and it will be an ExodusII file (the database type command is optional; it defaults to ExodusII). The variables we are requesting are the displacements, velocities, and contact diagnostics at the nodes, and stresses and strains at the elements.

```
begin results output output_adagio
  database name = eraser.e
  database type = exodusII
  at step 0, increment = 1
  nodal displacement as displ
  nodal velocity as vel
  nodal contact_tangential_direction
  nodal contact_normal_direction
  nodal contact_accumulated_slip_vector
  nodal contact_status
  nodal contact_normal_traction_magnitude
  nodal contact_tangential_traction_magnitude
  nodal contact_incremental_slip_magnitude
  nodal contact_accumulated_slip
  nodal contact_frictional_energy_density
  nodal contact_area
  element stress as stress
  element log_strain as strain
end results output output_adagio
```

The final part of the input deck is the Adagio solver commands. Because we have sliding contact, we must use the solver command block. Nested inside the solver block we define the control contact block as well as the nonlinear cg solver block.

```
begin solver
  begin control contact name
    target relative residual = 0.001 # default
    maximum iterations = 2000
  end control contact name
  begin cg
    target residual tolerance = 0.0001 # default
    maximum iterations = 2000
    minimum iterations = 1
    begin full tangent preconditioner
      linear solver = feti
    end full tangent preconditioner
  end cg
end solver
```

Now we have defined the Adagio region and procedure blocks and so we close them using the following lines:

```
  end adagio region agio_region
end adagio procedure agio_procedure
```

The final thing that we need to define for this file is the linear solver. In the above solver command block, we included the `full tangent preconditioner` block, which has a nested command line `linear solver = feti`. This command line refers to a linear solver called `feti` that must be defined outside the "Procedure" scope but within the "Sierra" scope and can come at the top of the file prior to the "Procedure" definition or after it as is the case here. The input `feti` on this line command is a user defined string that can have any useful label. The following command block defines the linear solver labeled `feti`. Note that the command block has the label `feti` at the end of the `Begin` line and that this label is referred to from within the above `full tangent preconditioner` command block. The FETI parameters are all set to reasonable values for most problems, so there is no need to set any of them, but if it were necessary to set any of them to non-default values, that would be done within the `feti equation solver` block.

```
begin feti equation solver feti
  # use default settings
end feti equation solver feti
```

Finally, we finish the input file and close the sierra command block:

```
end sierra eraser
```

Appendix C

Command Summary

This appendix gives all of the Sierra/SM commands in the proper scope.

```
# SIERRA scope specification
BEGIN SIERRA <string>name
  #
  # Title
  TITLE = <string list>title

  # Restart commands
  RESTART TIME = <real>restart_time
  RESTART = AUTOMATIC

  # User subroutine file
  USER SUBROUTINE FILE = <string>file name

  # Function definition
  BEGIN DEFINITION FOR FUNCTION <string>function_name
    TYPE = <string>CONSTANT|PIECEWISE LINEAR|PIECEWISE CONSTANT|
      ANALYTIC|PIECEWISE ANALYTIC
    ABSCISSA = <string>abscissa_label
      [scale = <real>abscissa_scale(1.0)]
      [offset = <real>abscissa_offset(0.0)]
    ORDINATE = <string>ordinate_label
      [scale = <real>ordinate_scale(1.0)]
      [offset = <real>ordinate_offset(0.0)]
    X SCALE = <real>x_scale(1.0)
    X OFFSET = <real>x_offset(0.0)
    Y SCALE = <real>y_scale(1.0)
    Y OFFSET = <real>y_offset(0.0)
    BEGIN VALUES
      <real>x_1   <real>y_1
      <real>x_2   <real>y_2
      ...
```

```
    <real>x_n   <real>y_n
  END [VALUES]
  BEGIN EXPRESSIONS
    <real>x_1   <string>analytic_expression_1
    <real>x_2   <string>analytic_expression_2
    ...
    <real>x_n   <string>analytic_expression_n
  END
  AT DISCONTINUITY EVALUATE TO <string>LEFT|RIGHT(LEFT)
  EVALUATE EXPRESSION = <string>analytic_expression1;
    analytic_expression2;...
  DEBUG = ON|OFF(OFF)
END [DEFINITION FOR FUNCTION <string>function_name]

# Definitions

DEFINE POINT <string>point_name WITH COORDINATES
  <real>value_1 <real>value_2 <real>value_3

DEFINE DIRECTION <string>direction_name WITH VECTOR
  <real>value_1 <real>value_2 <real>value_3

DEFINE AXIS <string>axis_name WITH POINT
  <string>point_1 POINT <string>point_2

DEFINE AXIS <string>axis_name WITH POINT
  <string>point_name DIRECTION <string>direction

# Local coordinate system

BEGIN ORIENTATION <string>orientation_name
  SYSTEM = <string>RECTANGULAR|Z RECTANGULAR|CYLINDRICAL|
    SPHERICAL(RECTANGULAR)
  #
  POINT A = <real>global_ax <real>global_ay <real>global_az
  POINT B = <real>global_bx <real>global_by <real>global_bz
  #
  ROTATION ABOUT <integer> 1|2|3(1) = <real>theta(0.0)
END [ORIENTATION <string>orientation_name]

# Rigid bodies

BEGIN RIGID BODY <string>rb_name
  MASS = <real>mass
  POINT MASS = <real>mass [AT <real>X <real>Y <real>Z]
  REFERENCE LOCATION = <real>X <real>Y <real>Z
  INERTIA = <real>Ixx <real>Iyy <real>Izz <real>Ixy
```

```
    <real>Iyz <real>Izx
  POINT INERTIA = <real>Ixx <real>Iyy <real>Izz <real>Ixy
    <real>Iyz <real>Izx
  MAGNITUDE = <real>magnitude_of_velocity
  DIRECTION = <string>direction_definition
  ANGULAR VELOCITY = <real>omega
  CYLINDRICAL AXIS = <string>axis_definition
  INCLUDE NODES IN <string>surface_name
    [if <string>field_name <|<=|=|>=|> <real>value]
END [RIGID BODY <string>rb_name]

# Elastic material

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
  END [PARAMETERS FOR MODEL ELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Thermoelastic material

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
```

```
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL THERMOELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
  END [PARAMETERS FOR MODEL THERMOELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# neo-Hookean material

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL NEO_HOOKEAN
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
  END [PARAMETERS FOR MODEL NEO_HOOKEAN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic fracture material

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
```

```
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC_FRACTURE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    MAX STRESS = <real>max_stress
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
  END [PARAMETERS FOR MODEL ELASTIC_FRACTURE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic-plastic material

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ELASTIC_PLASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING MODULUS = <real>hardening_modulus
    BETA = <real>beta_parameter(1.0)
  END [PARAMETERS FOR MODEL ELASTIC_PLASTIC]
```

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic-plastic power-law hardening

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL EP_POWER_HARD
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN = <real>luders_strain
  END [PARAMETERS FOR MODEL EP_POWER_HARD]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic plastic power-law hardening with failure

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
```

```
  BEGIN PARAMETERS FOR MODEL DUCTILE_FRACTURE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    HARDENING CONSTANT = <real>hardening_constant
    HARDENING EXPONENT = <real>hardening_exponent
    LUDERS STRAIN <real>luders_strain
    CRITICAL TEARING PARAMETER = <real>crit_tearing
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
  END [PARAMETERS FOR MODEL DUCTILE_FRACTURE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Multilinear elastic plastic

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL MULTILINEAR_EP
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    BETA = <real>beta_parameter(1.0)
    HARDENING FUNCTION = <string>hardening_function_name
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
    YIELD STRESS FUNCTION =
      <string>yield_stress_function_name
  END [PARAMETERS FOR MODEL MULTILINEAR_EP]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]
```

```
# Multilinear elastic plastic with failure

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ML_EP_FAIL
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    BETA = <real>beta_parameter(1.0)
    HARDENING FUNCTION = <string>hardening_function_name
    YOUNGS MODULUS FUNCTION = <string>ym_function_name
    POISSONS RATIO FUNCTION = <string>pr_function_name
    YIELD STRESS FUNCTION =
      <string>yield_stress_function_name
    CRITICAL TEARING PARAMETER = <real>crit_tearing
    CRITICAL CRACK OPENING STRAIN = <real>critical_strain
    CRITICAL BIAXIALITY RATIO = <real>critical_ratio(0.0)
  END [PARAMETERS FOR MODEL ML_EP_FAIL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# BCJ plasticity

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
```

```
  <string>thermal_strain_y_function
THERMAL STRAIN Z FUNCTION =
  <string>thermal_strain_z_function
#
BEGIN PARAMETERS FOR MODEL BCJ
  YOUNGS MODULUS = <real>youngs_modulus
  POISSONS RATIO = <real>poissons_ratio
  SHEAR MODULUS = <real>shear_modulus
  BULK MODULUS = <real>bulk_modulus
  LAMBDA = <real>lambda
  C1 = <real>c1
  C2 = <real>c2
  C3 = <real>c3
  C4 = <real>c4
  C5 = <real>c5
  C6 = <real>c6
  C7 = <real>c7
  C8 = <real>c8
  C9 = <real>c9
  C10 = <real>c10
  C11 = <real>c11
  C12 = <real>c12
  C13 = <real>c13
  C14 = <real>c14
  C15 = <real>c15
  C16 = <real>c16
  C17 = <real>c17
  C18 = <real>c18
  C19 = <real>c19
  C20 = <real>c20
  DAMAGE EXPONENT = <real>damage_exponent
  INITIAL ALPHA_XX = <real>alpha_xx
  INITIAL ALPHA_YY = <real>alpha_yy
  INITIAL ALPHA_ZZ = <real>alpha_zz
  INITIAL ALPHA_XY = <real>alpha_xy
  INITIAL ALPHA_YZ = <real>alpha_yz
  INITIAL ALPHA_XZ = <real>alpha_xz
  INITIAL KAPPA = <real>initial_kappa
  INITIAL DAMAGE = <real>initial_damage
  YOUNGS MODULUS FUNCTION = <string>ym_function_name
  POISSONS RATIO FUNCTION = <string>pr_function_name
  SPECIFIC HEAT = <real>specific_heat
  THETA OPT = <integer>theta_opt
  FACTOR = <real>factor
  RHO = <real>rho
  TEMP0 = <real>temp0
END [PARAMETERS FOR MODEL BCJ]
```

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Power law creep

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL POWER_LAW_CREEP
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CREEP CONSTANT = <real>creep_constant
    CREEP EXPONENT = <real>creep_exponent
    THERMAL CONSTANT = <real>thermal_constant
  END [PARAMETERS FOR MODEL POWER_LAW_CREEP]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Soil and crushable foam

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL SOIL_FOAM
```

```
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A0 = <real>const_coeff_yieldsurf
    A1 = <real>lin_coeff_yieldsurf
    A2 = <real>quad_coeff_yieldsurf
    PRESSURE CUTOFF = <real>pressure_cutoff
    PRESSURE FUNCTION = <string>function_press_volstrain
  END [PARAMETERS FOR MODEL SOIL_FOAM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name

# Foam plasticity

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL FOAM_PLASTICITY
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    PHI = <real>phi
    SHEAR STRENGTH  = <real>shear_strength
    SHEAR HARDENING = <real>shear_hardening
    SHEAR EXPONENT  = <real>shear_exponent
    HYDRO STRENGTH  = <real>hydro_strength
    HYDRO HARDENING = <real>hydro_hardening
    HYDRO EXPONENT  = <real>hydro_exponent
    BETA = <real>beta
  END [PARAMETERS FOR MODEL FOAM_PLASTICITY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Low density foam
```

```
BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL LOW_DENSITY_FOAM
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    A = <real>A
    B = <real>B
    C = <real>C
    NAIR  = <real>NAir
    P0  = <real>P0
    PHI = <real>Phi
  END [PARAMETERS FOR MODEL LOW_DENSITY_FOAM]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic three-dimensional orthotropic

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  BEGIN PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC
    # general parameters (any two are required)
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    # required parameters
    YOUNGS MODULUS AA = <real>Eaa_value
    YOUNGS MODULUS BB = <real>Ebb_value
    YOUNGS MODULUS CC = <real>Ecc_value
    POISSONS RATIO AB = <real>NUab_value
```

```
    POISSONS RATIO BC = <real>NUbc_value
    POISSONS RATIO CA = <real>NUca_value
    SHEAR MODULUS AB = <real>Gab_value
    SHEAR MODULUS BC = <real>Gbc_value
    SHEAR MODULUS CA = <real>Gca_value
    COORDINATE SYSTEM = <string>coordinate_system_name
    DIRECTION FOR ROTATION = <real>1|2|3
    ALPHA = <real>alpha_in_degrees
    SECOND DIRECTION FOR ROTATION = <real>1|2|3
    SECOND ALPHA = <real>second_alpha_in_degrees
    THERMAL STRAIN AA FUNCTION = <string>ethaa_function_name
    THERMAL STRAIN BB FUNCTION = <string>ethbb_function_name
    THERMAL STRAIN CC FUNCTION = <string>ethcc_function_name
  END [PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Wire mesh

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL WIRE_MESH
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD FUNCTION = <string>yield_function
    TENSION = <real>tensile_strength
  END [PARAMETERS FOR MODEL WIRE_MESH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Orthotropic crush

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
```

```
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    EX  = <real>modulus_x
    EY  = <real>modulus_y
    EZ  = <real>modulus_z
    GXY = <real>shear_modulus_xy
    GYZ = <real>shear_modulus_yz
    GZX = <real>shear_modulus_zx
    VMIN = <real>min_crush_volume
    CRUSH XX = <string>stress_volume_xx_function_name
    CRUSH YY = <string>stress_volume_yy_function_name
    CRUSH ZZ = <string>stress_volume_zz_function_name
    CRUSH XY =
      <string>shear_stress_volume_xy_function_name
    CRUSH YZ =
      <string>shear_stress_volume_yz_function_name
    CRUSH ZX =
      <string>shear_stress_volume_zx_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Orthotropic rate

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
```

```
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL ORTHOTROPIC_RATE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    YIELD STRESS = <real>yield_stress
    MODULUS TTTT = <real>modulus_tttt
    MODULUS TTLL = <real>modulus_ttll
    MODULUS TTWW = <real>modulus_ttww
    MODULUS LLLL = <real>modulus_llll
    MODULUS LLWW = <real>modulus_llww
    MODULUS WWWW = <real>modulus_wwww
    MODULUS TLTL = <real>modulus_tltl
    MODULUS LWLW = <real>modulus_lwlw
    MODULUS WTWT = <real>modulus_wtwt
    TX = <real>tx
    TY = <real>ty
    TZ = <real>tz
    LX = <real>lx
    LY = <real>ly
    LZ = <real>lz
    MODULUS FUNCTION = <string>modulus_function_name
    RATE FUNCTION = <string>rate_function_name
    T FUNCTION = <string>t_function_name
    L FUNCTION = <string>l_function_name
    W FUNCTION = <string>w_function_name
    TL FUNCTION = <string>tl_function_name
    LW FUNCTION = <string>lw_function_name
    WT FUNCTION = <string>wt_function_name
  END [PARAMETERS FOR MODEL ORTHOTROPIC_RATE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Elastic laminate

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BEGIN PARAMETERS FOR MODEL ELASTIC_LAMINATE
    A11 = <real>a11_value
    A12 = <real>a12_value
```

```
    A16 = <real>a16_value
    A22 = <real>a22_value
    A26 = <real>a26_value
    A66 = <real>a66_value
    A44 = <real>a44_value
    A45 = <real>a45_value
    A55 = <real>a55_value
    B11 = <real>b11_value
    B12 = <real>b12_value
    B16 = <real>b16_value
    B22 = <real>b22_value
    B26 = <real>b26_value
    B66 = <real>b66_value
    D11 = <real>d11_value
    D12 = <real>d12_value
    D16 = <real>d16_value
    D22 = <real>d22_value
    D26 = <real>d26_value
    D66 = <real>d66_value
    COORDINATE SYSTEM = <string>coord_sys_name
    DIRECTION FOR ROTATION = 1|2|3
    ALPHA = <real>alpha_value_in_degrees
    THETA = <real>theta_value_in_degrees
    NTH11 FUNCTION = <string>nth11_function_name
    NTH22 FUNCTION = <string>nth22_function_name
    NTH12 FUNCTION = <string>nth12_function_name
    MTH11 FUNCTION = <string>mth11_function_name
    MTH22 FUNCTION = <string>mth22_function_name
    MTH12 FUNCTION = <string>mth12_function_name
  END [PARAMETERS FOR MODEL ELASTIC_LAMINATE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Fiber membrane

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
```

```
  BEGIN PARAMETERS FOR MODEL FIBER_MEMBRANE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CORD DENSITY = <real>cord_density
    CORD DIAMETER = <real>cord_diameter
    MATRIX DENSITY = <real>matrix_density
    TENSILE TEST FUNCTION = <string>test_function_name
    PERCENT CONTINUUM = <real>percent_continuum
    EPL = <real>epl
    AXIS X = <real>axis_x
    AXIS Y = <real>axis_y
    AXIS Z = <real>axis_z
    MODEL = <string>RECTANGULAR
    STIFFNESS SCALE = <real>stiffness_scale
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL FIBER_MEMBRANE]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Fiber shell

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL FIBER_SHELL
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    CORD DENSITY = <real>cord_density
    CORD DIAMETER = <real>cord_diameter
    MATRIX DENSITY = <real>matrix_density
    TENSILE TEST FUNCTION = <string>test_function_name
    PERCENT CONTINUUM = <real>percent_continuum
```

```
    EPL = <real>epl
    AXIS X = <real>axis_x
    AXIS Y = <real>axis_y
    AXIS Z = <real>axis_z
    MODEL = <string>RECTANGULAR
    ALPHA1 = <real>alpha1
    ALPHA2 = <real>alpha2
    ALPHA3 = <real>alpha3
    STIFFNESS SCALE = <real>stiffness_scale
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL FIBER_SHELL]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Incompressible solid

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    K SCALING = <real>k_scaling
    2G SCALING = <real>2g_scaling
    TARGET E = <real>target_e
    MAX POISSONS RATIO = <real>max_poissons_ratio
    REFERENCE STRAIN = <real>reference_strain
    SCALING FUNCTION = <string>scaling_function_name
  END [PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Mooney Rivlin

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
```

```
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL MOONEY_RIVLIN
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    C10 = <real>c10
    C01 = <real>c01
    C10 FUNCTION = <string>c10_function_name
    C01 FUNCTION = <string>c01_function_name
    BULK FUNCTION = <string>bulk_function_name
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
    MAX POISSONS RATIO = <real>max_poissons_ratio
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL MOONEY_RIVLIN]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# NLVE 3D Orthotropic

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL NLVE_3D_ORTHOTROPIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    SHEAR MODULUS = <real>shear_modulus
    BULK MODULUS = <real>bulk_modulus
    LAMBDA = <real>lambda
    COORDINATE SYSTEM = <string>coordinate_system_name
    DIRECTION FOR ROTATION = <real>1|2|3
    ALPHA = <real>alpha_in_degrees
    SECOND DIRECTION FOR ROTATION = <real>1|2|3
    SECOND ALPHA = <real>second_alpha_in_degrees
    FICTITIOUS LOGA FUNCTION = <string>fict_loga_function_name
    FICTITIOUS LOGA SCALE FACTOR = <real>fict_loga_scale_factor
    # In each of the five ‘‘PRONY’’ command lines and in
    # the RELAX TIME command line, the value of i can be from
    # 1 through 30
    1PSI PRONY <integer>i = <real>psi1_i
    2PSI PRONY <integer>i = <real>psi2_i
    3PSI PRONY <integer>i = <real>psi3_i
```

```
4PSI PRONY <integer>i = <real>psi4_i
5PSI PRONY <integer>i = <real>psi5_i
RELAX TIME <integer>i = <real>tau_i
REFERENCE TEMP = <real>tref
REFERENCE DENSITY = <real>rhoref
WLF C1 = <real>wlf_c1
WLF C2 = <real>wlf_c2
B SHIFT CONSTANT = <real>b_shift
SHIFT REF VALUE = <real>shift_ref
WWBETA 1PSI = <real>wwb_1psi
WWTAU 1PSI = <real>wwt_1psi
WWBETA 2PSI = <real>wwb_2psi
WWTAU 2PSI = <real>wwt_2psi
WWBETA 3PSI = <real>wwb_3psi
WWTAU 3PSI = <real>wwt_3psi
WWBETA 4PSI = <real>wwb_4psi
WWTAU 4PSI = <real>wwt_4psi
WWBETA 5PSI = <real>wwb_5psi
WWTAU 5PSI = <real>wwt_5psi
DOUBLE INTEG FACTOR = <real>dble_int_fac
REF RUBBERY HCAPACITY = <real>hcapr
REF GLASSY HCAPACITY = <real>hcapg
GLASS TRANSITION TEM = <real>tg
REF GLASSY C11 = <real>c11g
REF RUBBERY C11 = <real>c11r
REF GLASSY C22 = <real>c22g
REF RUBBERY C22 = <real>c22r
REF GLASSY C33 = <real>c33g
REF RUBBERY C33 = <real>c33r
REF GLASSY C12 = <real>c12g
REF RUBBERY C12 = <real>c12r
REF GLASSY C13 = <real>c13g
REF RUBBERY C13 = <real>c13r
REF GLASSY C23 = <real>c23g
REF RUBBERY C23 = <real>c23r
REF GLASSY C44 = <real>c44g
REF RUBBERY C44 = <real>c44r
REF GLASSY C55 = <real>c55g
REF RUBBERY C55 = <real>c55r
REF GLASSY C66 = <real>c66g
REF RUBBERY C66 = <real>c66r
REF GLASSY CTE1 = <real>cte1g
REF RUBBERY CTE1 = <real>cte1r
REF GLASSY CTE2 = <real>cte2g
REF RUBBERY CTE2 = <real>cte2r
REF GLASSY CTE3 = <real>cte3g
REF RUBBERY CTE3 = <real>cte3r
```

```
LINEAR VISCO TEST = <real>lvt
T DERIV GLASSY C11 = <real>dc11gdT
T DERIV RUBBERY C11 = <real>dc11rdT
T DERIV GLASSY C22 = <real>dc22gdT
T DERIV RUBBERY C22 = <real>dc22rdT
T DERIV GLASSY C33 = <real>dc33gdT
T DERIV RUBBERY C33 = <real>dc33rdT
T DERIV GLASSY C12 = <real>dc12gdT
T DERIV RUBBERY C12 = <real>dc12rdT
T DERIV GLASSY C13 = <real>dc13gdT
T DERIV RUBBERY C13 = <real>dc13rdT
T DERIV GLASSY C23 = <real>dc23gdT
T DERIV RUBBERY C23 = <real>dc23rdT
T DERIV GLASSY C44 = <real>dc44gdT
T DERIV RUBBERY C44 = <real>dc44rdT
T DERIV GLASSY C55 = <real>dc55gdT
T DERIV RUBBERY C55 = <real>dc55rdT
T DERIV GLASSY C66 = <real>dc66gdT
T DERIV RUBBERY C66 = <real>dc66rdT
T DERIV GLASSY CTE1 = <real>dcte1gdT
T DERIV RUBBERY CTE1 = <real>dcte1rdT
T DERIV GLASSY CTE2 = <real>dcte2gdT
T DERIV RUBBERY CTE2 = <real>dcte2rdT
T DERIV GLASSY CTE3 = <real>dcte3gdT
T DERIV RUBBERY CTE3 = <real>dcte3rdT
T DERIV GLASSY HCAPACITY = <real>dhcapgdT
T DERIV RUBBERY HCAPACITY = <real>dhcaprdT
REF PSIC = <real>psic_ref
T DERIV PSIC = <real>dpsicdT
T 2DERIV PSIC = <real>d2psicdT2
PSI EQ 2T = <real>psitt
PSI EQ 3T = <real>psittt
PSI EQ 4T = <real>psitttt
PSI EQ XX 11 = <real>psiXX11
PSI EQ XX 22 = <real>psiXX22
PSI EQ XX 33 = <real>psiXX33
PSI EQ XX 12 = <real>psiXX12
PSI EQ XX 13 = <real>psiXX13
PSI EQ XX 23 = <real>psiXX23
PSI EQ XX 44 = <real>psiXX44
PSI EQ XX 55 = <real>psiXX55
PSI EQ XX 66 = <real>psiXX66
PSI EQ XXT 11 = <real>psiXXT11
PSI EQ XXT 22 = <real>psiXXT22
PSI EQ XXT 33 = <real>psiXXT33
PSI EQ XXT 12 = <real>psiXXT12
PSI EQ XXT 13 = <real>psiXXT13
```

```
PSI EQ XXT 23 = <real>psiXXT23
PSI EQ XXT 44 = <real>psiXXT44
PSI EQ XXT 55 = <real>psiXXT55
PSI EQ XXT 66 = <real>psiXXT66
PSI EQ XT 1 = <real>psiXT1
PSI EQ XT 2 = <real>psiXT2
PSI EQ XT 3 = <real>psiXT3
PSI EQ XTT 1 = <real>psiXTT1
PSI EQ XTT 2 = <real>psiXTT2
PSI EQ XTT 3 = <real>psiXTT3
REF PSIA 11 = <real>psiA11
REF PSIA 22 = <real>psiA22
REF PSIA 33 = <real>psiA33
REF PSIA 12 = <real>psiA12
REF PSIA 13 = <real>psiA13
REF PSIA 23 = <real>psiA23
REF PSIA 44 = <real>psiA44
REF PSIA 55 = <real>psiA55
REF PSIA 66 = <real>psiA66
T DERIV PSIA 11 = <real>dpsiA11dT
T DERIV PSIA 22 = <real>dpsiA22dT
T DERIV PSIA 33 = <real>dpsiA33dT
T DERIV PSIA 12 = <real>dpsiA12dT
T DERIV PSIA 13 = <real>dpsiA13dT
T DERIV PSIA 23 = <real>dpsiA23dT
T DERIV PSIA 44 = <real>dpsiA44dT
T DERIV PSIA 55 = <real>dpsiA55dT
T DERIV PSIA 66 = <real>dpsiA66dT
REF PSIB 1 = <real>psiB1
REF PSIB 2 = <real>psiB2
REF PSIB 3 = <real>psiB3
T DERIV PSIB 1 = <real>dpsiB1dT
T DERIV PSIB 2 = <real>dpsiB2dT
T DERIV PSIB 3 = <real>dpsiB3dT
PSI POT TT = <real>psipotTT
PSI POT TTT = <real>psipotTTT
PSI POT TTTT = <real>psipotTTTT
PSI POT XT 1 = <real>psipotXT1
PSI POT XT 2 = <real>psipotXT2
PSI POT XT 3 = <real>psipotXT3
PSI POT XTT 1 = <real>psipotXTT1
PSI POT XTT 2 = <real>psipotXTT2
PSI POT XTT 3 = <real>psipotXTT3
PSI POT XXT 11 = <real>psipotXXT11
PSI POT XXT 22 = <real>psipotXXT22
PSI POT XXT 33 = <real>psipotXXT33
PSI POT XXT 12 = <real>psipotXXT12
```

```
    PSI POT XXT 13 = <real>psipotXXT13
    PSI POT XXT 23 = <real>psipotXXT23
    PSI POT XXT 44 = <real>psipotXXT44
    PSI POT XXT 55 = <real>psipotXXT55
    PSI POT XXT 66 = <real>psipotXXT66
  END [PARAMETERS FOR MODEL NLVE_3D_ORTHOTROPIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Stiff elastic

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  # thermal strain option
  THERMAL STRAIN FUNCTION = <string>thermal_strain_function
  # or all three of the following
  THERMAL STRAIN X FUNCTION =
    <string>thermal_strain_x_function
  THERMAL STRAIN Y FUNCTION =
    <string>thermal_strain_y_function
  THERMAL STRAIN Z FUNCTION =
    <string>thermal_strain_z_function
  #
  BEGIN PARAMETERS FOR MODEL STIFF_ELASTIC
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    SCALE FACTOR = <real>scale_factor
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL STIFF_ELASTIC]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Swanson

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL SWANSON
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
```

```
    A1 = <real>a1
    P1 = <real>p1
    B1 = <real>b1
    Q1 = <real>q1
    C1 = <real>c1
    R1 = <real>r1
    CUT OFF STRAIN = <real>ecut
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
    MAX POISSONS RATIO = <real>max_poissons_ratio
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL SWANSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Viscoelastic Swanson

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  BIOTS COEFFICIENT = <real>biots_value
  #
  BEGIN PARAMETERS FOR MODEL VISCOELASTIC_SWANSON
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
    BULK MODULUS = <real>bulk_modulus
    SHEAR MODULUS = <real>shear_modulus
    LAMBDA = <real>lambda
    A1 = <real>a1
    P1 = <real>p1
    B1 = <real>b1
    Q1 = <real>q1
    C1 = <real>c1
    R1 = <real>r1
    CUT OFF STRAIN = <real>ecut
    THERMAL EXPANSION FUNCTION = <string>eth_function_name
    PRONY SHEAR INFINITY = <real>ginf
    PRONY SHEAR 1 = <real>g1
    PRONY SHEAR 2 = <real>g2
    PRONY SHEAR 3 = <real>g3
    PRONY SHEAR 4 = <real>g4
    PRONY SHEAR 5 = <real>g5
    PRONY SHEAR 6 = <real>g6
    PRONY SHEAR 7 = <real>g7
    PRONY SHEAR 8 = <real>g8
    PRONY SHEAR 9 = <real>g9
    PRONY SHEAR 10 = <real>g10
    SHEAR RELAX TIME 1 = <real>tau1
```

```
    SHEAR RELAX TIME 2 = <real>tau2
    SHEAR RELAX TIME 3 = <real>tau3
    SHEAR RELAX TIME 4 = <real>tau4
    SHEAR RELAX TIME 5 = <real>tau5
    SHEAR RELAX TIME 6 = <real>tau6
    SHEAR RELAX TIME 7 = <real>tau7
    SHEAR RELAX TIME 8 = <real>tau8
    SHEAR RELAX TIME 9 = <real>tau9
    SHEAR RELAX TIME 10 = <real>tau10
    WLF COEF C1 = <real>wlf_c1
    WLF COEF C2 = <real>wlf_c2
    WLF TREF = <real>wlf_tref
    NUMERICAL SHIFT FUNCTION = <string>ns_function_name
    TARGET E = <real>target_e
    TARGET E FUNCTION = <string>etar_function_name
    MAX POISSONS RATIO = <real>max_poissons_ratio
    REFERENCE STRAIN = <real>reference_strain
  END [PARAMETERS FOR MODEL VISCOELASTIC_SWANSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Traction Decay

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL TRACTION_DECAY
    NORMAL DECAY LENGTH = <real>
    TANGENTIAL DECAY LENGTH = <real>
  END [PARAMETERS FOR MODEL TRACTION_DECAY]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Tvergaard Hutchinson

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON
    INIT TRACTION METHOD = IGNORE|ADD (IGNORE)
    LAMBDA_1 = <real>
    LAMBDA_2 = <real>
    NORMAL LENGTH SCALE = <real>
    TANGENTIAL LENGTH SCALE = <real>
    PEAK TRACTION = <real>
    PENETRATION STIFFNESS MULTIPLIER = <real>
    NORMAL INITIAL TRACTION DECAY LENGTH = <real>
    TANGENTIAL INITIAL TRACTION DECAY LENGTH = <real>
    USE ELASTIC UNLOADING = NO|YES (YES)
```

```
  END [PARAMETERS FOR MODEL TVERGAARD_HUTCHINSON]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Thouless Parmigiani

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL THOULESS_PARMIGIANI
    INIT TRACTION METHOD = IGNORE|ADD (IGNORE)
    LAMBDA_1_N = <real>
    LAMBDA_2_N = <real>
    LAMBDA_1_T = <real>
    LAMBDA_2_T = <real>
    NORMAL LENGTH SCALE = <real>
    TANGENTIAL LENGTH SCALE = <real>
    PEAK NORMAL TRACTION = <real>
    PEAK TANGENTIAL TRACTION = <real>
    PENETRATION STIFFNESS MULTIPLIER = <real>
    USE ELASTIC UNLOADING = NO|YES (YES)
  END [PARAMETERS FOR MODEL THOULESS_PARMIGIANI]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Compliant Joint

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value

  BEGIN COMPLIANT JOINT MODEL <string>name
    NORMAL MODULUS = <real>norm_mod
    SHEAR MODULUS = <real>shear_mod
    APERTURE = <real>aperture
    APERTURE LIMIT = <real>aperture_limit
    FRICTION COEFFICIENT = <real>fric_coef
    COHESION = <real>cohesion
  END [PARAMETERS FOR MODEL COMPLIANT_JOINT]
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# RVE

BEGIN PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name
  DENSITY = <real>density_value
  #
  BEGIN PARAMETERS FOR MODEL RVE
    YOUNGS MODULUS = <real>youngs_modulus
    POISSONS RATIO = <real>poissons_ratio
  END [PARAMETERS FOR MODEL RVE]
```

```
END [PROPERTY SPECIFICATION FOR MATERIAL <string>mat_name]

# Define mesh

BEGIN FINITE ELEMENT MODEL <string>mesh_descriptor
  DATABASE NAME = <string>mesh_file_name
  DATABASE TYPE = <string>database_type(exodusII)
  ALIAS <string>mesh_identifier AS <string>user_name
  OMIT BLOCK <string>block_list
  COMPONENT SEPARATOR CHARACTER = <string>separator
  BEGIN PARAMETERS FOR BLOCK [<string list>block_names]
    MATERIAL <string>material_name
    SOLID MECHANICS USE MODEL <string>model_name
    INCLUDE ALL BLOCKS
    REMOVE BLOCK = <string list>block_names
    SECTION = <string>section_id
    LINEAR BULK VISCOSITY =
      <real>linear_bulk_viscosity_value(0.06)
    QUADRATIC BULK VISCOSITY =
      <real>quad_bulk_viscosity_value(1.20)
    HOURGLASS STIFFNESS =
      <real>hour_glass_stiff_value(solid = 0.05,
      shell/membrane = 0.0)
    HOURGLASS VISCOSITY =
      <real>hour_glass_visc_value(solid = 0.0,
      shell/membrane = 0.0)
    MEMBRANE HOURGLASS STIFFNESS =
      <real>memb_hour_glass_stiff_value(0.0)
    MEMBRANE HOURGLASS VISCOSITY =
      <real>memb_hour_glass_visc_value(0.0)
    BENDING HOURGLASS STIFFNESS =
      <real>bend_hour_glass_stiff_value(0.0)
    BENDING HOURGLASS VISCOSITY =
      <real>bend_hour_glass_visc_value(0.0)
    TRANSVERSE SHEAR HOURGLASS STIFFNESS =
      <real>tshr_hour_glass_stiff_value(0.0)
    TRANSVERSE SHEAR HOURGLASS VISCOSITY =
      <real>tshr_hour_glass_visc_value(0.0)
    EFFECTIVE MODULI MODEL = <string>PRESTO|PRONTO|
      CURRENT|ELASTIC(PRONTO)
    ELEMENT NUMERICAL FORMULATION = <string>OLD|NEW(OLD)
    ACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
      <string list>period_names
    INACTIVE FOR PROCEDURE <string>proc_name DURING PERIODS
      <string list>period_names
  END [PARAMETERS FOR BLOCK <string list>block_names]
END [FINITE ELEMENT MODEL <string>mesh_descriptor]
```

Margin markers: *Explicit* (beside LINEAR BULK VISCOSITY, QUADRATIC BULK VISCOSITY, ELEMENT NUMERICAL FORMULATION)

```
# Element sections

BEGIN SOLID SECTION <string>solid_section_name
  COORDINATE SYSTEM = <string>Coordinate_system_name
  FORMULATION = <string>MEAN_QUADRATURE|
    SELECTIVE_DEVIATORIC|FULLY_INTEGRATED|
    VOID(MEAN_QUADRATURE)
  DEVIATORIC PARAMETER = <real>deviatoric_param
  STRAIN INCREMENTATION = <string>MIDPOINT_INCREMENT|
    STRONGLY_OBJECTIVE|NODE_BASED(MIDPOINT_INCREMENT)
  NODE BASED ALPHA FACTOR = <real>bulk_stress_weight(0.01)
  NODE BASED BETA FACTOR = <real>shear stress_weight(0.35)
  NODE BASED STABILIZATION METHOD = <string>EFFECTIVE_MODULI|
    MATERIAL(MATERIAL)
  HOURGLASS FORMULATION = <string>TOTAL|INCREMENTAL(INCREMENTAL)
  HOURGLASS INCREMENT = <string>ENDSTEP|MIDSTEP (ENDSTEP)
  HOURGLASS ROTATION = <string> APPROXIMATE|SCALED (APPROXIMATE)
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [SOLID SECTION <string>solid_section_name]

BEGIN COHESIVE SECTION <string>cohesive_section_name
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points(1)
END [COHESIVE SECTION <string>cohesive_section_name]

BEGIN SHELL SECTION <string>shell_section_name
  THICKNESS = <real>shell_thickness
  THICKNESS MESH VARIABLE =
    <string>THICKNESS|<string>var_name
  THICKNESS TIME STEP = <real>time_value
  THICKNESS SCALE FACTOR = <real>thick_scale_factor(1.0)
  INTEGRATION RULE = TRAPEZOID|GAUSS|LOBATTO|SIMPSONS|
    USER|ANALYTIC|DEFAULT(DEFAULT)
  NUMBER OF INTEGRATION POINTS = <integer>num_int_points(5)
  FORMULATION = KH_SHELL|BT_SHELL|NQUAD(BT_SHELL)
  BEGIN USER INTEGRATION RULE
    <real>location_1 <real>weight_1
    <real>location_2 <real>weight_2
    .
    .
    <real>location_n <real>weight_n
  END [USER INTEGRATION RULE]
  LOFTING FACTOR = <real>lofting_factor(0.5)
  OFFSET MESH VARIABLE = <string>var_name
  ORIENTATION = <string>orientation_name
```

```
  DRILLING STIFFNESS FACTOR = <real>stiffness_factor(1.0e-5)
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [SHELL SECTION <string>shell_section_name]

BEGIN MEMBRANE SECTION <string>membrane_section_name
  THICKNESS = <real>mem_thickness
  THICKNESS MESH VARIABLE =
    <string>THICKNESS|<string>var_name
  THICKNESS TIME STEP = <real>time_value
  THICKNESS SCALE FACTOR = <real>thick_scale_factor(1.0)
  FORMULATION = <string>MEAN_QUADRATURE|
    SELECTIVE_DEVIATORIC(MEAN QUADRATURE)
  DEVIATORIC PARAMETER = <real>deviatoric_param
  LOFTING FACTOR = <real>lofting_factor(0.5)
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [MEMBRANE SECTION <string>membrane_section_name]

BEGIN BEAM SECTION <string>beam_section_name
  SECTION = <string>ROD|TUBE|BAR|BOX|I
  WIDTH = <real>section_width
  WIDTH VARIABLE = <string>width_var
  HEIGHT = <real>section_width
  HEIGHT VARIABLE= <string>height_var
  WALL THICKNESS = <real>wall_thickness
  WALL THICKNESS VARIABLE = <string>wall_thickness_var
  FLANGE THICKNESS = <real>flange_thickness
  FLANGE THICKNESS VARIABLE = <string>flange_thickness_var
  T AXIS = <real>tx <real>ty <real>tz
  T AXIS VARIABLE = <string>t_axis_var
  REFERENCE AXIS = <string>CENTER|RIGHT|
    TOP|LEFT|BOTTOM(CENTER)
  AXIS OFFSET = <real>s_offset <real>t_offset
  AXIS OFFSET GLOBAL = <real>x_offset <real>y_offset <real>z_offset
  AXIS OFFSET VARIABLE = <string>axis_offset_var
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [BEAM SECTION <string>beam_section_name]

BEGIN TRUSS SECTION <string>truss_section_name
  AREA = <real>cross_sectional_area
  INITIAL LOAD = <real>initial_load
  PERIOD = <real>period
```

```
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
END [TRUSS SECTION <string>truss_section_name]
```

Explicit

```
BEGIN SPRING SECTION <string>spring_section_name
  FORCE STRAIN FUNCTION = <string>force_strain_function
  DEFAULT STIFFNESS <real>default_stiffness
  PRELOAD = <real>preload_value
  PRELOAD DURATION = <real>preload_duration
  RESET INITIAL LENGTH AFTER PRELOAD = <string>NO|YES
  MASS PER UNIT LENGTH = <real>mass_per_unit_length
END [SPRING SECTION <string>spring_section_name]
```

Explicit

```
BEGIN DAMPER SECTION <string>damper_section_name
  AREA = <real>damper_cross_sectional_area
END [DAMPER SECTION <string>damper_section_name]
```

Explicit

```
BEGIN POINT MASS SECTION <string>pointmass_section_name
  VOLUME = <real>volume
  MASS = <real>mass
  IXX = <real>Ixx
  IYY = <real>Iyy
  IZZ = <real>Izz
  IXY = <real>Ixy
  IXZ = <real>Ixz
  IYZ = <real>Iyz
  RIGID BODY = <string>rigid_body_name
  RIGID BODIES FROM ATTRIBUTES = <integer>first_id
    TO <integer>last_id
  MASS VARIABLE = <string>mass_variable_name
  INERTIA VARIABLE = <string>inertia_variable_name
  OFFSET VARIABLE = <string>offset_variable_name
  ATTRIBUTES VARIABLE NAME = <string>attrib_variable_name
END [POINT MASS SECTION <string>pointmass_section_name]
```

Explicit

```
BEGIN PARTICLE SECTION <string>sph_section_name
  RADIUS MESH VARIABLE =
    <string>var_name|<string>attribute|SPHERE INITIAL
    RADIUS = <real>rad
  RADIUS MESH VARIABLE TIME STEP = <string>time
  PROBLEM DIMENSION = <integer>1|2|3(3)
  CONSTANT SPHERE RADIUS
  FINAL RADIUS MULTIPLICATION FACTOR = <real>factor(1.0)
  FORMULATION = <string>MASS_PARTICLE|SPH(SPH)
  MONAGHAN EPSILON = <real>monaghan_epsilon(0.0)
  MONAGHAN N = <real>monaghan_n(0.0)
```

```
  SPH ALPHAQ PARAMETER = <real>alpha(1.0)
  SPH BETAQ PARAMETER = <real>beta(2.0)
  DENSITY FORMULATION = <string>MATERIAL|KERNEL(MATERIAL)
END [PARTICLE SECTION <string>sph_section_name]

BEGIN SUPERELEMENT SECTION <string>section_name
  BEGIN MAP
    <integer>node_index_1 <integer>component_index_1
    <integer>node_index_2 <integer>component_index_2
    ...
    <integer>node_index_n <integer>component_index_n
  END
  BEGIN STIFFNESS MATRIX
    <real>k_1_1 <real>k_1_2 ...  <real>k_1_n
    <real>k_2_1 <real>k_2_2 ...  <real>k_2_n
    ...         ...         ...  ...
    <real>k_n_1 <real>k_n_2 ...  <real>k_n_n
  END
  BEGIN DAMPING MATRIX
    <real>c_1_1 <real>c_1_2 ...  <real>c_1_n
    <real>c_2_1 <real>c_2_2 ...  <real>c_2_n
    ...         ...         ...  ...
    <real>c_n_1 <real>c_n_2 ...  <real>c_n_n
  END
  BEGIN MASS MATRIX
    <real>m_1_1 <real>m_1_2 ...  <real>m_1_n
    <real>m_2_1 <real>m_2_2 ...  <real>m_2_n
    ...          ...        ...  ...
    <real>m_n_1 <real>m_n_2 ...  <real>m_n_n
  END
  FILE = <string>netcdf_file_name
END [SUPERELEMENT SECTION <string>section_name]
```

Explicit

```
 BEGIN PERIDYNAMICS SECTION <string>section_name
   MATERIAL MODEL FORMULATION = <string>CLASSICAL|PERIDYNAMICS
   HORIZON = <real>horizon [SCALE BY ELEMENT RADIUS]
   INFLUENCE FUNCTION = <string>influence_function
   BOND DAMAGE MODEL = <string>CRITICAL_STRETCH|ELEMENT_VARIABLE|NONE(NONE)
      [<string>damage_model_options]
   HOURGLASS STIFFNESS = <real>stiffness(0.0)
   BOND CUTTING BLOCK = <string list>block_names
   BOND VISUALIZATION = <string>OFF|ON(OFF)
 END [PERIDYNAMICS SECTION <string>section_name]

# Zoltan parameters
```

Explicit

```
BEGIN ZOLTAN PARAMETERS <string>parameter_name
```

```
  LOAD BALANCING METHOD  = <string>recursive coordinate
    bisection|recursive inertial bisection|hilbert space
    filling curve|octree
  DETERMINISTIC DECOMPOSITION = <string>false|true
  IMBALANCE TOLERANCE = <real>imb_tol
  OVER ALLOCATE MEMORY = <real>over_all_mem
  REUSE CUTS = <string>false|true
  ALGORITHM DEBUG LEVEL = <integer>alg_level
    #  0<=(alg_level)<=3
  CHECK GEOMETRY = <string>false|true
  KEEP CUTS = <string>false|true
  LOCK RCB DIRECTIONS = <string>false|true
  SET RCB DIRECTIONS = <string>do not order cuts|xyz|xzy|
    yzx|yxz|zxy|zyx
  RECTILINEAR RCB BLOCKS = <string>false|true
  RENUMBER PARTITIONS = <string>false|true
  OCTREE DIMENSION = <integer>oct_dimension
  OCTREE METHOD = <string>morton indexing|grey code|hilbert
  OCTREE MIN OBJECTS = <integer>min_obj  #  1<=(min_obj)
  OCTREE MAX OBJECTS = <integer>max_obj  #  1<=(max_obj)
  ZOLTAN DEBUG LEVEL = <integer>zoltan_level
    #  0<=(zoltan_level)<=10
  DEBUG PROCESSOR NUMBER = <integer>proc  #  1<=proc
  TIMER = <string>wall|cpu
  DEBUG MEMORY = <integer>dbg_mem  #  0<=(dbg_mem)<=3
END [ZOLTAN PARAMETERS <string>parameter_name]

# Output scheduler

BEGIN OUTPUT SCHEDULER <string>scheduler_name
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps          (Explicit)
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
END [OUTPUT SCHEDULER <string>scheduler_name]

# FETI equation solver

BEGIN FETI EQUATION SOLVER <string>name                  (Implicit)
  #
```

```
  # convergence commands
  MAXIMUM ITERATIONS = <integer>max_iter(500)
  RESIDUAL NORM TOLERANCE = <real>resid_tol(1.0e-6)
  #
  # diagnostic commands, off by default.
  # solver turns on print statements from FETI
  # matrix dumps the matrix to a matlab file (in serial)
  PARAM-STRING "debugMask" VALUE <string>"solver"|"matrix"
  #
  # memory usage commands
  PARAM-STRING "precision" VALUE <string>"single"|"double"
    ("double")
  PRECONDITIONING METHOD = NONE|LUMPED|DIRICHLET(DIRICHLET)
  MAXIMUM ORTHOGONALIZATION = <integer>max_orthog(500)
  #
  # solver commands
  LOCAL SOLVER = SKYLINE|SPARSE|ITERATIVE(SPARSE)
  COARSE SOLVER = SKYLINE|SPARSE|ITERATIVE(SPARSE)
  #
  # This command is only used in conjunction with
  # local solver = iterative.
  NUM LOCAL SUBDOMAINS = <integer>num_local_subdomains
END [ FETI EQUATION SOLVER <string>name]

# Begin Procedure scope
```

Explicit
Implicit

```
BEGIN ADAGIO PROCEDURE <string>procedure_name
BEGIN PRESTO PROCEDURE <string>procedure_name
```

Explicit

```
  PRINT BANNER INTERVAL = <integer>print_banner_interval(MAX_INT)

  BEGIN PROCEDURAL TRANSFER <string>name

    BLOCK = <string list>block_name
    INCLUDE ALL BLOCKS
    REMOVE BLOCK = <string list>block_name

    BEGIN INTERPOLATION TRANSFER <string>name

      BLOCK BY BLOCK
      NEAREST ELEMENT COPY

      SEND BLOCKS = <string list>block_name
      SEND COORDINATES = ORIGINAL|CURRENT

      RECEIVE BLOCKS = <string list>block_name
      RECEIVE COORDINATES = ORIGINAL|CURRENT
```

```
    TRANSFORMATION TYPE = NONE|RIGIDBODY

  END [INTERPOLATION TRANSFER <string>name]
END [PROCEDURAL TRANSFER <string>name]

# Time block

BEGIN TIME CONTROL
  BEGIN TIME STEPPING BLOCK <string>time_block_name
    START TIME = <real>start_time_value

    # Explicit dynamics time stepping commands

    BEGIN PARAMETERS FOR PRESTO REGION <string>region_name
      INITIAL TIME STEP = <real>initial_time_step_value
      TIME STEP SCALE FACTOR =
        <real>time_step_scale_factor(1.0)
      TIME STEP INCREASE FACTOR =
        <real>time_step_increase_factor(1.1)
      STEP INTERVAL = <integer>nsteps(100)
      USER TIME STEP = <real>time_step
    END [PARAMETERS FOR PRESTO REGION <string>region_name]

    # Implicit time stepping commands

    BEGIN PARAMETERS FOR ADAGIO REGION <string>region_name
      TIME INCREMENT = <real>time_increment_value
      NUMBER OF TIME STEPS = <integer>nsteps
      TIME INCREMENT FUNCTION = <string>time_function
    END [PARAMETERS FOR ADAGIO REGION <string>region_name]
  END [TIME STEPPING BLOCK <string>time_block_name]
  TERMINATION TIME = <real>termination_time
END TIME CONTROL

# Begin Region scope

BEGIN PRESTO REGION <string>region_name
BEGIN ADAGIO REGION <string>region_name

  USE FINITE ELEMENT MODEL <string>model_name
  GLOBAL ENERGY REPORTING = EXACT|APPROXIMATE|OFF (EXACT)
  EXTENSIVE RIGID BODY VARS OUTPUT = OFF|HISTORY|RESULTS|ALL (ALL)
  #
  # The intent was to use this line command to turn on more
  # or less output to the log file.  In reality, it is not
  # used extensively throughout the code and may become
```

Margin markers: "Explicit" beside BEGIN PARAMETERS FOR PRESTO REGION and BEGIN PRESTO REGION; "Implicit" beside BEGIN PARAMETERS FOR ADAGIO REGION and BEGIN ADAGIO REGION.

```
# deprecated in the future.
#
LOGFILE DETAIL = <integer>(0) #-1 less output, 1 more output.

# implicit dynamic time integration

BEGIN IMPLICIT DYNAMICS
  ACTIVE PERIODS = <string list>period_names
  USE HHT INTEGRATION
  ALPHA = <real>alpha(0.0) [DURING <string list>period_names]
  GAMMA = <real>beta(0.5) [DURING <string list>period_names]
  BETA = <real>beta(0.25) [DURING <string list>period_names]
  TIME INTEGRATION CONTROL = <string>ADAPTIVE|COMPUTERESIDUAL|
    IGNORE(IGNORE) [DURING <string list>period_names]
  INCREASE ERROR THRESHOLD = <real>increase_threshold(0.02)
    [DURING <string list>period_names]
  HOLD ERROR THRESHOLD = <real>hold_threshold(0.10)
    [DURING <string list>period_names]
  DECREASE ERROR THRESHOLD = <real>decrease_threshold(0.25)
    [DURING <string list>period_names]
END [IMPLICIT DYNAMICS]

# Time step control using Lanczos

BEGIN LANCZOS PARAMETERS <string>lanczos_name
  NUMBER EIGENVALUES = <integer>num_eig(20)
  STARTING VECTOR = <string>STRETCH_X|STRETCH_Y|STRETCH_Z
    (STRETCH_X)
  VECTOR SCALE = <real>vec_scale(1.0e-5)
  TIME SCALE = <real>time_scale(0.9)
  STEP INTERVAL = <integer>step_int(1)
  INCREMENT INTERVAL = <integer>incr_int(5)
  TIME STEP LIMIT = <real>step_lim(0.10)
END [LANCZOS PARAMETERS <string>lanczos_name]

# Time step control using nodes

BEGIN NODE BASED TIME STEP PARAMETERS <string>nbased_name
  INCREMENT INTERVAL = <integer>incr_int(5)
  STEP INTERVAL = <integer>step_int(500)
  TIME STEP LIMIT = <real>step_lim(0.10)
END [NODE BASED TIME STEP PARAMETERS <string>nbased_name]

# Mass scaling

BEGIN MASS SCALING
  #
```

Implicit (margin label at BEGIN IMPLICIT DYNAMICS); Explicit (margin labels at BEGIN LANCZOS PARAMETERS, BEGIN NODE BASED TIME STEP PARAMETERS, BEGIN MASS SCALING)

```
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  TARGET TIME STEP = <real>target_time_step
  ALLOWABLE MASS INCREASE RATIO
    = <real>mass_increase_ratio
  #
  # additional command
  ACTIVE PERIODS = <string list>periods
  INACTIVE PERIODS = <string list>periods
END MASS SCALING

# Torsional spring
```

Explicit

```
BEGIN TORSIONAL SPRING MECHANISM <string>spring_name
  NODE SETS = <string>nodelist_int1
    <string>nodelist_int2
    <string>nodelist_int3 <string>nodelist_int4
  TORSIONAL STIFFNESS = <real>stiffness
  INITIAL TORQUE = <real>init_load
  PERIOD = <real>time_period
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [TORSIONAL SPRING MECHANISM <string>spring_name]

# Mass property calculations
```

Explicit

```
BEGIN MASS PROPERTIES
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  # structure command
  STRUCTURE NAME = <string>structure_name
END [MASS PROPERTIES]

# SPH utility commands
```

Explicit

```
BEGIN SPH OPTIONS
```

```
                SPH SYMMETRY PLANE <string>+X|+Y|+Z|-X|-Y|-Z
                  <real>position_on_axis(0.0)
                SPH DECOUPLE STRAINS: <string>material1 <string>material2
              END [SPH OPTIONS]

              # Element death

              BEGIN ELEMENT DEATH <string>death_name
                #
                # block set commands
                BLOCK = <string list>block_names
                INCLUDE ALL BLOCKS
                REMOVE BLOCK = <string list>block_names
                #
                # criterion commands
Explicit        CRITERION IS AVG|MAX|MIN NODAL VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance
Explicit        CRITERION IS ELEMENT VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance [KILL WHEN
                  <integer>num_intg INTEGRATION POINTS REMAIN]
Explicit        CRITERION IS GLOBAL VALUE OF
                  <string>var_name <|<=|=|>=|> <real>tolerance
Explicit        CRITERION IS ALWAYS TRUE
                MATERIAL CRITERION
                  = <string list>material_model_names [KILL WHEN
                  <integer>num_intg INTEGRATION POINTS REMAIN]
                #
                # subroutine commands
Explicit        ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
Explicit        SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
Explicit        SUBROUTINE REAL PARAMETER: <string>param_name
                  = <real>param_value
Explicit        SUBROUTINE INTEGER PARAMETER: <string>param_name
                  = <integer>param_value
Explicit        SUBROUTINE STRING PARAMETER: <string>param_name
                  = <string>param_value
                #
                # evaluation commands
Explicit        CHECK STEP INTERVAL = <integer>num_steps
Explicit        CHECK TIME INTERVAL = <real>delta_t
Explicit        DEATH START TIME = <real>time
                #
                # miscellaneous option commands
                SUMMARY OUTPUT STEP INTERVAL = <integer>output_step_interval
                SUMMARY OUTPUT TIME INTERVAL = <real>output_time_interval
Explicit        DEATH ON INVERSION = OFF|ON(OFF)
Explicit        DEATH ON ILL DEFINED CONTACT = OFF|ON(OFF)
```

```
Explicit      DEATH STEPS = <integer>death_steps(1)
Explicit      FORCE VALID ACME CONNECTIVITY
Explicit      AGGRESSIVE CONTACT CLEANUP = <string>OFF|ON(OFF)
Explicit      DEATH METHOD = <string>DEACTIVATE ELEMENT|
                         DEACTIVATE NODAL MPCS|DISCONNECT ELEMENT|
                         INSERT COHESIVE ZONES(DEACTIVATE ELEMENT)
              #
              # death on proximity commands
Explicit      BEGIN DEATH PROXIMITY
                BLOCK = <string list>block_names
                NODE SET = <string list>nodelist_names
                SURFACE = <string list>surface_names
                REMOVE BLOCK = <string list>block_names
                REMOVE NODE SET = <string list>nodelist_names
                REMOVE SURFACE = <string list>surface_names
                TOLERANCE = <real>search_tolerance
              END [DEATH PROXIMITY]
              #
              # particle conversion commands
Explicit      BEGIN PARTICLE CONVERSION
                PARTICLE SECTION = <string>section_name
                NEW MATERIAL = <string>material_name
                  MODEL NAME = <string>model_name
                TRANSFER = <string>NONE|STRESS_ONLY|ALL
                MAX NUM PARTICLES = <integer>num
                MIN PARTICLE CONVERSION VOLUME = <real>vol
              END [PARTICLE CONVERSION]
              ACTIVE PERIODS = <string list>period_names
              INACTIVE PERIODS = <string list>period_names
              #
              # cohesive zone setup commands
              COHESIVE SECTION = <string>sect_name
              COHESIVE MATERIAL = <string>mat_name
              COHESIVE MODEL = <string>model_name
              COHESIVE ZONE INITIALIZATION METHOD = <string>NONE|
                ELEMENT STRESS AVG(NONE)
            END [ELEMENT DEATH <string>death_name]

            # Particle Embedding

Explicit    BEGIN PARTICLE EMBEDDING <string>particle_embedding_name
              BLOCK = <string>block_name
              INCLUDE ALL BLOCKS
              REMOVE BLOCK = <string>block_name
              PARTICLE SECTION = <string>section_name
              NEW MATERIAL = <string>material_name
                MODEL NAME = <string>model_name
```

```
  TRANSFER = <string>NONE|ALL(ALL)
  MAX NUM PARTICLES = <integer>num(1)
  MIN PARTICLE CONVERSION VOLUME = <real>vol
END [PARTICLE EMBEDDING]

# Derived output

BEGIN DERIVED OUTPUT
  COMPUTE AND STORE VARIABLE =
    <string>derived_quantity_name
END DERIVED OUTPUT

# Mesh rebalance
```

Explicit

```
BEGIN REBALANCE
  ELEMENT GROUPING TYPE = SPLIT SPH AND STANDARD ELEMENTS|
    UNIFORM UNIFIED(SPLIT SPH AND STANDARD ELEMENTS)
  INITIAL REBALANCE = ON|OFF(OFF)
  PERIODIC REBALANCE = ON|OFF|AUTO(OFF)
  REBALANCE STEP INTERVAL = <integer>stepInterval
  LOAD RATIO THRESHOLD = <real>loadRatio
  COMMUNICATION RATIO THRESHOLD = <real>communicationRatio
  ZOLTAN PARAMETERS = <string>parameterName
END [REBALANCE]

# Remeshing
```

Explicit

```
BEGIN REMESH
  MAX REMESH STEP INTERVAL = <integer>stepInterval (Infinity)
  MAX REMESH TIME INTERVAL = <real>timeInterval (Infinity)
  NEW MESH MAX EDGE LENGTH RATIO = <real>newMaxRatio (1.25)
  NEW MESH MIN EDGE LENGTH RATIO = <real>newMinRatio (0.25)
  NEW MESH MIN SHAPE = <real>newMinShape (0.125)
  REMESH AT MAX EDGE LENGTH RATIO = <real>maxCutoffRatio
    (newMaxEdgeLengthRatio*1.75)
  REMESH AT MIN EDGE LENGTH RATIO = <real>minCutoffRatio
    (newMinEdgeLengthRatio*0.25)
  REMESH AT SHAPE = <real>cutoffShape (0.025)
  CONTACT CLEANUP = AUTO|OFF|ON (AUTO)
  DEBUG OUTPUT LEVEL = <integer>level (0)
  MAX REMESH REBALANCE METRIC = <integer>rebalanceMetric (1.25)
  MIN SHAPE IMPROVEMENT = <real>minShapeImprovement(0.001)
  MAX NUM PATCH RECREATIONS = <integer>maxNumPatchRecreations(5)
  POISON CRITICAL VALUE = <integer>poisonCriticalValue(4)
  POISON SUBTRACTION INTERVAL =
    <integer>poisonSubtractionInterval(10)
  MAX ITERATIONS = <integer>maxIterations(5)
```

```
  BEGIN REMESH BLOCK SET
    BLOCK = <string list>blockNames
    REMOVE BLOCK = <string list>blockNames
    INCLUDE ALL BLOCKS
  END [REMESH BLOCK SET]

  BEGIN ADAPTIVE REFINEMENT
    #
    # adaptive refinement control commands
    ADAPT TYPE = NODE_PROXIMITY|POINT_PROXIMITY|
      SHARP_EDGE_PROXIMITY|FACE_PROXIMITY
    RADIUS = <real>radius
    ADAPT SHARP ANGLE = <real>angle (45)
    ADAPT SMOOTH ANGLE = <real>angle (10)
    GEOMETRIC POINT COORDINATES = <real>x <real>y <real>z
    ADAPT SIZE RATIO = <real>ratio (2.0)
    TARGET MESH SIZE = <real>target_size
    #
    # tool mesh entity commands
    NODE SET = <string list>nodelist_names
    REMOVE NODE SET = <string list>nodelist_names
    SURFACE = <string list>surface_names
    REMOVE SURFACE = <string list>surface_names
    BLOCK = <string list>block_names
    REMOVE BLOCK = <string list>block_names
    INCLUDE ALL BLOCKS
    #
    # activation commands
    ACTIVE PERIODS = <string>period_names
    INACTIVE PERIODS = <string>period_names
  END [ADAPTIVE REFINEMENT]
END [REMESH]

# Initial Mesh Modification

BEGIN INITIAL MESH MODIFICATION
  ROTATE BLOCK <string list>block_names
    ANGLE <real>angle
    ABOUT ORIGIN <real>Ox <real>Oy <real>Oz
    DIRECTION <real>Dx <real>Dy <real>Dz
  MOVE BLOCK <string list>block_names
    X <real>Mx  Y <real>My  Y <real>Mz
END [INITIAL MESH MODIFICATION]

# Initial conditions
```

```
BEGIN INITIAL CONDITION
  #
  # mesh-entity set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # variable identification commands
  INITIALIZE VARIABLE NAME = <string>var_name
  VARIABLE TYPE = [NODE|EDGE|FACE|ELEMENT|GLOBAL]
  #
  # specification command
  MAGNITUDE = <real list>initial_values
  #
  # Weibull probability distribution commands
  WEIBULL SHAPE = <real>shapeVal
  WEIBULL SCALE = <real>scaleVal
  WEIBULL MEDIAN = <real>medianVal
  WEIBULL SEED = <int>seedVal (123456)
  WEIBULL SCALING FIELD TYPE = NODE|EDGE|FACE|ELEMENT|GLOBAL (ELEMENT)
  WEIBULL SCALING FIELD NAME = <string>fieldVal (VOLUME)
  WEIBULL SCALING REFERENCE VALUE = <real>value
  WEIBULL SCALING EXPONENT SCALE = <real>scaleVal (1.0)
  #
  # input mesh commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>subroutine_name |
    ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional command
```

```
  SCALE FACTOR = <real>scale_factor(1.0)
END [INITIAL CONDITION]

# Boundary conditions

BEGIN FIXED DISPLACEMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # component commands
  COMPONENT = <string>X/Y/Z | COMPONENTS =
    <string>X/Y/Z
  #
  # additional command
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [FIXED DISPLACEMENT]

BEGIN PRESCRIBED DISPLACEMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
```

```
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED DISPLACEMENT]

BEGIN PRESCRIBED VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z |
    CYLINDRICAL AXIS = <string>defined_axis |
    RADIAL AXIS = <string>defined_axis
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
```

```
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED VELOCITY]

BEGIN PRESCRIBED ACCELERATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
```

```
END [PRESCRIBED ACCELERATION]

BEGIN PERIODIC
  SLAVE  = <string>slave_node_set
  MASTER = <string>master_node_or_face_set
  FACE CONSTRAINTS = FALSE|TRUE(FALSE)
  COMPONENT = <string>X/Y/Z
  PRESCRIBED QUANTITY = DISPLACEMENT|FORCE|VELOCITY
  SEARCH TOLERANCE = <real>tol
  FUNCTION = <string>func_name
  SCALE FACTOR = <real>func_scale
  POINT ON AXIS = <string>point_name
  REFERENCE AXIS = <string>direction_name
  THETA = <real>theta
  ACTIVE PERIODS = <string list>periods_names
  INACTIVE PERIODS = <string list>periods_names
END [PERIODIC]

BEGIN FIXED ROTATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # component commands
  COMPONENT = <string>X/Y/Z | COMPONENTS =
    <string>X/Y/Z
  #
  # additional command
  ACTIVE PERIODS = <string list>periods_names
  INACTIVE PERIODS = <string list>periods_names
END [FIXED ROTATION]

BEGIN PRESCRIBED ROTATION
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
```

```
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED ROTATION]

BEGIN PRESCRIBED ROTATIONAL VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
```

```
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # external database commands
  READ VARIABLE = <string>var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED ROTATIONAL VELOCITY]
```

Implicit

```
BEGIN REFERENCE AXIS ROTATION
  #
  # block command
  BLOCK = <string list>block_names
  #
  # specification commands
  REFERENCE AXIS X FUNCTION = <string>function_name
  REFERENCE AXIS Y FUNCTION = <string>function_name
  REFERENCE AXIS Z FUNCTION = <string>function_name
  #
  # rotation commands
  ROTATION = <string>function_name
  ROTATIONAL VELOCITY = <string>function_name
  #
  # torque command
  TORQUE = <string>function_name
  #
  # additional command
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [REFERENCE AXIS ROTATION]

BEGIN INITIAL VELOCITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
```

```
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # direction commands
  COMPONENT = <string>X|Y|Z |
    DIRECTION = <string>defined_direction
  MAGNITUDE = <real>magnitude_of_velocity
  #
  # angular velocity commands
  CYLINDRICAL AXIS = <string>defined_axis
  ANGULAR VELOCITY = <real>angular_velocity
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
END [INITIAL VELOCITY]

BEGIN PRESSURE
  #
  # surface set commands
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  #
  # function command
  FUNCTION = <string>function_name
  VELOCITY DAMPING COEFFICIENT = <real>coefficient
  #
  # user subroutine commands
  SURFACE SUBROUTINE = <string>subroutine_name |
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
```

```
  #
  # external pressure sources
  READ VARIABLE = <string>variable_name
  OBJECT TYPE = <string>NODE|FACE(NODE)
  TIME = <real>time
  FIELD VARIABLE = <string>field_variable
  #
  # output external forces from pressure
  EXTERNAL FORCE CONTRIBUTION OUTPUT NAME
    = <string>variable_name
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESSURE]

BEGIN TRACTION
  #
  # surface set commands
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  #
  # function commands
  DIRECTION = <string>direction_name
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [TRACTION]

BEGIN PRESCRIBED FORCE
  #
  # node set commands
  NODE SET = <string list>nodelist_names
```

```
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED FORCE]

BEGIN PRESCRIBED MOMENT
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # function commands
  DIRECTION = <string>defined_direction |
    COMPONENT = <string>X|Y|Z
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
```

```
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED MOMENT]

BEGIN GRAVITY
  #
  # node set commands
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  DIRECTION = <string>defined_direction
  FUNCTION = <string>function_name
  GRAVITATIONAL CONSTANT = <real>g_constant
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [GRAVITY]

BEGIN CENTRIPETAL FORCE
  NODE SET = <string list>nodelist_names
  SURFACE = <string list>surface_names
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list>surface_names
  REMOVE BLOCK = <string list>block_names
  #
  CYLINDRICAL AXIS = <string>cylindrical_axis
  ROTATIONAL VELOCITY FUNCTION = <string>rotational_vel_name
  ROTATIONAL VELOCITY SCALE FACTOR = <real>rvsf(1.0)
  FORCE SCALE FACTOR = <real>fsf(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
```

```
END [CENTRIPETAL FORCE]

BEGIN PRESCRIBED TEMPERATURE
  #
  # block set commands
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK
  #
  # function command
  FUNCTION = <string>function_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # read variable commands
  READ VARIABLE = <string>mesh_var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  TEMPERATURE TYPE = SOLID_ELEMENT|SHELL_ELEMENT(SOLID_ELEMENT)
  #
  # coupled analysis commands
  RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PRESCRIBED TEMPERATURE]

BEGIN PORE PRESSURE
  #
  # block set commands
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK
  #
  # function command
  FUNCTION = <string>function_name
```

```
  #
  # user subroutine commands
  ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # read variable commands
  READ VARIABLE = <string>mesh_var_name
  COPY VARIABLE = <string>var_name
    [FROM MODEL <string>model_name]
  TIME = <real>time
  #
  # coupled analysis commands
  RECEIVE FROM TRANSFER [FIELD TYPE = NODE|ELEMENT(NODE)]
  #
  # additional commands
  SCALE FACTOR = <real>scale_factor(1.0)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [PORE PRESSURE]

BEGIN FLUID PRESSURE
  #
  # surface set commands
  SURFACE = <string list>surface_names
  #
  # specification commands
  DENSITY = <real>fluid_density
  DENSITY FUNCTION = <string>density_function_name
  GRAVITATIONAL CONSTANT = <real>gravitational_acceleration
  FLUID SURFACE NORMAL = <string>global_component_names
  DEPTH = <real>fluid_depth
  DEPTH FUNCTION = <string>depth_function_name
  #
  # additional commands
  REFERENCE POINT = <string>reference_point_name
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [FLUID PRESSURE]

# Specialized boundary conditions
```

Explicit

```
BEGIN CAVITY EXPANSION
  EXPANSION RADIUS = <string>SPHERICAL|CYLINDRICAL
    (spherical)
  SURFACE = <string list>surface_ids
  REMOVE SURFACE = <string list>surface_ids
  FREE SURFACE = <real>top_surface_zcoord
    <real>bottom_surface_zcoord
  NODE SETS TO DEFINE BODY AXIS =
    <string>nodelist_1 <string>nodelist_id2
  TIP RADIUS = <real>tip_radius
  BEGIN LAYER <string>layer_name
    LAYER SURFACE = <real>top_layer_zcoord
      <real>bottom_layer_zcoord
    PRESSURE COEFFICIENTS = <real>c0 <real>c1
      <real>c2
    SURFACE EFFECT = <string>NONE|SIMPLE_ON_OFF(NONE)
    FREE SURFACE EFFECT COEFFICIENTS = <real>coeff1
      <real>coeff2
  END [LAYER <string>layer_name]
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [CAVITY EXPANSION]
```

Explicit

```
BEGIN SILENT BOUNDARY
  SURFACE = <string list>surface_names
  REMOVE SURFACE = <string list>surface_names
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [SILENT BOUNDARY]
```

Explicit

```
BEGIN SPOT WELD
  NODE SET = <string list>nodelist_ids
  REMOVE NODE SET = <string list>nodelist_ids
  SURFACE = <string list>surface_ids
  REMOVE SURFACE = <string list>surface_ids
  SECOND SURFACE = <string>surface_id
  NORMAL DISPLACEMENT FUNCTION =
    <string>function_nor_disp
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_nor_disp(1.0)
  TANGENTIAL DISPLACEMENT FUNCTION =
    <string>function_tang_disp
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_tang_disp(1.0)
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE FUNCTION = <string>fail_func_name
  FAILURE DECAY CYCLES = <integer>number_decay_cycles(10)
```

```
  SEARCH TOLERANCE = <real>search_tolerance
  IGNORE INITIAL OFFSET = NO|YES(NO)
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [SPOT WELD]
```

Explicit

```
BEGIN LINE WELD
  SURFACE = <string list> surface_names
  REMOVE SURFACE = <string list> surface_names
  BLOCK = <string list> block_names
  REMOVE BLOCK = <string list>block_names
  SEARCH TOLERANCE = <real>search_tolerance
  R DISPLACEMENT FUNCTION =
    <string>r_disp_function_name
  R DISPLACEMENT SCALE FACTOR = <real>r_disp_scale
  S DISPLACEMENT FUNCTION =
    <string>s_disp_function_name
  S DISPLACEMENT SCALE FACTOR = <real>s_disp_scale
  T DISPLACEMENT FUNCTION =
    <string>t_disp_function_name
  T DISPLACEMENT SCALE FACTOR = <real>t_disp_scale
  R ROTATION FUNCTION =
    <string>r_rotation_function_name
  R ROTATION SCALE FACTOR = <real>r_rotation_scale
  S ROTATION FUNCTION =
    <string>s_rotation_function_name
  S ROTATION SCALE FACTOR = <real>s_rotation_scale
  T ROTATION FUNCTION =
    <string>t_rotation_function_name
  T ROTATION SCALE FACTOR = <real>t_rotation_scale
  FAILURE ENVELOPE EXPONENT = <real>k(2.0)
  FAILURE DECAY CYCLES = <integer>number_decay_cycles
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [LINE WELD]
```

Explicit

```
BEGIN VISCOUS DAMPING <string>damp_name
  #
  # block set commands
  BLOCK = <string list>block_names
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = <string list>block_names
  #
  MASS DAMPING COEFFICIENT = <real>mass_damping
  STIFFNESS DAMPING COEFFICIENT = <real>stiff_damping
  #
  # additional command
```

```
  ACTIVE PERIODS = <string list>period names
  INACTIVE PERIODS = <string list>period_names
END [VISCOUS DAMPING <string>damp_name]
```

Explicit

```
BEGIN VOLUME REPULSION OLD <string>repulsion
  FRICTION COEFFICIENT = <real>fric_coeff
  SCALE FACTOR = <real>scale_factor
  OVERLAP TYPE = [NODAL|VOLUMETRIC]

  BEGIN BLOCK SET <string>set
    BLOCK = <string list>block_names
    INCLUDE ALL BLOCKS
    REMOVE BLOCK = <string list>block_names
    #
    SURFACE = <string list>surface_names
    REMOVE SURFACE = <string list>surface_names
    #
    ACTIVE PERIODS = <string list>period names
    INACTIVE PERIODS = <string list>period names
    #
    LINE CYLINDER RADIUS = <real>cylinder_radius
    ELEMENT REPRESENTATION = [BEAM_ELEMENT_CYLINDERS|
      TRUE_SOLID_VOLUME|NODES]
  END [BLOCK SET <string>set]
END [VOLUME REPULSION OLD <string>repulsion]

BEGIN MPC
  #
  # Master/Slave MPC commands
  MASTER NODE SET = <string list>master_nset
  MASTER NODES = <integer list>master_nodes
  MASTER SURFACE = <string list>master_surf
  MASTER BLOCK = <string list>master_block
  SLAVE NODE SET = <string list>slave_nset
  SLAVE NODES = <integer list>slave_nodes
  SLAVE SURFACE = <string list>slave_surf
  SLAVE BLOCK = <string list>slave_block
  #
  # Tied contact search command
  SEARCH TOLERANCE = <real>tolerance
  VOLUMETRIC SEARCH TOLERANCE = <real>vtolerance
  #
  # Tied MPC commands
  TIED NODES = <integer list>tied_nodes
  TIED NODE SET = <string list>tied_nset
  #
  # DOF subset selection
```

```
  COMPONENTS = <enum>X|Y|Z|RX|RY|RZ
  REMOVE GAPS AND OVERLAPS = OFF|ON(OFF)
END [MPC]

RESOLVE MULTIPLE MPCS = ERROR|FIRST WINS|LAST WINS(ERROR)

BEGIN SUBMODEL
  #
  EMBEDDED BLOCKS = <string list>embedded_block
  ENCLOSING BLOCKS = <string list>enclosing_block
END [SUBMODEL]

# Contact

BEGIN CONTACT DEFINITION <string>name
  #
  # contact surface definition commands
  SKIN ALL BLOCKS = <string>ON|OFF(OFF)
    [EXCLUDE <string list> block_names]
  CONTACT SURFACE <string>name CONTAINS <string list>surface_names
  #
  BEGIN CONTACT SURFACE <string>name
    BLOCK = <string list>block_names
    SURFACE = <string list>surface_names
    NODE SET = <string list>node_set_names
    REMOVE BLOCK = <string list>block_names
    REMOVE SURFACE = <string list>surface_names
    REMOVE NODE SET = <string list>nodelist_names
    SUBSET <string>subname WITH NORMAL <real> <real> <real>
  END [CONTACT SURFACE <string>name]
  #
  CONTACT NODE SET <string>surface_name
    CONTAINS <string>nodelist_names
  #
  BEGIN ANALYTIC PLANE <string>name
    NORMAL = <string>defined_direction
    POINT = <string>defined_point
    REFERENCE RIGID BODY = <string>rb_name
  END [ANALYTIC PLANE <string>name]
  #
  BEGIN ANALYTIC CYLINDER <string>name
    CENTER = <string>defined_point
    AXIAL DIRECTION = <string>defined_axis
    RADIUS = <real>cylinder_radius
    LENGTH = <real>cylinder_length
    CONTACT NORMAL = <string>OUTSIDE|INSIDE
  END [ANALYTIC CYLINDER <string>name]
```

Explicit (margin note beside BEGIN ANALYTIC PLANE)

Explicit (margin note beside BEGIN ANALYTIC CYLINDER)

```
#
BEGIN ANALYTIC SPHERE <string>name
  CENTER = <string>defined_point
  RADIUS = <real>sphere_radius
END [ANALYTIC SPHERE <string>name]
#
# friction model definition commands
BEGIN FRICTIONLESS MODEL <string>name
END [FRICTIONLESS MODEL <string>name]
#
BEGIN CONSTANT FRICTION MODEL <string>name
  FRICTION COEFFICIENT = <real>coeff
END [CONSTANT FRICTION MODEL <string>name]
#
BEGIN TIED MODEL <string>name
END [TIED MODEL <string>name]
#
BEGIN GLUED MODEL <string>name
END [GLUED MODEL <string>name]
#
BEGIN SPRING WELD MODEL <string>name
  NORMAL DISPLACEMENT FUNCTION = <string>func_name
  NORMAL DISPLACEMENT SCALE FACTOR =
    <real>scale_factor(1.0)
  TANGENTIAL DISPLACEMENT FUNCTION = <string>func_name
  TANGENTIAL DISPLACEMENT SCALE FACTOR =
    <real>scale_factor(1.0)
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [SPRING WELD MODEL <string>name]
#
BEGIN SURFACE WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [SURFACE WELD MODEL <string>name]
#
BEGIN AREA WELD MODEL <string>name
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
```

Explicit (margin markers beside ANALYTIC SPHERE, SPRING WELD MODEL, SURFACE WELD MODEL, AREA WELD MODEL)

```
END [AREA WELD MODEL <string>name]
#
BEGIN ADHESION MODEL <string>name
  ADHESION FUNCTION = <string>func_name
  ADHESION SCALE FACTOR = <real>scale_factor(1.0)
END [ADHESION MODEL <string>name]
#
BEGIN COHESIVE ZONE MODEL <string>name
  TRACTION DISPLACEMENT FUNCTION = <string>func_name
  TRACTION DISPLACEMENT SCALE FACTOR = <real>scale_factor(1.0)
  CRITICAL NORMAL GAP = <real>crit_norm_gap
  CRITICAL TANGENTIAL GAP = <real>crit_tangential_gap
END [COHESIVE ZONE MODEL <string>name]
#
BEGIN JUNCTION MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CUTOFF DISTANCE FOR TANGENTIAL TRACTION =
    <real>distance
END [JUNCTION MODEL <string>name]
#
BEGIN THREADED MODEL <string>name
  NORMAL TRACTION FUNCTION = <string>func_name
  NORMAL TRACTION SCALE FACTOR = <real>scale_factor(1.0)
  TANGENTIAL TRACTION FUNCTION = <string>func_name
  TANGENTIAL TRACTION SCALE FACTOR =
    <real>scale_factor(1.0)
  TANGENTIAL TRACTION GAP FUNCTION = <string>func_name
  TANGENTIAL TRACTION GAP SCALE FACTOR = <real>scale_factor(1.0)
  NORMAL CAPACITY = <real>normal_cap
  TANGENTIAL CAPACITY = <real>tangential_cap
  FAILURE ENVELOPE EXPONENT = <real>exponent(2.0)
  FAILURE DECAY CYCLES = <integer>num_cycles(1)
  FAILED MODEL = <string>failed_model_name|FRICTIONLESS
    (FRICTIONLESS)
END [THREADED MODEL <string>name]
#
BEGIN PV_DEPENDENT MODEL <string>name
  STATIC COEFFICIENT = <real>stat_coeff
  DYNAMIC COEFFICIENT = <real>dyn_coeff
  VELOCITY DECAY = <real>vel_decay
  REFERENCE PRESSURE = <real>p_ref
  OFFSET PRESSURE = <real>p_off
  PRESSURE EXPONENT = <real>p_exp
END [PV_DEPENDENT MODEL <string>name]
```

Explicit (margin labels beside ADHESION, COHESIVE ZONE, JUNCTION, THREADED, PV_DEPENDENT models)

```
#
BEGIN HYBRID MODEL <string>name
  INITIALLY CLOSE = <string>close_name
  INITIALLY FAR   = <string>far_name
END [HYBRID MODEL <string>name]
#
BEGIN TIME VARIANT MODEL <string>name
  MODEL = <string>model DURING PERIODS <string_list>time_periods
END [TIME VARIANT MODEL]
#
BEGIN USER SUBROUTINE MODEL <string>name
  INITIALIZE MODEL SUBROUTINE = <string>init_model_name
  INITIALIZE TIME STEP SUBROUTINE = <string>init_ts_name
  INITIALIZE NODE STATE DATA SUBROUTINE =
    <string>init_node_data_name
  LIMIT FORCE SUBROUTINE = <string>limit_force_name
  ACTIVE SUBROUTINE = <string>active_name
  INTERACTION TYPE SUBROUTINE = <string>interaction_name
END [USER SUBROUTINE MODEL <string>name]
#
#  interaction definition commands
BEGIN INTERACTION DEFAULTS [<string>name]
  CONTACT SURFACES = <string list>surface_names
  SELF CONTACT = <string>ON|OFF(OFF)
  GENERAL CONTACT = <string>ON|OFF(OFF)
  AUTOMATIC KINEMATIC PARTITION = <string>ON|OFF(OFF)
  INTERACTION BEHAVIOR = <string>SLIDING|
    INFINITESIMAL_SLIDING|NO_INTERACTION(SLIDING)
  FRICTION MODEL = <string>friction_model_name|
    FRICTIONLESS(FRICTIONLESS)
  CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
END [INTERACTION DEFAULTS <string>name]
#
BEGIN INTERACTION [<string>name]
  SURFACES = <string>surface1 <string>surface2
  MASTER = <string>surface
  SLAVE = <string>surface
  CONSTRAINT FORMULATION = <string>NODE_FACE|FACE_FACE
  #
  # tolerance commands
  CAPTURE TOLERANCE = <real>cap_tol
  NORMAL TOLERANCE = <real>norm_tol
  TANGENTIAL TOLERANCE = <real>tang_tol
  FRICTION MODEL = <string>model_name|TIED|FRICTIONLESS
    (FRICTIONLESS)
  FACE MULTIPLIER = <real>face_multiplier(0.1)
  OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
```

Explicit (margin note at BEGIN USER SUBROUTINE MODEL)

Implicit (margin note at CAPTURE TOLERANCE)

```
    OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
    INTERFACE MATERIAL = <string>int_matl_name
      MODEL = <string>int_model_name
    KINEMATIC PARTITION = <real>kin_part
    AUTOMATIC KINEMATIC PARTITION
    INTERACTION BEHAVIOR = <string>SLIDING|INFINITESIMAL_SLIDING|
      NO_INTERACTION(SLIDING)
    #
    # kinematic enforcement only
    FRICTION COEFFICIENT = <real>coeff                          Implicit
    FRICTION COEFFICIENT FUNCTION = <string>coeff_func          Implicit
    PUSHBACK FACTOR = <real>pushback_factor(1.0)                Implicit
    TENSION RELEASE = <real>ten_release                         Implicit
    TENSION RELEASE FUNCTION = <string>ten_release_func         Implicit
  END [INTERACTION <string>name]
  #
  ACTIVE PERIODS = <string list>period_names
  #
  BEGIN USER SEARCH BOX <string>name
    CENTER = <string>center_point
    X DISPLACEMENT FUNCTION = <string>x_disp_function_name
    Y DISPLACEMENT FUNCTION = <string>y_disp_function_name
    Z DISPLACEMENT FUNCTION = <string>z_disp_function_name
    X DISPLACEMENT SCALE FACTOR = <real>x_disp_scale_factor
    Y DISPLACEMENT SCALE FACTOR = <real>y_disp_scale_factor
    Z DISPLACEMENT SCALE FACTOR = <real>z_disp_scale_factor
  END [SEARCH OPTIONS <string>name]
  #
  # contact algorithm option commands
  ENFORCEMENT = <string>AL|KINEMATIC(AL)                        Implicit
  SEARCH = <string>ACME|DASH(ACME)
  UPDATE ALL SURFACES FOR ELEMENT DEATH = <string>ON|OFF(ON)    Explicit
  #
  BEGIN REMOVE INITIAL OVERLAP                                  Explicit
    OVERLAP NORMAL TOLERANCE = <real>over_norm_tol
    OVERLAP TANGENTIAL TOLERANCE = <real>over_tang_tol
    SHELL OVERLAP ITERATIONS = <integer>max_iter(10)
    SHELL OVERLAP TOLERANCE = <real>shell_over_tol(0.0)
  END [REMOVE INITIAL OVERLAP]
  #
  MULTIPLE INTERACTIONS = <string>ON|OFF(ON)                    Explicit
  MULTIPLE INTERACTIONS WITH ANGLE = <real>angle_in_deg(60.0)   Explicit
  BEGIN SURFACE NORMAL SMOOTHING
    ANGLE = <real>angle_in_deg
    DISTANCE = <real>distance
    RESOLUTION = <string>NODE|EDGE
  END [SURFACE NORMAL SMOOTHING]
```

```
Explicit    ERODED FACE TREATMENT = <string>NONE|ALL(ALL)
            #
            BEGIN SURFACE OPTIONS
              SURFACE = <string_list>surface_names
              REMOVE SURFACE = <string_list>removed_surface_names
              BEAM RADIUS = <real> radius
              LOFTING ALGORITHM = <string>ON|OFF(ON)
              COINCIDENT SHELL TREATMENT = <string>DISALLOW|IGNORE|
                SIMPLE(DISALLOW)
              COINCIDENT SHELL HEX TREATMENT = <string>DISALLOW|
                IGNORE|TAPERED|EMBEDDED(DISALLOW)
              CONTACT SHELL THICKNESS =
                ACTUAL_THICKNESS|LET_CONTACT_CHOOSE(ACTUAL_THICKNESS)
              ALLOWABLE SHELL THICKNESS TO ELEMENT SIZE RATIOS =
                <real>lower_bound(0.1) TO <real>upper_bound(1.0)
            END [SURFACE OPTIONS]
            #
            COMPUTE CONTACT VARIABLES = ON|OFF(OFF)
            #
            BEGIN SEARCH OPTIONS [<string>name]
              GLOBAL SEARCH INCREMENT = <integer>num_steps(1)
              GLOBAL SEARCH ONCE = <string>ON|OFF(OFF)
              SEARCH TOLERANCE = <string>AUTOMATIC|USER_DEFINED(AUTOMATIC)
              NORMAL TOLERANCE = <real>norm_tol
              TANGENTIAL TOLERANCE = <real>tang_tol
Implicit      CAPTURE TOLERANCE = <real>cap_tol
Implicit      TENSION RELEASE = <real>ten_release
Implicit      SLIP PENALTY = <real>slip_pen
              FACE MULTIPLIER = <real>face_multiplier(0.1)
            END [SEARCH OPTIONS <string>name]
            #
Explicit    BEGIN ENFORCEMENT OPTIONS [<string>name]
              MOMENTUM BALANCE ITERATIONS = <integer>num_iter(5)
              NUM GEOMETRY UPDATE ITERATIONS = <integer>num_iter(5)
            END [ENFORCEMENT OPTIONS <string>name]
            #
            BEGIN DASH OPTIONS
              INTERACTION DEFINITION SCHEME = EXPLICIT|AUTOMATIC (AUTOMATIC)
              SEARCH LENGTH SCALING = <real>scale(0.15)
              ACCURACY LEVEL = <real>accuracy(1.0)
              SUBDIVISION LEVEL = <int>sublevel(0)
            END
          END [CONTACT DEFINITION <string>name]

          # Results specification

          BEGIN RESULTS OUTPUT <string>results_name
```

```
DATABASE NAME = <string>results_file_name
DATABASE TYPE =
  <string>database_type(exodusII)
OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
  (ON|TRUE|YES)
TITLE <string>user_title
NODE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | NODAL VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name] ...
      <string>variable_name [AS
      <string>dbase_variable_name]
NODESET VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | NODESET VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>nodelist_names
      ... <string>variable_name
      [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
      <string list>nodelist_names
FACE VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | FACE VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>surface_names
      ...  <string>variable_name
      [AS <string>dbase_variable_name] INCLUDE|ON|EXCLUDE
      <string list>surface_names
ELEMENT VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
  <string>dbase_variable_name]
  | ELEMENT VARIABLES = <string>variable_name
      [AS <string>dbase_variable_name]
      INCLUDE|ON|EXCLUDE <string list>block_names
      ... <string>variable_name
      [AS dbase_variable_name] INCLUDE|ON|EXCLUDE
      <string list>block_names
GLOBAL VARIABLES = <string>variable_name
  [AS <string>dbase_variable_name] ...
  <string>variable_name [AS
```

```
    <string>dbase_variable_name]
  INCLUDE = <string>list_of_included_entities
  EXCLUDE = <string>list_of_excluded_entities
  OUTPUT MESH = EXPOSED_SURFACE|BLOCK_SURFACE
  COMPONENT SEPARATOR CHARACTER = <string>character|NONE
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  SYNCHRONIZE OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
END [RESULTS OUTPUT <string>results_name]

# User output

BEGIN FILTER <string>filter_name
  ACOEFF = <real_list>a_coeff
  BCOEFF = <real_list>b_coeff
  INTERPOLATION TIME STEP = <real>ts
END [FILTER]

BEGIN USER OUTPUT
  #
  # mesh-entity set commands
  NODE SET = <string_list>nodelist_names
  SURFACE = <string_list>surface_names
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list> surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # compute result commands
  COMPUTE GLOBAL <string>result_var_name AS
    SUM|AVERAGE|MAX|MIN|MAX ABSOLUTE VALUE
    OF NODAL|ELEMENT <string>source_var_name
  COMPUTE NODAL <string>result_var_name AS
```

Explicit

```
    MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
    OF NODAL <string>source_var_name
  COMPUTE ELEMENT <string>result_var_name AS
    MAX OVER TIME|MIN OVER TIME|ABSOLUTE VALUE MAX OVER TIME
    OF ELEMENT <string>source_var_name
  COMPUTE ELEMENT <string>result_var_name AS
    MAX|MIN|AVERAGE OF ELEMENT <string>source_var_name
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>subroutine_name |
    ELEMENT BLOCK SUBROUTINE = <string>subroutine_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # copy command
  COPY ELEMENT VARIABLE <string>ev_name TO NODAL
    VARIABLE <string>nv_name
  #
  # variable transformation command
  TRANSFORM NODAL|ELEMENT VARIABLE <string>variable_name
    TO COORDINATE SYSTEM <string>coord_sys_name
    AS <string> transformed_name
  #
  # data filtering
  FILTER <string>new_var FROM NODAL|ELEMENT <string>source_var
    USING <string>filter_name
  #
  # compute for element death
  COMPUTE AT EVERY TIME STEP
  #
  # additional command
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names

END [USER OUTPUT]

BEGIN VARIABLE INTERPOLATION [<string>var_interp_name]
  TRANSFER TYPE = INTERPOLATE FROM NEAREST FACE|
    SUM TO NEAREST ELEMENT
  SOURCE VARIABLE = ELEMENT|NODAL|
    SURFACE NORMAL NODAL <string>source_var_name
```

Explicit

```
  SOURCE ELEMENT BLOCK = <string>source_block
  SOURCE SURFACE = <string>source_surface
  TARGET VARIABLE = ELEMENT|FACE|NODAL
    <string>target_var_name
  TARGET DERIVATIVE VARIABLE = ELEMENT|FACE|NODAL
    <string>target_deriv_var_name
  TARGET SURFACE = <string>target_surface
  SEARCH TOLERANCE = <real>search_tol
  PROXIMITY SEARCH TYPE = RAY SEARCH|SPHERE SEARCH
  RAY SEARCH DIRECTION = <real>vecx <real>vecy <real>vecz
  RAY SEARCH DIRECTION = ORTHOGONAL TO LINE <real>p1x
    <real>p1y <real>p1z <real>p2x <real>p2y <real>p2z
  RAY SEARCH DIRECTION = TARGET SURFACE NORMAL
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names
END [VARIABLE INTERPOLATION <string>var_interp_name]

# Time step initialization
```

Explicit

```
BEGIN TIME STEP INITIALIZATION
  #
  # mesh-entity set commands
  NODE SET = <string_list>nodelist_names
  SURFACE = <string_list>surface_names
  BLOCK = <string_list>block_names
  INCLUDE ALL BLOCKS
  REMOVE NODE SET = <string list>nodelist_names
  REMOVE SURFACE = <string list> surface_names
  REMOVE BLOCK = <string list>block_names
  #
  # user subroutine commands
  NODE SET SUBROUTINE = <string>subroutine_name |
    SURFACE SUBROUTINE = <string>sub_name |
    ELEMENT BLOCK SUBROUTINE = <string>sub_name
  SUBROUTINE DEBUGGING OFF | SUBROUTINE DEBUGGING ON
  SUBROUTINE REAL PARAMETER: <string>param_name
    = <real>param_value
  SUBROUTINE INTEGER PARAMETER: <string>param_name
    = <integer>param_value
  SUBROUTINE STRING PARAMETER: <string>param_name
    = <string>param_value
  #
  # additional command
  ACTIVE PERIODS = <string list>period_names
  INACTIVE PERIODS = <string list>period_names

END TIME STEP INITIALIZATION
```

```
# User variable

BEGIN USER VARIABLE <string>var_name
  TYPE = <string>NODE|ELEMENT|GLOBAL
    [<string>REAL|INTEGER LENGTH = <integer>length]|
    [<string>SYM_TENSOR|FULL_TENSOR|VECTOR]
  GLOBAL OPERATOR = <string>SUM|MIN|MAX
  INITIAL VALUE = <real list>values
  INITIAL VARIATION = <real list>values LINEAR DISTRIBUTION
  USE WITH RESTART
END [USER VARIABLE <string>var_name]

# History specification

BEGIN HISTORY OUTPUT <string>history_name
  DATABASE NAME = <string>history_file_name
  DATABASE TYPE =
    <string>database_type(exodusII)
  OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
   (ON|TRUE|YES)
  TITLE <string>user_title
  GLOBAL <string>variable_name
    [AS <string>history_variable_name]
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
    [AS <string>history_variable_name]
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    NEAREST LOCATION <real>global_x,
      <real>global_y>, <real>global_z
    [AS <string>history_variable_name]
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  SYNCHRONIZE OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|
    SIGHUP|SIGINT|SIGPIPE|SIGQUIT|SIGTERM|
    SIGUSR1|SIGUSR2|SIGABRT|
```

Explicit

```
    SIGKILL|SIGILL|SIGSEGV
END [HISTORY OUTPUT <string>history_name]

# Heartbeat specification

BEGIN HEARTBEAT OUTPUT <string>heartbeat_name
  STREAM NAME = <string>heartbeat_file_name
  FORMAT = SPYHIS|DEFAULT
  GLOBAL <string>variable_name
    [AS <string>heartbeat_variable_name]
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    AT NODE|NODAL|EDGE|FACE|ELEMENT <integer>entity_id
    [AS <string>heartbeat_variable_name]
  NODE|NODAL|EDGE|FACE|ELEMENT <string>variable_name
    NEAREST LOCATION <real>global_x,
      <real>global_y>, <real>global_z
    [AS <string>heartbeat_variable_name]
  START TIME = <real>output_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  SYNCHRONIZE OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
  PRECISION = <integer>precision
  LABELS = <string>OFF|ON
  LEGEND = <string>OFF|ON
  TIMESTAMP FORMAT <string>timestamp_format
  MONITOR = <string>RESULTS|RESTART|HISTORY
END [HEARTBEAT OUTPUT <string>heartbeat_name]

# Restart specification

BEGIN RESTART DATA <string>restart_name
  DATABASE NAME = <string>restart_file_name
  INPUT DATABASE NAME = <string>restart_input_file
  OUTPUT DATABASE NAME =
    <string>restart_output_file
```

Explicit (margin marker beside TIMESTEP ADJUSTMENT INTERVAL)

```
  DATABASE TYPE =
    <string>database_type(exodusII)
  OVERWRITE = <string>OFF|ON|TRUE|FALSE|YES|NO
    (ON|TRUE|YES)
  START TIME = <real>restart_start_time
  TIMESTEP ADJUSTMENT INTERVAL = <integer>steps
  AT TIME <real>time_begin INCREMENT =
    <real>time_increment_dt
  ADDITIONAL TIMES = <real>output_time1
    <real>output_time2 ...
  AT STEP <integer>step_begin INCREMENT =
    <integer>step_increment
  ADDITIONAL STEPS = <integer>output_step1
    <integer>output_step2 ...
  TERMINATION TIME = <real>termination_time_value
  OVERLAY COUNT = <integer>overlay_count
  CYCLE COUNT = <integer>cycle_count
  SYNCHRONIZE OUTPUT
  USE OUTPUT SCHEDULER <string>scheduler_name
  OUTPUT ON SIGNAL = <string>SIGALRM|SIGFPE|SIGHUP|SIGINT|
    SIGPIPE|SIGQUIT|SIGTERM|SIGUSR1|SIGUSR2|SIGABRT|
    SIGKILL|SIGILL|SIGSEGV
  OPTIONAL
END [RESTART DATA <string>restart_name]

# Solver commands

BEGIN SOLVER
  #
  # nonlinear conjugate gradient (cg) solver commands
  BEGIN CG
    #
    # convergence commands
    # Default is not defined.
    TARGET RESIDUAL = <real>target_resid
      [DURING <string list>period_names]
    TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-4)
      [DURING <string list>period_names]
    # Default is not defined.
    ACCEPTABLE RESIDUAL = <real>accept_resid
      [DURING <string list>period_names]
    # Default is 10 times target relative residual
    ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid(1.0e-3)
      [DURING <string list>period_names]
    REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|ENERGY(EXTERNAL)
      [DURING <string list>period_names]
    # Default is not defined.
```

Explicit (margin label beside TIMESTEP ADJUSTMENT INTERVAL)

Implicit (margin label beside BEGIN SOLVER)

```
MINIMUM RESIDUAL IMPROVEMENT = <real>resid_improvement
  [DURING <string list>period_names]
MINIMUM ITERATIONS  = <integer>min_iter(0)
  [DURING <string list>period_names]
RESIDUAL ROUNDOFF TOLERANCE = <real>(1.0e-15)
#
# default = 100 for tangent preconditioner.
# default = max(NumNodes,1000) for all other preconditioners.
MAXIMUM ITERATIONS  = <integer>max_iter
  [DURING <string list>period_names]
#
# preconditioner commands
PRECONDITIONER = BLOCK|BLOCK_INITIAL|DIAGONAL|
  DIAGSCALING|ELASTIC|IDENTITY|PROBE|TANGENT(ELASTIC)
  [<real>scaling_factor]
PRECONDITIONER ITERATION UPDATE = <integer>iter_update
BALANCE PROBE = <integer>balance_probe(1.0e-6)
NODAL PROBE FACTOR = <real>probe_factor(1.0e-6)
NODAL DIAGONAL SCALE = <real>nodal_diag_scale(0.0)
NODAL DIAGONAL SHIFT = <real>nodal_diag_shift(0.0)
BEGIN FULL TANGENT PRECONDITIONER [<string>name]
  #
  # solver selection commands
  #
  # This next line is defaulted to use FETI.
  LINEAR SOLVER = <string>linear_solver_name(FETI)
  NODAL PRECONDITIONER METHOD = ELASTIC|PROBE|
    DIAGONAL|BLOCK(ELASTIC)
  #
  # tangent matrix formation commands
  PROBE FACTOR = <real>probe_factor(1.0e-6)
  BALANCE PROBE = <integer>balance_probe(1)
  CONSTRAINT ENFORCEMENT = SOLVER|PENALTY(PENALTY)
  PENALTY FACTOR = <real>penalty_factor(100.0)
  TANGENT DIAGONAL SCALE = <real>tangent_diag_scale(0.0)
  TANGENT DIAGONAL SHIFT = <real>tangent_diag_shift(0.0)
  CONDITIONING = NO_CHECK|CHECK|AUTO_REGULARIZATION(CHECK)
  #
  # reset and iteration commands
  MAXIMUM RESETS FOR MODELPROBLEM = <integer>max_mp_resets
    (100000) [DURING <string list>period_names]
  MAXIMUM RESETS FOR LOADSTEP = <integer>max_ls_resets
    (100000) [DURING <string list>period_names]
  MAXIMUM ITERATIONS FOR MODELPROBLEM =
    <integer>max_mp_iter(100000)
    [DURING <string list>period_names]
  MAXIMUM ITERATIONS FOR LOADSTEP = <integer>max_ls_iter
```

```
    (100000) [DURING <string list>period_names]
  ITERATION UPDATE = <integer>iter_update
    [DURING <string list>period_names]
  SMALL NUMBER OF ITERATIONS = <integer>small_num_iter
    [DURING <string list>period_names]
  MINIMUM SMOOTHING ITERATIONS = <integer>min_smooth_iter(0)
    [DURING <string list>period_names]
  MAXIMUM SMOOTHING ITERATIONS = <integer>max_smooth_iter(0)
    [DURING <string list>period_names]
  TARGET SMOOTHING RESIDUAL = <integer>tgt_smooth_resid(0)
    [DURING <string list>period_names]
  TARGET SMOOTHING RELATIVE RESIDUAL
    = <integer>tgt_smooth_rel_resid(0)
    [DURING <string list>period_names]
  #
  # fall-back strategy commands
  STAGNATION THRESHOLD = <real>stagnation(1.0e-12)
    [DURING <string list>period_names]
  MINIMUM CONVERGENCE RATE = <real>min_conv_rate(1.0e-4)
    [DURING <string list>period_names]
  ADAPTIVE STRATEGY = SWITCH|UPDATE(SWITCH)
    [DURING <string list>period_names]
END [FULL TANGENT PRECONDITIONER [<string>name]]
#
# line search command, default is secant
LINE SEARCH ACTUAL|TANGENT [DURING <string list>period_names]
LINE SEARCH SECANT [<real>scale_factor] # default
#
# diagnostic output commands
ITERATION PRINT = <integer>iter_print(25)
  [DURING <string list>period_names]
ITERATION PLOT = <integer>iter_plot
  [DURING <string list>period_names]
ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
#
# cg algorithm commands
ITERATION RESET = <integer>iter_reset(10000)
  [DURING <string list>period_names]
ORTHOGONALITY MEASURE FOR RESET = <real>ortho_reset(0.5)
  [DURING <string list>period_names]
RESET LIMITS <integer>iter_start <integer>iter_reset
  <real>reset_growth <real>reset_orthogonality
  [DURING <string list>period_names]
BETA METHOD = FletcherReeves|PolakRibiere|
  PolakRibierePlus(PolakRibiere)
  [DURING <string list>period_names]
# line search step length bounds
```

```
  MINIMUM STEP LENGTH = <real>min_step(-infinity)
  MAXIMUM STEP LENGTH = <real>max_step(infinity)
END [CG]
#
# control contact commands
BEGIN CONTROL CONTACT
  #
  # convergence commands
  TARGET RESIDUAL = <real>target_resid
    [DURING <string list>period_names]
  TARGET RELATIVE RESIDUAL = <real>target_rel_resid(1.0e-3)
    [DURING <string list>period_names]
  TARGET RELATIVE CONTACT RESIDUAL =
    <real>target_rel_cont_resid(1.0e-3)
    [DURING <string list>period_names]
  ACCEPTABLE RESIDUAL = <real>accept_resid
    [DURING <string list>period_names]
  ACCEPTABLE RELATIVE RESIDUAL = <real>accept_rel_resid(1.0e-2)
    [DURING <string list>period_names]
  ACCEPTABLE RELATIVE CONTACT RESIDUAL =
    <real>accept_rel_cont_resid(1.0e-1)
    [DURING <string list>period_names]
  REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL(EXTERNAL)
    [DURING <string list>period_names]
  MINIMUM ITERATIONS  = <integer>min_iter(0)
    [DURING <string list>period_names]
  MAXIMUM ITERATIONS  = <integer>max_iter(100)
    [DURING <string list>period_names]
  #
  # level selection command
  LEVEL = <integer>contact_level(1)
  #
  # diagnostic output commands, off by default
  ITERATION PLOT = <integer>iter_plot
    [DURING <string list>period_names]
  ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
  #
  # Augmented Lagrange enforcement commands used with
  # enforcement = al defined in the contact definition
  #
  # Adaptive Penalty Options
  LAGRANGE ADAPTIVE PENALTY = OFF|SEPARATE|UNIFORM(OFF)
    [DURING <string list>period_names]
  LAGRANGE ADAPTIVE PENALTY GROWTH FACTOR = <real>(2.0)
    [DURING <string list>period_names]
  LAGRANGE ADAPTIVE PENALTY REDUCTION FACTOR = <real>(0.5)
    [DURING <string list>period_names]
```

```
  LAGRANGE ADAPTIVE PENALTY THRESHOLD = <real>(0.25)
    [DURING <string list>period_names]
  LAGRANGE MAXIMUM PENALTY MULTIPLIER = <real>(100.0)
    [DURING <string list>period_names]
  #
  # Tolerance Options
  LAGRANGE TARGET GAP = <real>(unused)
    [DURING <string list>period_names]
  LAGRANGE TARGET RELATIVE GAP = <real>(unused)
    [DURING <string list>period_names]
  LAGRANGE TOLERANCE = <real>(0.0)
    [DURING <string list>period_names]
  #
  # Miscellaneous Options
  LAGRANGE FLOATING CONSTRAINT ITERATIONS = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE INITIALIZE = BOTH|MULTIPLIER|NONE|PENALTY(MULTIPLIER)
    [DURING <string list>period_names]
  LAGRANGE LIMIT UPDATE = OFF|ON(OFF)
    [DURING <string list>period_names]
  LAGRANGE MAXIMUM UPDATES = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE SEARCH UPDATE = <integer>(infinity)
    [DURING <string list>period_names]
  LAGRANGE NODAL STIFFNESS MULTIPLIER = <real>(0.0)
    [DURING <string list>period_names]
END [CONTROL CONTACT]
#
# control stiffness commands
BEGIN CONTROL STIFFNESS [<string>stiffness_name]
  #
  # convergence commands
  TARGET <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
    = <real>target [DURING <string list>period_names]
  TARGET RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
    = <real>target_rel [DURING <string list>period_names]
  ACCEPTABLE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
    = <real>accept [DURING <string list>period_names]
  ACCEPTABLE RELATIVE <string>RESIDUAL|AXIAL FORCE INCREMENT|
    PRESSURE INCREMENT|SDEV INCREMENT|STRESS INCREMENT
    = <real>accept_rel [DURING <string list>period_names]
  REFERENCE = EXTERNAL|INTERNAL|BELYTSCHKO|RESIDUAL|
    ENERGY(EXTERNAL)
    [DURING <string list>period_names]
```

```
    MINIMUM ITERATIONS  = <integer>min_iter(0)
      [DURING <string list>period_names]
    MAXIMUM ITERATIONS  = <integer>max_iter
      [DURING <string list>period_names]
    #
    # level selection command
    LEVEL = <integer>stiffness_level
    #
    # diagnostic output commands, off by default.
    ITERATION PLOT = <integer>iter_plot
      [DURING <string list>period_names]
    ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
  END [CONTROL STIFFNESS <string>stiffness_name]
  #
  # control failure commands
  BEGIN CONTROL FAILURE [<string>failure_name]
    #
    #  convergence control command
    MAXIMUM ITERATIONS = <integer>max_iter
      [DURING <string list>period_names]
    #
    # level selection command
    LEVEL = <integer>failure_level
    #
    # diagnostic output commands, off by default
    ITERATION PLOT = <integer>iter_plot
      [DURING <string list>period_names]
    ITERATION PLOT OUTPUT BLOCKS  = <string list>plot_blocks
  END [CONTROL FAILURE <string>failure_name]
  #
  # predictor commands
  BEGIN LOADSTEP PREDICTOR
    TYPE = <string>SCALE_FACTOR|SECANT|EXTERNAL|EXTERNAL_FIRST
    SCALE FACTOR = <real>factor(1.0)
      [<real>first_scale_factor]
      [DURING <string list>period_names]
    SLIP SCALE FACTOR = <real>slip_factor(1.0)
      [DURING <string list>period_names]
  END [LOADSTEP PREDICTOR]
  LEVEL 1 PREDICTOR = <string>NONE|DEFAULT(DEFAULT)
END [SOLVER]

# Adaptive time stepping

BEGIN ADAPTIVE TIME STEPPING
  METHOD = <string>SOLVER|MATERIAL(SOLVER)
    [DURING <string list>period_names]
```

Implicit

```
      TARGET ITERATIONS = <integer>target_iter(0)
        [DURING <string list>period_names]
      ITERATION WINDOW = <integer>iter_window
        [DURING <string list>period_names]
      CUTBACK FACTOR = <real>cutback_factor(0.5)
        [DURING <string list>period_names]
      GROWTH FACTOR = <real>growth_factor(1.5)
        [DURING <string list>period_names]
      MAXIMUM FAILURE CUTBACKS = <integer>max_cutbacks(5)
        [DURING <string list>period_names]
      MAXIMUM MULTIPLIER = <real>max_multiplier(infinity)
        [DURING <string list>period_names]
      MINIMUM MULTIPLIER = <real>min_multiplier(1.0e-8)
        [DURING <string list>period_names]
      RESET AT NEW PERIOD = TRUE|FALSE(TRUE)
        [DURING <string list>period_names]
      ACTIVE PERIODS = <string list>period_names
    END [ADAPTIVE TIME STEPPING]

Implicit    JAS MODE [SOLVER|CONTACT|OUTPUT]

    # J-Integral

    BEGIN J INTEGRAL <jint_name>
      #
      # integral parameter specification commands
      CRACK DIRECTION = <real>dir_x <real>dir_y <real>dir_z
      CRACK PLANE SIDE SET = <string list>side_sets
      CRACK TIP NODE SET = <string list>node_sets
      INTEGRATION RADIUS = <real>int_radius
      NUMBER OF DOMAINS = <integer>num_domains
      FUNCTION = PLATEAU|PLATEAU_RAMP|LINEAR(PLATEAU)
      SYMMETRY = OFF|ON(OFF)
      DEBUG OUTPUT = OFF|ON(OFF) WITH <integer>num_nodes
        NODES ON THE CRACK FRONT
      #
      # time period selection commands
      ACTIVE PERIODS = <string list>period_names
      INACTIVE PERIODS = <string list>period_names
    END J INTEGRAL <jint_name>

Explicit  END [PRESTO REGION <string>region_name]
Explicit  END [ADAGIO REGION <string>region_name]

  # Control modes region

  BEGIN CONTROL MODES REGION
```

```
#
# model setup
USE FINITE ELEMENT MODEL <string>model_name
CONTROL BLOCKS [WITH <string>coarse_block] =
  <string list>control_blocks
#
# solver commands
BEGIN SOLVER                                              (Implicit)
  BEGIN LOADSTEP PREDICTOR
    TYPE = <string>SCALE_FACTOR|SECANT|EXTERNAL|EXTERNAL_FIRST
    SCALE FACTOR = <real>factor(1.0)
      [<real>first_scale_factor(scale_factor)]
      [DURING <string list>period_names]
    SLIP SCALE FACTOR = <real>slip_factor(1.0)
      [DURING <string list>period_names]
  END [LOADSTEP PREDICTOR]
  BEGIN CG
    #
    #      Parameters for CG
    #
  END [CG]
END SOLVER

JAS MODE [SOLVER|CONTACT|OUTPUT]                          (Implicit)
#
# time step control
TIME STEP RATIO SCALING = <real>cm_time_scale(1.0)        (Explicit)
TIME STEP RATIO FUNCTION = <string>cm_time_func           (Explicit)
LANCZOS TIME STEP INTERVAL = <integer>lanczos_interval    (Explicit)
POWER METHOD TIME STEP INTERVAL = <integer>pm_interval    (Explicit)
#
# mass scaling
HIGH FREQUENCY MASS SCALING = <real>cm_mass_scale(1.0)    (Explicit)
#
# stiffness damping
HIGH FREQUENCY STIFFNESS DAMPING COEFFICIENT =            (Explicit)
  <real>cm_stiff_damp(0.0)
#
# kinematic boundary condition commands
BEGIN FIXED DISPLACEMENT
  #
  #  Parameters for fixed displacement
  #
END [FIXED DISPLACEMENT]
BEGIN PERIODIC                                            (Implicit)
  #
  # Parameters for periodic
```

```
      #
    END [PERIODIC]
    #
    # output commands
    BEGIN RESULTS OUTPUT <string> results_name
      #
      # Parameters for results output
      #
    END RESULTS OUTPUT <string> results_name
  END [CONTROL MODES REGION]

  # RVE Region

  BEGIN RVE REGION <string>rve_region_name
    #
    # Definition of RVEs
    ELEMENTS <integer>elem_i:<integer>elem_j
      BLOCKS <integer>blk_i:<integer>blk_j
      SURFACE|NODESET <integer>start_id INCREMENT <integer>incr
    #
    # ADAGIO REGION commands valid here with the exceptions
    # discussed in Section 10.1.  These include but are
    # not limited to:
    #
    # Boundary Conditions
    #
    # Results Output Definition
    #
    # Solver Definition
    #
  END [RVE REGION <string>rve_region_name]

Explicit  END [PRESTO PROCEDURE <string>procedure_name]
Implicit  END [ADAGIO PROCEDURE <string>procedure_name]

END [SIERRA <string>name]
```

Appendix D

Consistent Units

This appendix describes common consistent sets of units. In using Sierra/SM, it is crucial to maintain a consistent set of units when entering material properties and interpreting results. The only variables that have intrinsic units are rotations, which are in radians. All other variables depend on the consistent set of units that the user uses in inputting the material properties and dimensioning the geometry.

A consistent set of units is made by picking the base units, which when using SI unit systems are length, mass, and time. If English unit systems are used, these base units are length, force, and time. All other units are then derived from these base units. Table D.1 provides several examples of commonly used consistent sets of units. In general, the names of the unit systems in this table are taken from the names of the base units. For example, CGS stands for (centimeters, grams, seconds) and IPS stands for (inches, pounds, seconds).

One of the most common mistakes related to consistent units comes in when entering density. For example, in the IPS system, a common error is to enter the density of stainless steel as 0.289 lb/in^3, when it should be entered as 7.48e-4 $lb \cdot s^2/in$. The weight per unit volume should be divided by the gravitational constant (386.4 in/s^2 in this case) to obtain a mass per unit volume.

Table D.1: Consistent Unit Sets

Unit	Unit System				
	SI	CGS	IPS	FPS	MMTS
Mass	kg	g	$\frac{lb \cdot s^2}{in}$	$slug$	$tonne$
Length	m	cm	in	ft	mm
Time	s	s	s	s	s
Density	$\frac{kg}{m^3}$	$\frac{g}{cm^3}$	$\frac{lb \cdot s^2}{in^4}$	$\frac{slug}{ft^3}$	$\frac{tonne}{mm^3}$
Force	N	$dyne$	lb	lb	N
Pressure	Pa	$\frac{dyne}{cm^2}$	psi	psf	MPa
Moment	$N \cdot m$	$dyne \cdot cm$	$in \cdot lb$	$ft \cdot lb$	$N \cdot mm$
Temperature	K	K	$^{\circ}R$	$^{\circ}R$	K
Energy	J	erg	$lb \cdot in$	$lb \cdot ft$	mJ
Velocity	$\frac{m}{s}$	$\frac{cm}{s}$	$\frac{in}{s}$	$\frac{ft}{s}$	$\frac{mm}{s}$
Acceleration	$\frac{m}{s^2}$	$\frac{cm}{s^2}$	$\frac{in}{s^2}$	$\frac{ft}{s^2}$	$\frac{mm}{s^2}$

Appendix E

Constraint Enforcement Hierarchy

When a node has multiple constraints, they are enforced in a specific order. Table E.1 shows the order of enforcement of the various types of constraints.

Table E.1: Constraint Enforcement Order

1	Contact
2	Kinematic
3	MPC
4	Rigid Body

If any of the constraints are in conflict, the last constraint enforced will override previously enforced constraints. For example, if a kinematic boundary condition and a MPC are both active on a node and conflict with each other, the kinematic boundary condition will be enforced first, followed by the MPC. As a result, the MPC will be enforced and will override the kinematic boundary condition.

Index

1PSI PRONY
in NLVE 3D Orthotropic material model, 262
2G SCALING
in Incompressible Solid material model, 256
2PSI PRONY
in NLVE 3D Orthotropic material model, 262
3PSI PRONY
in NLVE 3D Orthotropic material model, 262
4PSI PRONY
in NLVE 3D Orthotropic material model, 262
5PSI PRONY
in NLVE 3D Orthotropic material model, 262

A
in Low Density Foam material model, 236
A0
in Soil and Crushable Foam material model, 227
A1
in Soil and Crushable Foam material model, 227
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
A11
in Elastic Laminate material model, 248
A12
in Elastic Laminate material model, 248
A16
in Elastic Laminate material model, 248
A2
in Soil and Crushable Foam material model, 227
A22
in Elastic Laminate material model, 248
A26
in Elastic Laminate material model, 248
A44
in Elastic Laminate material model, 248
A45
in Elastic Laminate material model, 248
A55
in Elastic Laminate material model, 248
A66
in Elastic Laminate material model, 248
ABSCISSA
in Definition for Function, 62
ACCEPTABLE AXIAL FORCE INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE PRESSURE INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE AXIAL FORCE INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE CONTACT RESIDUAL
in Control Contact, 153
description of, 157
ACCEPTABLE RELATIVE PRESSURE INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE RESIDUAL
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE SDEV INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE STRAIN INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RELATIVE STRESS INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE RESIDUAL
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
ACCEPTABLE SDEV INCREMENT
in Control Stiffness, 167
description of, 168
ACCEPTABLE STRESS INCREMENT
in Control Stiffness, 167
description of, 168
ACTIVE FOR PROCEDURE

in Finite Element Model – in Parameters For Block, 289
description of, 296
ACTIVE PERIODS, 83
description of, 83
in Adaptive Refinement, 370
usage in, 372
in Adaptive Time Stepping
description of, 188
in Cavity Expansion, 462
in Centripetal Force, 448
in Contact Definition, 488
use of, 562
in Element Death, 343
usage in, 357
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 408
in Fluid Pressure, 459
usage in, 461
in Gravity, 446
in Implicit Dynamics, 190
description of, 190
in J Integral, 686
in Line Weld, 471
in Mass Scaling, 114
usage in, 116
in Periodic, 405
in Pore Pressure, 454
usage in, 457
in Prescribed Acceleration, 400
usage in, 404
in Prescribed Displacement, 389
usage in, 393
in Prescribed Force, 438
usage in, 441
in Prescribed Moment, 442
usage in, 445
in Prescribed Rotation, 409
usage in, 413
in Prescribed Rotational Velocity, 414
usage in, 418
in Prescribed Temperature, 449
usage in, 453
in Prescribed Velocity, 395
usage in, 399
in Pressure, 428
usage in, 432
in REFERENCE AXIS ROTATION
usage in, 422
in Silent Boundary, 465
in Spot Weld, 466
in Time Step Initialization, 731
usage in, 733
in Torsional Spring Mechanism, 337
in Traction, 434
usage in, 437
in User Output, 592
description of, 600
in Viscous Damping, 474
usage in, 475
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
ACTIVE SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523
ADAGIO PROCEDURE, 76
ADAGIO REGION
in Adagio Procedure, 76
description of, 78
ADAPT SHARP ANGLE
in Adaptive Refinement, 370
usage in, 371
ADAPT SIZE RATIO
in Adaptive Refinement, 370
usage in, 371
ADAPT SMOOTH ANGLE
in Adaptive Refinement, 370
usage in, 371
ADAPT TYPE
in Adaptive Refinement, 370
usage in, 371
ADAPTIVE REFINEMENT, 370
in Remesh, 366
ADAPTIVE STRATEGY
in Full Tangent Preconditioner, 141
description of, 147
ADAPTIVE TIME STEPPING
in Adagio Region, 185
ADDITIONAL STEPS
in Heartbeat Output, 610
description of, 616
in History Output, 601
description of, 607
in Output Scheduler, 637
description of, 639
in Restart Data, 622
description of, 633
in Results Output, 577
description of, 590
ADDITIONAL TIMES
in Heartbeat Output, 610
description of, 616
in History Output, 601
description of, 607

in Output Scheduler, 637
description of, 638
in Restart Data, 622
description of, 633
in Results Output, 577
description of, 589
ADHESION FUNCTION
in Contact Definition – in Adhesion Model, 488
description of, 517
ADHESION MODEL
in Contact Definition, 488
description of, 517
ADHESION SCALE FACTOR
in Contact Definition – in Adhesion Model, 488
description of, 517
AGGRESSIVE CONTACT CLEANUP
in Element Death, 343
in Element Death
description of, 353
ALGORITHM DEBUG LEVEL
in Zoltan Parameters, 365
as default, 364
ALIAS
in Finite Element Model, 284
description of, 288
ALLOWABLE MASS INCREASE RATIO
in Mass Scaling, 114
description of, 115
ALPHA
in Elastic 3D Orthotropic material model, 237
in Elastic Laminate material model, 248
in Implicit Dynamics, 190
description of, 191
in NLVE 3D Orthotropic material model, 262
ALPHA1
in Fiber Shell material model, 254
ALPHA2
in Fiber Shell material model, 254
ALPHA3
in Fiber Shell material model, 254
ANALYTIC CYLINDER
in Contact Definition, 488
description of, 511
ANALYTIC PLANE
in Contact Definition, 488
description of, 510
ANALYTIC SPHERE
in Contact Definition, 488
description of, 511
ANGULAR VELOCITY
in Initial Velocity, 424
description of, 426
in Rigid Body command block, 332
AREA
in Damper Section, 319
in Truss Section, 317
AREA WELD MODEL
in Contact Definition, 488
description of, 516
AT DISCONTINUITY EVALUATE TO
in Definition for Function, 62
AT STEP
in Heartbeat Output, 610
description of, 616
in History Output, 601
description of, 607
in Output Scheduler, 637
description of, 638
in Restart Data, 622
description of, 633
in Results Output, 577
description of, 590
AT TIME
in Heartbeat Output, 610
description of, 616
in History Output, 601
description of, 606
in Output Scheduler, 637
description of, 638
in Restart Data, 622
description of, 632
in Results Output, 577
description of, 589
aupst_check_elem_var, 708
aupst_check_global_var, 717
aupst_check_node_var, 708
aupst_cyl_transform, 751
aupst_evaluate_function, 707
aupst_get_elem_centroid, 726
aupst_get_elem_nodes, 721
aupst_get_elem_topology, 721
aupst_get_elem_var, 708
aupst_get_elem_var_offset, 708
aupst_get_face_nodes, 721
aupst_get_face_topology, 721
aupst_get_global_var, 717
aupst_get_integer_param, 703
aupst_get_node_var, 708
aupst_get_point, 726
aupst_get_proc_num, 726
aupst_get_real_param, 703
aupst_get_string_param, 703
aupst_get_time, 707
aupst_local_put_global_var, 717
aupst_put_elem_var, 708
aupst_put_elem_var_offset, 708
aupst_put_global_var, 717
aupst_put_node_var, 708

aupst_rec_transform, 752
AUTOMATIC KINEMATIC PARTITION
in Contact Definition – in Interaction, 488, 530
description of, 535
in Contact Definition – in Interaction Defaults, 488, 525
description of, 527
AXIAL DIRECTION
in Contact Definition – in Analytic Cylinder, 488
description of, 511
AXIS
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
AXIS OFFSET
in Beam Section, 312
AXIS OFFSET GLOBAL
in Beam Section, 312
AXIS OFFSET VARIABLE
in Beam Section, 312
AXIS Y
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
AXIS Z
in Fiber Membrane material model, 251
in Fiber Shell material model, 254

B
in Low Density Foam material model, 236
B SHIFT CONSTANT
in NLVE 3D Orthotropic material model, 262
B1
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
B11
in Elastic Laminate material model, 248
B12
in Elastic Laminate material model, 248
B16
in Elastic Laminate material model, 248
B22
in Elastic Laminate material model, 248
B26
in Elastic Laminate material model, 248
B66
in Elastic Laminate material model, 248
BALANCE PROBE
in CG, 130
description of, 134
in Full Tangent Preconditioner, 141
description of, 143
BEAM SECTION, 312
BENDING HOURGLASS STIFFNESS
in Finite Element Model – in Parameters For Block, 289
description of, 292
BENDING HOURGLASS VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292
BETA
in Elastic-Plastic material model, 210
in Foam Plasticity material model, 233
in Implicit Dynamics, 190
description of, 191
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
BETA METHOD
in CG, 130
description of, 139
BIOT'S COEFFICIENT
about, 198
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Foam Plasticity material model, 233
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Thermoelastic material model, 204
in Wire Mesh material model, 240
BLOCK
description of, 376
in Adaptive Refinement, 370
usage in, 372
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
usage in, 498
in Element Death, 343
description of, 345
in Fixed Displacement, 387

usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Line Weld, 471
in Mass Properties, 341
usage in, 341
in Mass Scaling, 114
usage in, 115
in Pore Pressure, 454
usage in, 455
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Temperature, 449
usage in, 450
in Prescribed Velocity, 395
usage in, 396
in Procedural Transfer, 757
description of, 758
in Reference Axis, 420
in Remesh Block Set, 369
in Rotational Axis Rotation
usage in, 421
in Time Step Initialization, 731
usage in, 732
in User Output, 592
usage in, 594
in Viscous Damping, 474
usage in, 474
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
BLOCK BY BLOCK
in Procedural Transfer, 757
description of, 758
BLOCK SET
in Volume Repulsion Old, 476
BOND CUTTING BLOCK
in Peridynamics Section, 329
BOND DAMAGE MODEL
in Peridynamics Section, 329
BOND VISUALIZATION
in Peridynamics Section, 329
BULK FUNCTION
in Mooney-Rivlin material model, 259
BULK MODULUS
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Mooney-Rivlin material model, 259
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in NLVE 3D Orthotropic material model, 262
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Swanson material model, 268
in Thermoelastic material model, 204
in Viscoelastic Swanson material model, 271
in Wire Mesh material model, 240

C
in Low Density Foam material model, 236
C01
in Mooney-Rivlin material model, 259
C01 FUNCTION
in Mooney-Rivlin material model, 259
C1
in BCJ material model, 223
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
C10
in BCJ material model, 223
in Mooney-Rivlin material model, 259
C10 FUNCTION

in Mooney-Rivlin material model, 259
C11
in BCJ material model, 223
C12
in BCJ material model, 223
C13
in BCJ material model, 223
C14
in BCJ material model, 223
C15
in BCJ material model, 223
C16
in BCJ material model, 223
C17
in BCJ material model, 223
C18
in BCJ material model, 223
C19
in BCJ material model, 223
C2
in BCJ material model, 223
C20
in BCJ material model, 223
C3
in BCJ material model, 223
C4
in BCJ material model, 223
C5
in BCJ material model, 223
C6
in BCJ material model, 223
C7
in BCJ material model, 223
C8
in BCJ material model, 223
C9
in BCJ material model, 223
CAPTURE TOLERANCE
in Contact Definition – in Interaction, 488, 530
description of, 534
CAVITY EXPANSION, 462
CENTER
in Contact Definition – in Analytic Cylinder, 488
description of, 511
in Contact Definition – in Analytic Sphere, 488
description of, 511
in Contact Definition – in User Search Box, 488, 557
description of, 557
CENTRIPETAL FORCE, 448
CG, 130
in Adagio Region, 130
in Solver, 126, 175
usage in, 176
CHECK GEOMETRY
in Zoltan Parameters, 365
as default, 364
CHECK STEP INTERVAL
in Element Death, 343
description of, 351
CHECK TIME INTERVAL
in Element Death, 343
description of, 351
Classical Material Models
with Peridynamics, 690
Coarse Mesh
for Explicit Control Modes, 116
COARSE SOLVER
in FETI Equation Solver, 149
description of, 151
COHESIVE MATERIAL
in Element Death, 343
description of, 357
COHESIVE MODEL
in Element Death, 343
description of, 357
COHESIVE SECTION, 300
in Element Death, 343
description of, 357
COHESIVE ZONE INITIALIZATION METHOD
in Element Death, 343
description of, 357
COHESIVE ZONE MODEL
in Contact Definition, 488
description of, 517
COINCIDENT SHELL HEX TREATMENT
in Contact Definition – in Shell Lofting, 488
description of, 550
COINCIDENT SHELL TREATMENT
in Contact Definition – in Shell Lofting, 488
description of, 550
COMMUNICATION RATIO THRESHOLD
in Rebalance, 362
description of, 363
COMPONENT
in Fixed Displacement, 387
description of, 388
in Fixed Rotation, 407
description of, 408
in Initial Velocity, 424
description of, 425
in Periodic, 405
in Prescribed Acceleration, 400
description of, 401
in Prescribed Displacement, 389
description of, 391
in Prescribed Force, 438
description of, 439

in Prescribed Moment, 442
description of, 443
in Prescribed Rotational Velocity, 414
description of, 416
in Prescribed Velocity, 395
description of, 396
COMPONENT SEPARATOR CHARACTER
in Finite Element Model, 284
description of, 288
in Results Output, 577
description of, 587
COMPONENTS
in Fixed Displacement, 387
description of, 388
in Fixed Rotation, 407
description of, 408
in MPC, 478
COMPRESSIVE STRENGTH
in Karagozian and Case concrete material model, 230
COMPUTE AT EVERY TIME STEP, 592
in User Output
description of, 599
COMPUTE ELEMENT
in User Output, 592
description of, 595
COMPUTE GLOBAL
in User Output, 592
description of, 595
COMPUTE NODAL
in User Output, 592
description of, 595
CONDITIONING
in Full Tangent Preconditioner, 141
description of, 143
CONSTANT FRICTION MODEL
in Contact Definition, 488
description of, 512
CONSTANT SPHERE RADIUS
in Particle Section, 322
CONSTRAINT ENFORCEMENT
in Full Tangent Preconditioner, 141
description of, 143
CONSTRAINT FORMULATION
in Contact Definition – in Interaction Defaults, 525
description of, 529
Constraint Mesh
for Control Modes, 174
Contact
with Peridynamics, 692
CONTACT CLEANUP
in Remesh, 366
usage in, 367
CONTACT DEFINITION, 488
CONTACT NODE SET
in Contact Definition, 488
description of, 497
CONTACT NORMAL
in Contact Definition – in Analytic Cylinder, 488
description of, 511
CONTACT SURFACE
in Contact Definition, 488
description of, 496
CONTACT SURFACE (block)
in Contact Definition, 488
description of, 498
CONTACT VARIABLES
in Contact Definition
description of, 563
CONTROL BLOCKS
in Control Modes Region, 117, 175
description of, 176
usage in, 118
CONTROL CONTACT
in Solver, 126
description of, 153
CONTROL FAILURE
in Solver, 126
description of, 172
Control Modes, 174
CONTROL MODES REGION, 117, 175
usage of for Control Modes, 174
usage of for Explicit Control Modes, 116
CONTROL STIFFNESS
in Solver, 126
description of, 167
COORDINATE SYSTEM, 73
in Solid Section, 297
in Elastic 3D Orthotropic material model, 237
in Elastic Laminate material model, 248
in NLVE 3D Orthotropic material model, 262
COPY ELEMENT VARIABLE
in User Output, 592
description of, 597
COPY VARIABLE
in Initial Condition, 380
description of, 384
in Pore Pressure, 454
description of, 456
in Prescribed Acceleration, 400
description of, 403
in Prescribed Displacement, 389
description of, 392
in Prescribed Rotation, 409
description of, 412
in Prescribed Rotational Velocity, 414
description of, 417
in Prescribed Temperature, 449

description of, 451
in Prescribed Velocity, 395
description of, 398
copy_data, 753
CORD DENSITY
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
CORD DIAMETER
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
CRACK DIRECTION
in J Integral, 686
CRACK PLANE SIDE SET
in J Integral, 686
CRACK TIP NODE SET
in J Integral, 686
CREEP CONSTANT
in Power Law Creep material model, 225
CREEP EXPONENT
in Power Law Creep material model, 225
CRITERION IS
in Element Death – for Always True, 343
description of, 349
in Element Death – for Element Value Of, 343
description of, 347
in Element Death – for Global Value Of, 343
description of, 348
in Element Death – for Nodal Value Of, 343
description of, 346
CRITICAL BIAXIALITY RATIO
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
CRITICAL CRACK OPENING STRAIN
in Ductile Fracture material model, 214
in Elastic Fracture material model, 208
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
CRITICAL NORMAL GAP
in Contact Definition – in Cohesive Zone Model, 488
description of, 517
CRITICAL TANGENTIAL GAP
in Contact Definition – in Cohesive Zone Model, 488
description of, 517
CRITICAL TEARING PARAMETER
in Ductile Fracture material model, 214
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
CRUSH XX
in Orthotropic Crush material model, 242
CRUSH XY
in Orthotropic Crush material model, 242
CRUSH YY
in Orthotropic Crush material model, 242
CRUSH YZ
in Orthotropic Crush material model, 242
CRUSH ZX
in Orthotropic Crush material model, 242
CRUSH ZZ
in Orthotropic Crush material model, 242
CUT OFF STRAIN
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
CUTBACK FACTOR
in Adaptive Time Stepping
description of, 187
CYCLE COUNT
in Restart Data, 622
description of, 634
CYLINDRICAL AXIS
in Centripetal Force, 448
in Initial Velocity, 424
description of, 426
in Prescribed Displacement, 389
description of, 391
in Prescribed Velocity, 395
description of, 396
in Rigid Body command block, 332

D11
in Elastic Laminate material model, 248
D12
in Elastic Laminate material model, 248
D16
in Elastic Laminate material model, 248
D22
in Elastic Laminate material model, 248
D26
in Elastic Laminate material model, 248
D66
in Elastic Laminate material model, 248
DAMAGE EXPONENT
in BCJ material model, 223
DAMPER SECTION, 319
DAMPING MATRIX
in Superelement Section, 326
description of, 327
DATABASE NAME
in Finite Element Model, 284
description of, 287
in History Output, 601
in Restart Data, 622
description of, 623
in Results Output, 577
DATABASE TYPE
in Finite Element Model, 284
description of, 287

in History Output, 601
in Restart Data, 622
description of, 623
in Results Output, 577
DEATH METHOD
in Element Death, 343
in Element Death
description of, 353
DEATH ON ILL DEFINED CONTACT
in Element Death
description of, 352
DEATH ON INVERSION
in Element Death, 343
in Element Death
description of, 352
DEATH PROXIMITY
in Element Death, 343
description of, 354
DEATH START TIME
in Element Death, 343
description of, 351
DEATH STEPS
in Element Death, 343
description of, 352
DEBUG MEMORY
in Zoltan Parameters, 365
DEBUG OUTPUT
in J Integral, 686
DEBUG OUTPUT LEVEL
in Remesh, 366
usage in, 367
DEBUG PROCESSOR NUMBER
in Zoltan Parameters, 365
DECREASE ERROR THRESHOLD
in Implicit Dynamics, 190
description of, 191
DEFINE AXIS
with point and direction, 67
with two points, 67
DEFINE DIRECTION, 67
DEFINE POINT, 43, 67
DEFINITION FOR FUNCTION, 62
usage for thermal strains, 201
DENSITY
in Fluid Pressure, 459
about, 198
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic Laminate material model, 248
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fluid Pressure
usage in, 460
in Foam Plasticity material model, 233
in Johnson-Cook material model, 221
in Karagozian and Case concrete model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Thermoelastic material model, 204
in Wire Mesh material model, 240
Density
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Incompressible Solid material model, 256
in Mooney-Rivlin material model, 259
in NLVE 3D Orthotropic material model, 262
in Stiff Elastic material model, 266
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
DENSITY FORMULATION
in Particle Section, 322
DENSITY FUNCTION
in Fluid Pressure, 459
usage in, 460
DEPTH
in Fluid Pressure, 459
in Fluid Pressure
usage in, 460
DEPTH FUNCTION
in Fluid Pressure, 459
usage in, 460
DERIVED LOG STRAIN OUTPUT, 361
DERIVED OUTPUT, 361
DERIVED STRAIN OUTPUT, 361
DETERMINISTIC DECOMPOSITION
in Zoltan Parameters, 365
DEVIATORIC PARAMETER
in Membrane Section, 308
in Solid Section, 297
DIRECTION
in Gravity, 446
in Initial Velocity, 424
description of, 425
in Prescribed Acceleration, 400
description of, 401

in Prescribed Displacement, 389
description of, 391
in Prescribed Force, 438
description of, 439
in Prescribed Moment, 442
description of, 443
in Prescribed Rotation, 409
description of, 411
in Prescribed Rotational Velocity, 414
description of, 416
in Prescribed Velocity, 395
description of, 396
in Rigid Body command block, 332
in Traction, 434
description of, 435
DIRECTION FOR ROTATION
in Elastic 3D Orthotropic material model, 237
in Elastic Laminate material model, 248
in NLVE 3D Orthotropic material model, 262
DOUBLE INTEG FACTOR
in NLVE 3D Orthotropic material model, 262
DRILLING STIFFNESS FACTOR
in Shell Section, 302
DYNAMIC COEFFICIENT
in Contact Definition – in PV_Dependent Model, 488
description of, 520

EDGE
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
in Results Output, 577
EFFECTIVE MODULI MODEL
in Finite Element Model – in Parameters For Block, 289
description of, 294
EIGENVALUE CONVERGENCE TOLERANCE
in Lanczos Parameters, 102
in Power Method Parameters, 109
ELEMENT
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
ELEMENT BLOCK SUBROUTINE
as user subroutine command line, 728
description of, 729
in Element Death, 343
description of, 350
in Initial Condition, 380
description of, 385
in Pore Pressure, 454
in Time Step Initialization, 731
description of, 732
in User Output, 592
description of, 596
ELEMENT DEATH, 343
example of, 358
ELEMENT GROUPING TYPE
in Rebalance, 362
ELEMENT NUMERICAL FORMULATION
in Finite Element Model – in Parameters For Block, 289
description of, 295
ELEMENT REPRESENTATION
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
ELEMENT VARIABLES
in Results Output, 577
description of, 584
ELEMENTS
in RVE REGION, 682
EMBEDDED BLOCKS
in Submodel
usage in, 483
ENCLOSING BLOCKS
in Submodel
usage in, 483
ENFORCEMENT
in Contact Definition
use of, 542
ENFORCEMENT OPTIONS
in Contact Definition, 488
description of Options, 559
EPL
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
ERODED FACE TREATMENT
in Contact Definition, 488
description of, 549
EVALUATE EXPRESSION

in Definition For Function, 62
examples of, 65
rules and options for composing, 65
EX
in Orthotropic Crush material model, 242
EXCLUDE
in Results Output, 577, 587
EXPANSION RADIUS
in Cavity Expansion, 462
Explicit Control Modes, 116
EXTERNAL FORCE CONTRIBUTION OUTPUT NAME, 428
in Pressure
description of, 432
EY
in Orthotropic Crush material model, 242
EZ
in Orthotropic Crush material model, 242

FACE
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
FACE CONSTRAINTS
in Periodic, 405
FACE MULTIPLIER
in Contact Definition – in Interaction
description of, 534
in Contact Definition – in Search Options, 488, 554
description of, 555
FACE MULTIPLIER
in Contact Definition – in Interaction, 530
FACE VARIABLES
in Results Output, 577
description of, 582
FACTOR
in BCJ material model, 223
FAILED MODEL
in Contact Definition – in Area Weld Model, 488
description of, 516
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Contact Definition – in Surface Weld Model, 488
description of, 515
in Contact Definition – in Threaded Model, 488
description of, 519
FAILURE DECAY CYCLES
in Contact Definition – in Area Weld Model, 488
description of, 516
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Contact Definition – in Surface Weld Model, 488, 515
in Contact Definition – in Threaded Model, 488
description of, 519
in Line Weld, 471
in Spot Weld, 466
FAILURE ENVELOPE EXPONENT
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Contact Definition – in Threaded Model, 488
description of, 519
in Line Weld, 471
in Spot Weld, 466
FAILURE FUNCTION
in Spot Weld, 466
FETI EQUATION SOLVER, 149
FICTITIOUS LOGA FUNCTION
in NLVE 3D Orthotropic material model, 262
FICTITIOUS LOGA SCALE FACTOR
in NLVE 3D Orthotropic material model, 262
FIELD VARIABLE
in Pressure, 428
description of, 432
FILE
in Superelement Section, 326
description of, 327
FINAL RADIUS MULTIPLICATION FACTOR
in Particle Section, 322
FINITE ELEMENT MODEL, 284
usage of for Control Modes, 174
usage of for Explicit Control Modes, 116
FIXED DISPLACEMENT, 387
in Control Modes Region, 117, 175
usage in, 121, 177
FIXED ROTATION, 407
FLANGE THICKNESS
in Beam Section, 312
FLUID PRESSURE, 459
FLUID SURFACE NORMAL
in Fluid Pressure, 459
in Fluid Pressure
usage in, 460
FORCE SCALE FACTOR
in Centripetal Force, 448
FORCE STRAIN FUNCTION
in Spring Section, 318
FORCE VALID ACME CONNECTIVITY
in Element Death, 343
description of, 353

FORMAT
in Heartbeat Output, 610
FORMULATION
in Membrane Section, 308
in Particle Section, 322
in Solid Section, 297
FRACTIONAL DILATANCY
in Karagozian and Case concrete material model, 230
Fracture
with Peridynamics, 691
FREE SURFACE
in Cavity Expansion, 462
FREE SURFACE EFFECT COEFFICIENTS
in Cavity Expansion – in Layer, 462
description of, 464
FRICTION COEFFICIENT
in Contact Definition – in Constant Friction Model, 488
description of, 512
in Contact Definition – in Interaction
description of, 537
in Volume Repulsion Old, 476
FRICTION COEFFICIENT FUNCTION
in Contact Definition – in Interaction
description of, 537
FRICTION MODEL
in Contact Definition – in Interaction, 488, 530
description of, 535
in Contact Definition – in Interaction Defaults, 488, 525
description of, 527
FRICTIONLESS MODEL
in Contact Definition, 488
description of, 512
FULL TANGENT PRECONDITIONER
in CG, 130
description of, 134, 141
FUNCTION
in Gravity, 446
in J Integral, 686
in Periodic, 405
in Pore Pressure, 454
description of, 455
in Prescribed Acceleration, 400
description of, 401
in Prescribed Displacement, 389
description of, 391
in Prescribed Force, 438
description of, 439
in Prescribed Moment, 442
description of, 443
in Prescribed Rotation, 409
description of, 411
in Prescribed Rotational Velocity, 414
description of, 416
in Prescribed Temperature, 449
description of, 450
in Prescribed Velocity, 395
description of, 396
in Pressure, 428
description of, 430
in Traction, 434
description of, 435

GAMMA
in Implicit Dynamics, 190
description of, 191
GENERAL CONTACT
in Contact Definition – in Interaction Defaults, 488, 525
GEOMETRIC POINT COORDINATES
in Adaptive Refinement, 370
usage in, 371
GLASS TRANSITION TEM
in NLVE 3D Orthotropic material model, 262
GLOBAL
in Heartbeat Output – for global variables
description of, 612
in History Output – for global variables
description of, 603
GLOBAL OPERATOR
in User Variable, 734
GLOBAL OUTPUT OPTIONS
description of, 644
GLOBAL SEARCH INCREMENT
in Contact Definition – in Search Options, 488, 554
GLOBAL SEARCH ONCE
in Contact Definition – in Search Options, 488, 554
GLOBAL VARIABLES
in Results Output, 577
description of, 588
GLUED MODEL
in Contact Definition, 488
description of, 514
GRAVITATIONAL CONSTANT
in Fluid Pressure, 459
in Fluid Pressure
usage in, 460
in Gravity, 446
GRAVITY, 446
GROWTH FACTOR
in Adaptive Time Stepping
description of, 187
GXY
in Orthotropic Crush material model, 242
GYZ
in Orthotropic Crush material model, 242

GZX
in Orthotropic Crush material model, 242

HARDEN-SOFTEN FUNCTION
in Karagozian and Case concrete material model, 230
HARDENING CONSTANT
in Ductile Fracture material model, 214
in Elastic-Plastic Power-Law Hardening material model, 212
HARDENING EXPONENT
in Ductile Fracture material model, 214
in Elastic-Plastic Power-Law Hardening material model, 212
HARDENING FUNCTION
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
HARDENING MODULUS
in Elastic-Plastic material model, 210
HEARTBEAT OUTPUT, 610
HEIGHT
in Beam Section, 312
HIGH FREQUENCY MASS SCALING
in Control Modes Region, 117
usage in, 120
HIGH FREQUENCY STIFFNESS DAMPING
in Control Modes Region
usage in, 121
HISTORY OUTPUT, 601
HOLD ERROR THRESHOLD
in Implicit Dynamics, 190
description of, 191
HORIZON
in Peridynamics, 693
in Peridynamics Section, 329
HOURGLASS FORMULATION
in Solid Section, 297
HOURGLASS INCREMENT
in Solid Section, 297
HOURGLASS ROTATION
in Solid Section, 297
HOURGLASS STIFFNESS
in Finite Element Model – in Parameters For Block, 289
description of, 292
in Peridynamics Section, 329
with Peridynamics, 694
HOURGLASS VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292
HYDRO EXPONENT
in Foam Plasticity material model, 233
HYDRO HARDENING
in Foam Plasticity material model, 233
HYDRO STRENGTH
in Foam Plasticity material model, 233

IGNORE INITIAL OFFSET
in Spot Weld, 466
IMBALANCE TOLERANCE
in Zoltan Parameters, 365
IMPLICIT DYNAMICS, 190
INACTIVE FOR PROCEDURE
in Finite Element Model – in Parameters For Block, 289
INACTIVE PERIODS, 83
description of, 83
in Adaptive Refinement, 370
usage in, 372
in Cavity Expansion, 462
in Centripetal Force, 448
in Element Death, 343
usage in, 357
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 408
in Fluid Pressure, 459
usage in, 461
in Gravity, 446
in J Integral, 686
in Line Weld, 471
in Mass Scaling, 114
usage in, 116
in Periodic, 405
in Pore Pressure, 454
usage in, 457
in Prescribed Acceleration, 400
usage in, 404
in Prescribed Displacement, 389
usage in, 393
in Prescribed Force, 438
usage in, 441
in Prescribed Moment, 442
usage in, 445
in Prescribed Rotation, 409
usage in, 413
in Prescribed Rotational Velocity, 414
usage in, 418
in Prescribed Temperature, 449
usage in, 453
in Prescribed Velocity, 395
usage in, 399
in Pressure, 428
in REFERENCE AXIS ROTATION

usage in, 422
in Silent Boundary, 465
in Spot Weld, 466
in Torsional Spring Mechanism, 337
in Traction, 434
usage in, 437
in Viscous Damping, 474
usage in, 475
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
INCLUDE
in Results Output, 577, 587
INCLUDE ALL BLOCKS
description of, 376
in Adaptive Refinement, 370
usage in, 372
in Centripetal Force, 448
in Element Death, 343
description of, 345
in Finite Element Model, 284
description of, 291
in Finite Element Model – in Parameters For Block, 289
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Mass Properties, 341
usage in, 341
in Mass Scaling, 114
usage in, 115
in Pore Pressure, 454
usage in, 455
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Temperature, 449
usage in, 450
in Prescribed Velocity, 395
usage in, 396
in Procedural Transfer, 757
description of, 758
in Remesh Block Set, 369
in Time Step Initialization, 731
usage in, 732
in User Output, 592
usage in, 594
in Viscous Damping, 474
usage in, 474
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
INCLUDE NODES IN
in Rigid Body command block, 332
INCLUDEFILE, 50
INCREASE ERROR THRESHOLD
in Implicit Dynamics
description of, 191
INCREASE ERROR THRESHOLD
in Implicit Dynamics, 190
INCREASE OVER STEPS
in Lanczos Parameters, 102
in Power Method Parameters, 109
INCREMENT INTERVAL
in Node Based Time Step Parameters, 112
INERTIA
in Rigid Body command block, 332
INFLUENCE FUNCTION
in Peridynamics Section, 329
INITIAL ALPHA_XX
in BCJ material model, 223
INITIAL ALPHA_XY
in BCJ material model, 223
INITIAL ALPHA_XZ
in BCJ material model, 223
INITIAL ALPHA_YY
in BCJ material model, 223
INITIAL ALPHA_YZ
in BCJ material model, 223
INITIAL ALPHA_ZZ
in BCJ material model, 223
INITIAL CONDITION, 380
INITIAL DAMAGE
in BCJ material model, 223
INITIAL KAPPA
in BCJ material model, 223
INITIAL LOAD
in Truss Section, 317
INITIAL MESH MODIFICATION, 379
INITIAL REBALANCE
in Rebalance, 362
description of, 363

INITIAL TIME STEP
in Parameters For Presto Region, 90
description of, 91
INITIAL TORQUE
in Torsional Spring Mechanism, 337
INITIAL VALUE
in User Variable, 734
INITIAL VELOCITY, 424
INITIALIZE MODEL SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523
INITIALIZE NODE STATE DATA SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523
INITIALIZE TIME STEP SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523
INITIALIZE VARIABLE NAME
in Initial Condition, 380
description of, 382
INPUT DATABASE NAME
in Restart Data, 622
description of, 623
INTEGRATION RADIUS
in J Integral, 686
INTEGRATION RULE
in Shell Section, 302
INTERACTION
in Contact Definition, 488
description of, 530
INTERACTION BEHAVIOR
in Contact Definition – in Interaction, 488, 530
description of, 536
in Contact Definition – in Interaction Defaults, 488, 525
description of, 528
INTERACTION DEFAULTS
in Contact Definition, 488
description of, 525
INTERACTION TYPE SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523
INTERFACE MATERIAL
in Contact Definition – in Interaction
description of, 535
INTERPOLATION TRANSFER
in Procedural Transfer, 757
description of, 758
ITERATION PLOT
in CG, 130
description of, 138
in Control Contact, 153
description of, 159
in Control Failure, 172
description of, 173
in Control Stiffness, 167
description of, 171
ITERATION PLOT OUTPUT BLOCKS
in CG, 130
description of, 138
in Control Contact, 153
description of, 159
in Control Failure, 172
description of, 173
in Control Stiffness, 167
description of, 171
ITERATION PRINT
in CG, 130
description of, 138
ITERATION RESET
in CG, 130
description of, 139
ITERATION UPDATE
in Full Tangent Preconditioner, 141
description of, 145
ITERATION WINDOW
in Adaptive Time Stepping
description of, 186

J INTEGRAL, 686
JAS MODE
in Adagio Region, 181
in Control Modes Region, 175
description of, 176
JUNCTION MODEL
in Contact Definition, 488
description of, 518

K SCALING
in Incompressible Solid material model, 256
KEEP CUTS
in Zoltan Parameters, 365
KINEMATIC PARTITION
in Contact Definition – in Interaction, 488, 530
description of, 533

L FUNCTION
in Orthotropic Rate material model, 245
LABELS
in Heartbeat Output, 610
LAGRANGE ADAPTIVE PENALTY
in Control Contact, 153
description of, 159
LAGRANGE ADAPTIVE PENALTY GROWTH FACTOR

in Control Contact, 153
description of, 159
LAGRANGE ADAPTIVE PENALTY REDUCTION FACTOR
in Control Contact, 153
description of, 159
LAGRANGE ADAPTIVE PENALTY THRESHOLD
in Control Contact, 153
description of, 159
LAGRANGE FLOATING CONSTRAINT ITERATIONS
in Control Contact, 153
description of, 159
LAGRANGE INITIALIZE
in Control Contact, 153
description of, 159
LAGRANGE LIMIT UPDATE
in Control Contact, 153
description of, 159
LAGRANGE MAXIMUM PENALTY MULTIPLIER
in Control Contact, 153
description of, 159
LAGRANGE MAXIMUM UPDATES
in Control Contact, 153
description of, 159
LAGRANGE NODAL STIFFNESS MULTIPLIER
in Control Contact, 153
description of, 159
LAGRANGE SEARCH UPDATE
in Control Contact, 153
description of, 159
LAGRANGE TARGET GAP
in Control Contact, 153
description of, 159
LAGRANGE TARGET RELATIVE GAP
in Control Contact, 153
description of, 159
LAGRANGE TOLERANCE
in Control Contact, 153
description of, 159
LAMBDA
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Mooney-Rivlin material model, 259
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in NLVE 3D Orthotropic material model, 262
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Swanson material model, 268
in Thermoelastic material model, 204
in Viscoelastic Swanson material model, 271
in Wire Mesh material model, 240
LAMBDAM
in Karagozian and Case concrete material model, 230
LAMBDAZ
in Karagozian and Case concrete material model, 230
LANCZOS PARAMETERS, 102
LANCZOS TIME STEP INTERVAL
in Control Modes Region
usage in, 119
LAYER
in Cavity Expansion, 462
description of, 463
LAYER SURFACE
in Cavity Expansion – in Layer, 462
description of, 463
LEGEND
in Heartbeat Output, 610
LENGTH
in Contact Definition – in Analytic Cylinder, 488
description of, 511
LEVEL
in Control Contact, 153
description of, 158
in Control Failure, 172
description of, 173
in Control Stiffness, 167
description of, 170
LEVEL 1 PREDICTOR
in Solver, 126
description of, 180
LIMIT FORCE SUBROUTINE
in Contact Definition – in User Subroutine Model, 488
description of, 523

LINE CYLINDER RADIUS
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
LINE SEARCH
in CG, 130
description of, 137
LINE WELD, 471
LINEAR BULK VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292
Linear Peridynamic Solid Material Model
in Peridynamics, 690
LINEAR SOLVER
in Full Tangent Preconditioner, 141
description of, 142
LINEAR VISCO TEST
in NLVE 3D Orthotropic material model, 262
LOAD BALANCING METHOD
in Zoltan Parameters, 365
as default, 364
LOAD RATIO THRESHOLD
in Rebalance, 362
LOADSTEP PREDICTOR
in Solver, 175, 178
usage in, 176
LOCAL SOLVER
in FETI Equation Solver, 149
description of, 151
LOCALIZATION SECTION, 301
LOCK RCB DIRECTIONS
in Zoltan Parameters, 365
LOFTING ALGORITHM
in Contact Definition – in Shell Lofting, 488
description of, 550
LOFTING FACTOR
in Membrane Section, 308
in Shell Section, 302
LUDERS STRAIN
in Ductile Fracture material model, 214
in Elastic-Plastic Power-Law Hardening material model, 212
LW FUNCTION
in Orthotropic Rate material model, 245
LX
in Orthotropic Rate material model, 245
LY
in Orthotropic Rate material model, 245
LZ
in Orthotropic Rate material model, 245

MAGNITUDE
in Initial Condition, 380
description of, 382
in Initial Velocity, 424
description of, 425
in Rigid Body command block, 332
MAP
in Superelement Section, 326
description of, 327
MASS
in Rigid Body command block, 332
MASS DAMPING COEFFICIENT
in Viscous Damping, 474
description of, 475
MASS MATRIX
in Superelement Section, 326
description of, 327
MASS PROPERTIES, 341
MASS SCALING, 114
MASTER
in Contact Definition – in Interaction, 488, 530
description of, 531
in Periodic, 405
MASTER BLOCK
in MPC, 478
usage in, 478
MASTER NODE SET
in MPC, 478
usage in, 478
MASTER NODES
in MPC, 478
usage in, 478
MASTER SURFACE
in MPC, 478
usage in, 478
MATERIAL
in Finite Element Model – in Parameters For Block, 289
description of, 291
MATERIAL CRITERION
in Element Death, 343
description of, 349
MATERIAL MODEL FORMULATION
in Peridynamics Section, 329
MATRIX DENSITY
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
MAX NUMBER ELEMENTS
in Remesh, 366
MAX NUMBER ELEMENTS
in Remesh
usage in, 367
MAX POISSONS RATIO
in Incompressible Solid material model, 256
in Mooney-Rivlin material model, 259
in Swanson material model, 268

in Viscoelastic Swanson material model, 271
MAX REMESH REBALANCE METRIC
in Remesh, 366
usage in, 367
MAX REMESH STEP INTERVAL
in Remesh, 366
usage in, 367
MAX REMESH TIME INTERVAL
in Remesh, 366
usage in, 367
MAX STRESS
in Elastic Fracture material model, 208
MAXIMUM AGGREGATE SIZE
in Karagozian and Case concrete material model, 230
MAXIMUM FAILURE CUTBACKS
in Adaptive Time Stepping
description of, 187
MAXIMUM ITERATIONS
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Failure, 172
description of, 172
in Control Stiffness, 167
description of, 168
in FETI Equation Solver, 149
description of, 150
MAXIMUM ITERATIONS FOR LOADSTEP
in Full Tangent Preconditioner, 141
description of, 145
MAXIMUM ITERATIONS FOR MODELPROBLEM
in Full Tangent Preconditioner, 141
description of, 145
MAXIMUM MULTIPLIER
in Adaptive Time Stepping
description of, 187
MAXIMUM ORTHOGONALIZATION
in FETI Equation Solver, 149
description of, 150
MAXIMUM RESETS FOR LOADSTEP
in Full Tangent Preconditioner, 141
description of, 145
MAXIMUM RESETS FOR MODELPROBLEM
in Full Tangent Preconditioner, 141
description of, 145
MAXIMUM SMOOTHING ITERATIONS
in Full Tangent Preconditioner, 141
description of, 145
MAXIMUM STEP LENGTH
in CG, 130
in CG
description of, 139
MEMBRANE HOURGLASS STIFFNESS
in Finite Element Model – in Parameters For Block, 289
description of, 292
MEMBRANE HOURGLASS VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292
MEMBRANE SECTION, 308
METHOD
in Adaptive Time Stepping
description of, 186
MINIMUM CONVERGENCE RATE
in Full Tangent Preconditioner, 141
description of, 147
MINIMUM ITERATIONS
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
MINIMUM MULTIPLIER
in Adaptive Time Stepping
description of, 188
MINIMUM RESIDUAL IMPROVEMENT
in CG, 130
description of, 132
MINIMUM SMOOTHING ITERATIONS
in Full Tangent Preconditioner, 141
description of, 145
MINIMUM STEP LENGTH
in CG, 130
in CG
description of, 139
MODEL
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
MODULUS FUNCTION
in Orthotropic Rate material model, 245
MODULUS LLLL
in Orthotropic Rate material model, 245
MODULUS LLWW
in Orthotropic Rate material model, 245
MODULUS LWLW
in Orthotropic Rate material model, 245
MODULUS TLTL
in Orthotropic Rate material model, 245
MODULUS TTLL
in Orthotropic Rate material model, 245
MODULUS TTTT
in Orthotropic Rate material model, 245
MODULUS TTWW
in Orthotropic Rate material model, 245

MODULUS WTWT
in Orthotropic Rate material model, 245
MODULUS WWWW
in Orthotropic Rate material model, 245
MOMENTUM BALANCE ITERATIONS
in Contact Definition – in Enforcement Options, 488, 559
description of, 559
MONAGHAN EPSILON
in Particle Section, 322
MONAGHAN N
in Particle Section, 322
MONITOR
in Heartbeat Output, 610
description of, 621
MOVE BLOCK
in Initial Mesh Modification, 379
MPC, 478
MTH11 FUNCTION
in Elastic Laminate material model, 248
MTH12 FUNCTION
in Elastic Laminate material model, 248
MTH22 FUNCTION
in Elastic Laminate material model, 248
MULTIPLE INTERACTIONS
in Contact Definition, 488
description of, 545
MULTIPLE INTERACTIONS WITH ANGLE
in Contact Definition, 488
description of, 545

NAIR
in Low Density Foam material model, 236
NEAREST ELEMENT COPY
in Procedural Transfer, 757
description of, 758
NEW MATERIAL
in Particle Conversion, 355
NEW MESH MAX EDGE LENGTH RATIO
in Remesh, 366
usage in, 367
NEW MESH MIN EDGE LENGTH RATIO
in Remesh, 366
usage in, 367
NEW MESH MIN SHAPE
in Remesh, 366
usage in, 367
NODAL
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
NODAL DIAGONAL SCALE
in CG, 130
description of, 134
NODAL DIAGONAL SHIFT
in CG, 130
description of, 134
NODAL PRECONDITIONER METHOD
in Full Tangent Preconditioner, 141
description of, 142
NODAL PROBE FACTOR
in CG, 130
description of, 134
NODAL VARIABLES
in Results Output, 577
description of, 580
NODE
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
NODE BASED ALPHA FACTOR
in Solid Section, 297
NODE BASED BETA FACTOR
in Solid Section, 297
NODE BASED STABILIZATION METHOD
in Solid Section, 297
NODE BASED TIME STEP PARAMETERS, 112
NODE SET
description of, 376
in Adaptive Refinement, 370
usage in, 372
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
usage in, 498
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Heartbeat Output, 610

in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Mass Scaling, 114
usage in, 115
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Velocity, 395
usage in, 396
in Spot Weld, 466
in Time Step Initialization, 731
usage in, 732
in User Output, 592
usage in, 594
NODE SET SUBROUTINE
as user subroutine command line, 728
description of, 728
in Initial Condition, 380
description of, 385
in Initial Velocity, 424
description of, 426
in Pore Pressure
description of, 456
in Prescribed Acceleration, 400
description of, 402
in Prescribed Displacement, 389
description of, 392
in Prescribed Force, 438
description of, 440
in Prescribed Moment, 442
description of, 444
in Prescribed Rotation, 409
description of, 412
in Prescribed Rotational Velocity, 414
description of, 417
in Prescribed Temperature, 449
description of, 451
in Prescribed Velocity, 395
description of, 398
in Pressure, 428
description of, 430
in Time Step Initialization, 731
description of, 732
in Traction, 434
description of, 436
in User Output, 592
description of, 596
NODE SET VARIABLES
in Results Output, 577
description of, 581
NODE SETS
in Torsional Spring Mechanism, 337
NODE SETS TO DEFINE BODY AXIS
in Cavity Expansion, 462
NODE VARIABLES
in Results Output, 577
description of, 580
NODESET
in Heartbeat Output, 610
in Heartbeat Output – for mesh entities variables
description of, 613
in Heartbeat Output – for nearest point variables
description of, 613
in History Output, 601
about, 603, 612
in History Output – for mesh entities variables
description of, 604
in History Output – for nearest point variables
description of, 604
NODESET VARIABLES
in Results Output, 577
description of, 581
NORMAL
in Contact Definition – in Analytic Plane, 488
description of, 510
NORMAL CAPACITY
in Contact Definition – in Area Weld Model, 488
description of, 516
in Contact Definition – in Surface Weld Model, 488
description of, 515
in Contact Definition – in Threaded Model, 488
description of, 519
NORMAL CUTOFF DISTANCE FOR TANGENTIAL TRACTION
in Contact Definition – in Junction Model, 488
description of, 518
NORMAL DISPLACEMENT FUNCTION

in Contact Definition – in Spring Weld Model, 488
description of, 514
in Spot Weld, 466
NORMAL DISPLACEMENT SCALE FACTOR
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Spot Weld, 466
NORMAL TOLERANCE
in Contact Definition – in Interaction, 488, 530
description of, 534
in Contact Definition – in Search Options, 488, 554
description of, 555
NORMAL TRACTION FUNCTION
in Contact Definition – in Junction Model, 488
description of, 518
in Contact Definition – in Threaded Model, 488
description of, 519
NORMAL TRACTION SCALE FACTOR
in Contact Definition – in Junction Model, 488
description of, 518
in Contact Definition – in Threaded Model, 488
description of, 519
NTH11 FUNCTION
in Elastic Laminate material model, 248
NTH12 FUNCTION
in Elastic Laminate material model, 248
NTH22 FUNCTION
in Elastic Laminate material model, 248
NUM LOCAL SUBDOMAINS
in FETI Equation Solver, 149
description of, 151
NUMBER EIGENVALUES
in Lanczos Parameters, 102
NUMBER ITERATIONS
in Power Method Parameters, 109
NUMBER OF DOMAINS
in J Integral, 686
NUMBER OF INTEGRATION POINTS
in Cohesive Section, 300
in Localization Section, 301
in Shell Section, 302
NUMBER OF TIME STEPS
in Parameters For Adagio Region, 183, 184
NUMERICAL SHIFT FUNCTION
in Viscoelastic Swanson material model, 271

OBJECT TYPE
in Pressure, 428
description of, 431
OCTREE DIMENSION
in Zoltan Parameters, 365
OCTREE MAX OBJECTS
in Zoltan Parameters, 365
OCTREE METHOD
in Zoltan Parameters, 365
OCTREE MIN OBJECTS
in Zoltan Parameters, 365
OFFSET PRESSURE
in Contact Definition – in PV_Dependent Model, 488
description of, 520
OMIT BLOCK
in Finite Element Model, 284
description of, 288
ONE INCH
in Karagozian and Case concrete material model, 230
OPTIONAL
in Restart Data, 622
description of, 623
ORDINATE
in Definition for Function, 62
ORIENTATION, 69
in Shell Section, 302
ORIGIN
in Coordinate System, 73
ORIGIN NODE
in Coordinate System, 73
ORTHOGONALITY MEASURE FOR RESET
in CG, 130
description of, 139
OUTPUT DATABASE NAME
in Restart Data, 622
description of, 623
OUTPUT MESH
BLOCK_SURFACE, 587
EXPOSED_SURFACE, 587
in Results Output, 577
description of, 587
OUTPUT ON SIGNAL
in Heartbeat Output, 610
description of, 617
in History Output, 601
description of, 608
in Restart Data, 622
description of, 636
in Results Output, 577
description of, 591
OUTPUT SCHEDULER, 637
example of, 639
use of, 637
OVER ALLOCATE MEMORY
in Zoltan Parameters, 365
as default, 364
OVERLAP NORMAL TOLERANCE
in Contact Definition – in Interaction, 488, 530
description of, 534

in Contact Definition – in Remove Initial Overlap, 488
description of, 543
OVERLAP TANGENTIAL TOLERANCE
in Contact Definition – in Interaction, 488, 530
description of, 534
in Contact Definition – in Remove Initial Overlap, 488
description of, 543
OVERLAY COUNT
in Restart Data, 622
description of, 633
OVERWRITE
in History Output, 601
in Restart Data, 622
description of, 632
in Results Output, 577

P0
in Low Density Foam material model, 236
P1
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
PARAM-STRING debugMask VALUE
in FETI Equation Solver, 149
PARAM-STRING precision VALUE
in FETI Equation Solver, 149
description of, 150
PARAMETERS FOR ADAGIO REGION
in Time Stepping Block, 182
contents of, 183
PARAMETERS FOR BLOCK
in Finite Element Model, 284
about, 290
listing of, 289
PARAMETERS FOR MODEL
in Property Specification for Material command blocks
description of, 195
PARAMETERS FOR MODEL BCJ
in BCJ material model, 223
PARAMETERS FOR MODEL DUCTILE FRACTURE
in Ductile Fracture material model, 214
PARAMETERS FOR MODEL ELASTIC
in Elastic material model, 202
in Neo Hookean material model, 206
in Thermoelastic material model, 204
PARAMETERS FOR MODEL ELASTIC_3D_ORTHOTROPIC
in Elastic 3D Orthotropic material model, 237
PARAMETERS FOR MODEL ELASTIC_FRACTURE
in Elastic Fracture material model, 208
PARAMETERS FOR MODEL ELASTIC_LAMINATE
in Elastic Laminate material model, 248
PARAMETERS FOR MODEL ELASTIC_PLASTIC
in Elastic-Plastic material model, 210
PARAMETERS FOR MODEL EP_POWER_HARD
in Elastic-Plastic Power-Law Hardening material model, 212
PARAMETERS FOR MODEL FIBER_MEMBRANE
in Fiber Membrane material model, 251
PARAMETERS FOR MODEL FIBER_SHELL
in Fiber Shell material model, 254
PARAMETERS FOR MODEL FOAM_PLASTICITY
in Foam Plasticity material model, 233
in Karagozian and Case concrete material model, 230
PARAMETERS FOR MODEL INCOMPRESSIBLE_SOLID
in Incompressible Solid material model, 256
PARAMETERS FOR MODEL JOHNSON_COOK
in Johnson-Cook material model, 221
PARAMETERS FOR MODEL LOW_DENSITY_FOAM
in Low Density Foam material model, 236
PARAMETERS FOR MODEL ML_EP_FAIL
in Multilinear Elastic-Plastic Hardening Model material mode, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material mode, 218
PARAMETERS FOR MODEL MOONEY_RIVLIN
in Mooney-Rivlin material model, 259
PARAMETERS FOR MODEL NLVE_3D_ORTHOTROPIC
in NLVE 3D Orthotropic material model, 262
PARAMETERS FOR MODEL ORTHOTROPIC_CRUSH
in Orthotropic Crush material model, 242
PARAMETERS FOR MODEL ORTHOTROPIC_RATE
in Orthotropic Rate material model, 245
PARAMETERS FOR MODEL RVE, 681
PARAMETERS FOR MODEL SOIL_FOAM
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
PARAMETERS FOR MODEL STIFF_ELASTIC
in Stiff Elastic material model, 266
PARAMETERS FOR MODEL SWANSON
in Swanson material model, 268
PARAMETERS FOR MODEL VISCOELASTIC_SWANSON
in Viscoelastic Swanson material model, 271
PARAMETERS FOR MODEL WIRE_MESH
in Wire Mesh material model, 240
PARAMETERS FOR PRESTO REGION
in Time Stepping Block, 89
contents of, 90
PARTICLE CONVERSION
in Element Death, 343

description of, 355
PARTICLE EMBEDDING, 360
PARTICLE SECTION, 322
in Particle Conversion, 355
PENALTY FACTOR
in Full Tangent Preconditioner, 141
description of, 143
PERCENT CONTINUUM
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
Peridynamics
in Special Modeling Techniques, 689
PERIDYNAMICS SECTION, 329
PERIOD
in Torsional Spring Mechanism, 337
in Truss Section, 317
PERIODIC, 405
in Control Modes Region, 175
usage in, 177
PERIODIC REBALANCE
in Rebalance, 362
description of, 363
PHI
in Foam Plasticity material model, 233
in Low Density Foam material model, 236
POINT
in Coordinate System, 73
in Contact Definition – in Analytic Plane, 488
description of, 510
POINT A
in Orientation, 69
POINT B
in Orientation, 69
POINT INERTIA
in Rigid Body command block, 332
POINT MASS
in Rigid Body command block, 332
POINT MASS SECTION, 320
POINT NODE
in Coordinate System, 73
POINT ON AXIS
in Periodic, 405
POISSONS RATIO
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Mooney-Rivlin material model, 259
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in NLVE 3D Orthotropic material model, 262
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Swanson material model, 268
in Thermoelastic material model, 204
in Viscoelastic Swanson material model, 271
in Wire Mesh material model, 240
POISSONS RATIO AB
in Elastic 3D Orthotropic material model, 237
POISSONS RATIO BC
in Elastic 3D Orthotropic material model, 237
POISSONS RATIO CA
in Elastic 3D Orthotropic material model, 237
POISSONS RATIO FUNCTION
in BCJ material model, 223
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Thermoelastic material model, 204
PORE PRESSURE, 454
POWER METHOD PARAMETERS, 109
POWER METHOD TIME STEP INTERVAL
in Control Modes Region
usage in, 119
PRECISION
in Heartbeat Output, 610
PRECONDITIONER
in CG, 130
description of, 134
PRECONDITIONER ITERATION UPDATE
in CG, 130
description of, 134
PRECONDITIONING METHOD
in FETI Equation Solver, 149
description of, 150
PREDICTOR TYPE
in Loadstep Predictor
description of, 178
PRESCRIBED ACCELERATION, 400

PRESCRIBED DISPLACEMENT, 389
PRESCRIBED FORCE, 438
PRESCRIBED MOMENT, 442
PRESCRIBED QUANTITY
in Periodic, 405
PRESCRIBED ROTATION, 409
PRESCRIBED ROTATIONAL Velocity, 414
PRESCRIBED TEMPERATURE, 449
usage for thermal strains, 201
PRESCRIBED VELOCITY, 395
PRESSURE, 428
PRESSURE COEFFICIENTS
in Cavity Expansion – in Layer, 462
description of, 463
PRESSURE CUTOFF
in Soil and Crushable Foam material model, 227
PRESSURE EXPONENT
in Contact Definition – in PV_Dependent Model, 488
description of, 520
PRESSURE FUNCTION
in Karagozian and Case concrete material model, 230
in Soil and Crushable Foam material model, 227
PRESTO PROCEDURE, 76
description of, 77
PRESTO REGION
in Presto Procedure, 76
description of, 78
PROBE FACTOR
in Full Tangent Preconditioner, 141
description of, 143
PROBLEM DIMENSION
in Particle Section, 322
PROCEDURAL TRANSFER, 757
PRONY SHEAR 1
in Viscoelastic Swanson material model, 271
PRONY SHEAR 10
in Viscoelastic Swanson material model, 271
PRONY SHEAR 2
in Viscoelastic Swanson material model, 271
PRONY SHEAR 3
in Viscoelastic Swanson material model, 271
PRONY SHEAR 4
in Viscoelastic Swanson material model, 271
PRONY SHEAR 5
in Viscoelastic Swanson material model, 271
PRONY SHEAR 6
in Viscoelastic Swanson material model, 271
PRONY SHEAR 7
in Viscoelastic Swanson material model, 271
PRONY SHEAR 8
in Viscoelastic Swanson material model, 271
PRONY SHEAR 9
in Viscoelastic Swanson material model, 271
PRONY SHEAR INFINITY
in Viscoelastic Swanson material model, 271
PROPERTY SPECIFICATION FOR MATERIAL, 194
about, 291
PSI EQ 2T
in NLVE 3D Orthotropic material model, 262
PSI EQ 3T
in NLVE 3D Orthotropic material model, 262
PSI EQ 4T
in NLVE 3D Orthotropic material model, 262
PSI EQ XT 1
in NLVE 3D Orthotropic material model, 262
PSI EQ XT 2
in NLVE 3D Orthotropic material model, 262
PSI EQ XT 3
in NLVE 3D Orthotropic material model, 262
PSI EQ XTT 1
in NLVE 3D Orthotropic material model, 262
PSI EQ XTT 2
in NLVE 3D Orthotropic material model, 262
PSI EQ XTT 3
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 11
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 12
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 13
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 22
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 23
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 33
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 44
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 55
in NLVE 3D Orthotropic material model, 262
PSI EQ XX 66
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 11
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 12
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 13
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 22
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 23
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 33
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 44

in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 55
in NLVE 3D Orthotropic material model, 262
PSI EQ XXT 66
in NLVE 3D Orthotropic material model, 262
PSI POT TT
in NLVE 3D Orthotropic material model, 262
PSI POT TTT
in NLVE 3D Orthotropic material model, 262
PSI POT TTTT
in NLVE 3D Orthotropic material model, 262
PSI POT XT 1
in NLVE 3D Orthotropic material model, 262
PSI POT XT 2
in NLVE 3D Orthotropic material model, 262
PSI POT XT 3
in NLVE 3D Orthotropic material model, 262
PSI POT XTT 1
in NLVE 3D Orthotropic material model, 262
PSI POT XTT 2
in NLVE 3D Orthotropic material model, 262
PSI POT XTT 3
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 11
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 12
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 13
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 22
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 23
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 33
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 44
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 55
in NLVE 3D Orthotropic material model, 262
PSI POT XXT 66
in NLVE 3D Orthotropic material model, 262
PUSHBACK FACTOR
in Contact Definition – in Interaction
description of, 536
PV_DEPENDENT MODEL
in Contact Definition, 488
description of, 520

Q1
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
QUADRATIC BULK VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292

R DISPLACEMENT FUNCTION
in Line Weld, 471
R DISPLACEMENT SCALE FACTOR
in Line Weld, 471
R ROTATION FUNCTION
in Line Weld, 471
R ROTATION SCALE FACTOR
in Line Weld, 471
R1
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
RADIAL AXIS
in Prescribed Displacement, 389
description of, 391
in Prescribed Velocity, 395
description of, 396
RADIUS
in Adaptive Refinement, 370
usage in, 371
in Contact Definition – in Analytic Cylinder, 488
description of, 511
in Contact Definition – in Analytic Sphere, 488
description of, 511
RADIUS MESH VARIABLE
in Particle Section, 322
RADIUS MESH VARIABLE TIME STEP
in Particle Section, 322
RATE FUNCTION
in Orthotropic Rate material model, 245
RATE SENSITIVITY FUNCTION
in Karagozian and Case concrete material model, 230
READ VARIABLE
in Initial Condition, 380
description of, 384
in Pore Pressure, 454
description of, 456
in Prescribed Acceleration, 400
description of, 403
in Prescribed Displacement, 389
description of, 392
in Prescribed Rotation, 409
description of, 412
in Prescribed Rotational Velocity, 414
description of, 417
in Prescribed Temperature, 449
description of, 451
in Prescribed Velocity, 395
description of, 398
in Pressure, 428
description of, 431
REBALANCE, 362

REBALANCE STEP INTERVAL
in Rebalance, 362
RECEIVE BLOCKS
in Procedural Transfer, 757
description of, 758
RECEIVE COORDINATES
in Procedural Transfer, 757
description of, 758
RECEIVE FROM TRANSFER
in Pore Pressure, 454
in Prescribed Temperature, 449
RECTILINEAR RCB BLOCKS
in Zoltan Parameters, 365
REF GLASSY C11
in NLVE 3D Orthotropic material model, 262
REF GLASSY C12
in NLVE 3D Orthotropic material model, 262
REF GLASSY C13
in NLVE 3D Orthotropic material model, 262
REF GLASSY C22
in NLVE 3D Orthotropic material model, 262
REF GLASSY C23
in NLVE 3D Orthotropic material model, 262
REF GLASSY C33
in NLVE 3D Orthotropic material model, 262
REF GLASSY C44
in NLVE 3D Orthotropic material model, 262
REF GLASSY C55
in NLVE 3D Orthotropic material model, 262
REF GLASSY C66
in NLVE 3D Orthotropic material model, 262
REF GLASSY CTE1
in NLVE 3D Orthotropic material model, 262
REF GLASSY CTE2
in NLVE 3D Orthotropic material model, 262
REF GLASSY CTE3
in NLVE 3D Orthotropic material model, 262
REF GLASSY HCAPACITY
in NLVE 3D Orthotropic material model, 262
REF PSIA 11
in NLVE 3D Orthotropic material model, 262
REF PSIA 12
in NLVE 3D Orthotropic material model, 262
REF PSIA 13
in NLVE 3D Orthotropic material model, 262
REF PSIA 22
in NLVE 3D Orthotropic material model, 262
REF PSIA 23
in NLVE 3D Orthotropic material model, 262
REF PSIA 33
in NLVE 3D Orthotropic material model, 262
REF PSIA 44
in NLVE 3D Orthotropic material model, 262
REF PSIA 55
in NLVE 3D Orthotropic material model, 262
REF PSIA 66
in NLVE 3D Orthotropic material model, 262
REF PSIB 1
in NLVE 3D Orthotropic material model, 262
REF PSIB 2
in NLVE 3D Orthotropic material model, 262
REF PSIB 3
in NLVE 3D Orthotropic material model, 262
REF PSIC
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C11
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C12
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C13
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C22
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C23
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C33
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C44
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C55
in NLVE 3D Orthotropic material model, 262
REF RUBBERY C66
in NLVE 3D Orthotropic material model, 262
REF RUBBERY CTE1
in NLVE 3D Orthotropic material model, 262
REF RUBBERY CTE2
in NLVE 3D Orthotropic material model, 262
REF RUBBERY CTE3
in NLVE 3D Orthotropic material model, 262
REF RUBBERY HCAPACITY
in NLVE 3D Orthotropic material model, 262
REFERENCE
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
REFERENCE AXIS, 420
in Beam Section, 312
in Periodic, 405
REFERENCE AXIS X FUNCTION
in Reference Axis, 420
description of, 421
REFERENCE AXIS Y FUNCTION
in Reference Axis, 420
description of, 421
REFERENCE AXIS Z FUNCTION

in Reference Axis, 420
description of, 421
REFERENCE DENSITY
in NLVE 3D Orthotropic material model, 262
REFERENCE LOCATION
in Rigid Body command block, 332
Reference Mesh
for Control Modes, 174
for Explicit Control Modes, 116
REFERENCE POINT
in Fluid Pressure, 459
in Fluid Pressure
usage in, 460
REFERENCE PRESSURE
in Contact Definition – in PV_Dependent Model, 488
description of, 520
REFERENCE STRAIN
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Incompressible Solid material model, 256
in Mooney-Rivlin material model, 259
in Stiff Elastic material model, 266
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
REFERENCE TEMP
in NLVE 3D Orthotropic material model, 262
RELAX TIME
in NLVE 3D Orthotropic material model, 262
REMESH, 366
REMESH AT MAX EDGE LENGTH RATIO
in Remesh, 366
usage in, 367
REMESH AT MIN EDGE LENGTH RATIO
in Remesh, 366
usage in, 367
REMESH AT SHAPE
in Remesh, 366
usage in, 367
REMESH BLOCK SET, 369
in Remesh, 366
REMOVE BLOCK
description of, 377
in Adaptive Refinement, 370
usage in, 372
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
in Contact Definition –in Contact Surface (block)
usage in, 498
in Element Death, 343
description of, 345
in Finite Element Model, 284
description of, 292
in Finite Element Model – in Parameters For Block, 289
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Line Weld, 471
in Mass Properties, 341
usage in, 341
in Mass Scaling, 114
usage in, 115
in Pore Pressure, 454
usage in, 455
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Temperature, 449
usage in, 450
in Prescribed Velocity, 395
usage in, 396
in Procedural Transfer, 757
description of, 758
in Remesh Block Set, 369
in Time Step Initialization, 731
usage in, 732
in User Output, 592
usage in, 594
in Viscous Damping, 474
usage in, 474
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
REMOVE GAPS AND OVERLAPS
in MPC, 478
REMOVE INITIAL OVERLAP
in Contact Definition, 488
description of, 543
REMOVE NODE SET
description of, 377

in Adaptive Refinement, 370
usage in, 372
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
usage in, 498
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Mass Scaling, 114
usage in, 115
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Velocity, 395
usage in, 396
in Spot Weld, 466
in Time Step Initialization, 731
usage in, 732
in User Output, 592
usage in, 594
REMOVE SURFACE
description of, 377
in Adaptive Refinement, 370
usage in, 372
in Cavity Expansion, 462
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
usage in, 498
in Contact Definition – in Shell Lofting
description of, 550
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Line Weld , 471
in Mass Scaling, 114
usage in, 115
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409
usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Velocity, 395
usage in, 396
in Pressure, 428
description of, 429
in Silent Boundary, 465
in Spot Weld, 466
in Time Step Initialization, 731
usage in, 732
in Traction, 434
description of, 435
in User Output, 592
usage in, 594
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
RENUMBER PARTITIONS
in Zoltan Parameters, 365
RESET AT NEW PERIOD
in Adaptive Time Stepping
description of, 188
RESET LIMITS
in CG, 130
description of, 139
RESIDUAL NORM TOLERANCE
in FETI Equation Solver, 149
description of, 150
RESIDUAL ROUNDOFF TOLERANCE
in CG
description of, 132
in CG, 130
RESOLVE MULTIPLE MPCS, 478
RESTART, 61
about, 60
RESTART DATA, 622
about, 60

about auto read and write, 624
about overwriting, 630
about recovering, 631
about user-controlled read and write, 627
RESTART TIME, 61
about, 60
with Restart Data, 622
RESULTS OUTPUT, 577
in Control Modes Region
usage in, 121
REUSE CUTS
in Zoltan Parameters, 365
as default, 364
RHO
in BCJ material model, 223
RIGID BODIES FROM ATTRIBUTES
in Beam Section, 312
in Membrane Section, 308
in Point Mass Section, 320
in Shell Section, 302
in Solid Section, 297
in Truss Section, 317
RIGID BODY
as command block, 332
in Beam Section, 312
in Membrane Section, 308
in Point Mass Section, 320
in Shell Section, 302
in Solid Section, 297
in Truss Section, 317
ROTATE BLOCK
in Initial Mesh Modification, 379
ROTATION
in Reference Axis, 420
in Reference Axis Rotation
description of, 422
ROTATION ABOUT
in Orientation, 69
ROTATIONAL SCALE FACTOR
in Centripetal Force, 448
ROTATIONAL VELOCITY
in Reference Axis, 420
in Reference Axis Rotation
description of, 422
ROTATIONAL VELOCITY FUNCTION
in Centripetal Force, 448
RVE REGION, 682

S DISPLACEMENT FUNCTION
in Line Weld, 471
S DISPLACEMENT SCALE FACTOR
in Line Weld, 471
S ROTATION FUNCTION
in Line Weld, 471
S ROTATION SCALE FACTOR
in Line Weld, 471
SCALE FACTOR
in Gravity, 446
in Initial Condition, 380
description of, 385
in Lanczos Parameters, 102
in Loadstep Predictor, 178
description of, 179
in Periodic, 405
in Pore Pressure, 454
description of, 457
in Power Method Parameters, 109
in Prescribed Acceleration, 400
description of, 404
in Prescribed Displacement, 389
description of, 393
in Prescribed Force, 438
description of, 441
in Prescribed Moment, 442
description of, 445
in Prescribed Rotation, 409
description of, 413
in Prescribed Rotational Velocity, 414
description of, 418
in Prescribed Temperature, 449
description of, 453
in Prescribed Velocity, 395
description of, 399
in Pressure, 428
description of, 432
in Reference Axis, 420
in REFERENCE AXIS ROTATION
usage in, 422
in Stiff Elastic material model, 266
in Traction, 434
description of, 437
in Volume Repulsion Old, 476
SCALING FUNCTION
in Incompressible Solid material model, 256
SEARCH OPTIONS
in Contact Definition, 488
description of, 554
SEARCH TOLERANCE
in Contact Definition – in Search Options, 488, 554
description of, 555
in Line Weld, 471
in MPC, 478
usage in, 478, 479
in Periodic, 405
in Spot Weld, 466
SECOND ALPHA
in Elastic 3D Orthotropic material model, 237
in NLVE 3D Orthotropic material model, 262

SECOND DIRECTION FOR ROTATION
in Elastic 3D Orthotropic material model, 237
in NLVE 3D Orthotropic material model, 262
SECOND SURFACE
in Spot Weld, 466
SECTION
in Beam Section, 312
in Finite Element Model – in Parameters For Block, 289
description of, 292
general overview, 290
Section command blocks
about, 297
SELF CONTACT
in Contact Definition – in Interaction Defaults, 488, 525
description of, 526
SEND BLOCKS
in Procedural Transfer, 757
description of, 758
SEND COORDINATES
in Procedural Transfer, 757
description of, 758
SET RCB DIRECTIONS
in Zoltan Parameters, 365
SHEAR EXPONENT
in Foam Plasticity material model, 233
SHEAR HARDENING
in Foam Plasticity material model, 233
SHEAR MODULUS
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Mooney-Rivlin material model, 259
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in NLVE 3D Orthotropic material model, 262
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Swanson material model, 268
in Thermoelastic material model, 204
in Viscoelastic Swanson material model, 271
in Wire Mesh material model, 240
SHEAR MODULUS AB
in Elastic 3D Orthotropic material model, 237
SHEAR MODULUS BC
in Elastic 3D Orthotropic material model, 237
SHEAR MODULUS CA
in Elastic 3D Orthotropic material model, 237
SHEAR RELAX TIME 1
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 10
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 2
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 3
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 4
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 5
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 6
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 7
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 8
in Viscoelastic Swanson material model, 271
SHEAR RELAX TIME 9
in Viscoelastic Swanson material model, 271
SHEAR STRENGTH
in Foam Plasticity material model, 233
SHELL LOFTING
in Contact Definition, 488
description of, 550
SHELL OVERLAP ITERATIONS
in Contact Definition – in Remove Initial Overlap, 488
description of, 543
SHELL OVERLAP TOLERANCE
in Contact Definition – in Remove Initial Overlap, 488
description of, 543
SHELL SECTION, 302
SHIFT REF VALUE
in NLVE 3D Orthotropic material model, 262
SIERRA, 59
SILENT BOUNDARY, 465
SINGLE RATE ENHANCEMENT

in Karagozian and Case concrete material model, 230
SKIN ALL BLOCKS
in Contact Definition, 488
description of, 500
SLAVE
in Contact Definition – in Interaction, 488, 530
description of, 531
in Periodic, 405
SLAVE BLOCK
in MPC, 478
usage in, 478
SLAVE NODE SET
in MPC, 478
usage in, 478
SLAVE NODES
in MPC, 478
usage in, 478
SLAVE SURFACE
in MPC, 478
usage in, 478
SLIP SCALE FACTOR
in Loadstep Predictor, 178
description of, 179
SMALL NUMBER OF ITERATIONS
in Full Tangent Preconditioner, 141
description of, 145
SMALL STRAIN
in Lanczos Parameters, 102
in Power Method Parameters, 109
SOLID MECHANICS USE MODEL
in Finite Element Model – in Parameters For Block, 289
description of, 291
SOLID SECTION, 297
SOLVER, 126
in Control Modes Region, 175
usage in, 176
SPECIFIC HEAT
in BCJ material model, 223
SPH ALPHAQ PARAMETER
in Particle Section, 322
SPH BETAQ PARAMETER
in Particle Section, 322
SPH DECOUPLE STRAINS
in SPH Options, 324
SPH OPTIONS, 324
SPH SYMMETRY PLANE
in SPH Options, 324
SPHERE INITIAL RADIUS
in Particle Section, 322
SPOT WELD, 466
SPRING SECTION, 318
SPRING WELD MODEL
in Contact Definition, 488
description of, 514
STAGNATION THRESHOLD
in Full Tangent Preconditioner, 141
description of, 147
START TIME
in Heartbeat Output, 610
description of, 615
in History Output, 601
description of, 606
in Output Scheduler, 637
description of, 638
in Restart Data, 622
description of, 632
in Results Output, 577
description of, 589
in Time Stepping Block, 89, 182
STARTING VECTOR
in Lanczos Parameters, 102
in Power Method Parameters, 109
STATIC COEFFICIENT
in Contact Definition – in PV_Dependent Model, 488
description of, 520
STEP INTERVAL
in Node Based Time Step Parameters, 112
in Parameters For Presto Region, 90
description of, 91
in Rebalance
description of, 363
STIFFNESS DAMPING COEFFICIENT
in Viscous Damping, 474
description of, 475
STIFFNESS MATRIX
in Superelement Section, 326
description of, 327
STIFFNESS SCALE
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
STRAIN INCREMENTATION
in Solid Section, 297
STREAM NAME
in Heartbeat Output, 610
STRUCTURE NAME
in Mass Properties, 341
description of, 342
Submodel, 483
SUBROUTINE DEBUGGING OFF
as user subroutine command line, 728
description of, 729
in Element Death, 343
usage in, 350
in Initial Condition, 380
usage in, 385

in Initial Velocity, 424
usage in, 426
in Pore Pressure, 454
usage in, 456
in Prescribed Acceleration, 400
usage in, 403
in Prescribed Displacement, 389
usage in, 392
in Prescribed Force, 438
usage in, 440
in Prescribed Moment, 442
usage in, 444
in Prescribed Rotation, 409
usage in, 412
in Prescribed Rotational Velocity, 414
usage in, 417
in Prescribed Temperature, 449
usage in, 451
in Prescribed Velocity, 395
usage in, 398
in Pressure, 428
usage in, 431
in Time Step Initialization, 731
usage in, 732
in Traction, 434
usage in, 436
in User Output, 592
usage in, 597
SUBROUTINE DEBUGGING ON
as user subroutine command line, 728
description of, 729
in Element Death, 343
usage in, 350
in Initial Condition, 380
usage in, 385
in Initial Velocity, 424
usage in, 426
in Pore Pressure, 454
usage in, 456
in Prescribed Acceleration, 400
usage in, 403
in Prescribed Displacement, 389
usage in, 392
in Prescribed Force, 438
usage in, 440
in Prescribed Moment, 442
usage in, 444
in Prescribed Rotation, 409
usage in, 412
in Prescribed Rotational Velocity, 414
usage in, 417
in Prescribed Temperature, 449
usage in, 451
in Prescribed Velocity, 395
usage in, 398
in Pressure, 428
usage in, 431
in Time Step Initialization, 731
usage in, 732
in Traction, 434
usage in, 436
in User Output, 592
usage in, 597
SUBROUTINE INTEGER PARAMETER
as user subroutine command line, 728
description of, 729
in Element Death, 343
usage in, 350
in Initial Condition, 380
usage in, 385
in Initial Velocity, 424
usage in, 426
in Pore Pressure, 454
usage in, 456
in Prescribed Acceleration, 400
usage in, 403
in Prescribed Displacement, 389
usage in, 392
in Prescribed Force, 438
usage in, 440
in Prescribed Moment, 442
usage in, 444
in Prescribed Rotation, 409
usage in, 412
in Prescribed Rotational Velocity, 414
usage in, 417
in Prescribed Temperature, 449
usage in, 451
in Prescribed Velocity, 395
usage in, 398
in Pressure, 428
usage in, 431
in Time Step Initialization, 731
usage in, 732
in Traction, 434
usage in, 436
in User Output, 592
usage in, 597
SUBROUTINE REAL PARAMETER
as user subroutine command line, 728
description of, 729
in Element Death, 343
usage in, 350
in Initial Condition, 380
usage in, 385
in Initial Velocity, 424
usage in, 426
in Pore Pressure, 454

usage in, 456
in Prescribed Acceleration, 400
usage in, 403
in Prescribed Displacement, 389
usage in, 392
in Prescribed Force, 438
usage in, 440
in Prescribed Moment, 442
usage in, 444
in Prescribed Rotation, 409
usage in, 412
in Prescribed Rotational Velocity, 414
usage in, 417
in Prescribed Temperature, 449
usage in, 451
in Prescribed Velocity, 395
usage in, 398
in Pressure, 428
usage in, 431
in Time Step Initialization, 731
usage in, 732
in Traction, 434
usage in, 436
in User Output, 592
usage in, 597
usage with query function, 703
SUBROUTINE STRING PARAMETER
as user subroutine command line, 728
description of, 729
in Element Death, 343
usage in, 350
in Initial Condition, 380
usage in, 385
in Initial Velocity, 424
usage in, 426
in Pore Pressure, 454
usage in, 456
in Prescribed Acceleration, 400
usage in, 403
in Prescribed Displacement, 389
usage in, 392
in Prescribed Force, 438
usage in, 440
in Prescribed Moment, 442
usage in, 444
in Prescribed Rotation, 409
usage in, 412
in Prescribed Rotational Velocity, 414
usage in, 417
in Prescribed Temperature, 449
usage in, 451
in Prescribed Velocity, 395
usage in, 398
in Pressure, 428
usage in, 431
in Time Step Initialization, 731
usage in, 732
in Traction, 434
usage in, 436
in User Output, 592
usage in, 597
Subsetting
in Results Output, 587
SUMMARY OUTPUT STEP INTERVAL
in Element Death, 343
description of, 351
SUMMARY OUTPUT TIME INTERVAL
in Element Death, 343
description of, 351
SUPERELEMENT SECTION, 326
Support, 56
SURFACE
in Fluid Pressure, 459
description of, 376
in Adaptive Refinement, 370
usage in, 372
in Cavity Expansion , 462
in Centripetal Force, 448
in Contact Definition – in Contact Surface (block), 488
in Contact Definition – in Contact Surface(block)
usage in, 498
in Contact Definition – in Shell Lofting
description of, 550
in Fixed Displacement, 387
usage in, 388
in Fixed Rotation, 407
usage in, 407
in Fluid Pressure
usage in, 460
in Gravity, 446
usage in, 446
in Initial Condition, 380
usage in, 381
in Initial Velocity, 424
usage in, 425
in Line Weld, 471
in Mass Scaling, 114
usage in, 115
in Prescribed Acceleration, 400
usage in, 401
in Prescribed Displacement, 389
usage in, 390
in Prescribed Force, 438
usage in, 439
in Prescribed Moment, 442
usage in, 443
in Prescribed Rotation, 409

usage in, 411
in Prescribed Rotational Velocity, 414
usage in, 415
in Prescribed Velocity, 395
usage in, 396
in Pressure, 428
description of, 429
in Silent Boundary, 465
in Spot Weld, 466
in Time Step Initialization, 731
usage in, 732
in Traction, 434
description of, 435
in User Output, 592
usage in, 594
in Volume Repulsion Old
usage in, 476
in Volume Repulsion Old – in BLOCK SET, 476
SURFACE EFFECT
in Cavity Expansion – in Layer, 462
description of, 464
SURFACE NORMAL SMOOTHING
in Contact Definition, 488
description of, 547
SURFACE SUBROUTINE
as user subroutine command line, 728
description of, 729
in Initial Condition, 380
description of, 385
in Pressure, 428
description of, 430
in Time Step Initialization, 731
description of, 732
in User Output, 592
description of, 596
SURFACE WELD MODEL
in Contact Definition, 488
description of, 515
SURFACES
in Contact Definition – in Interaction, 488, 530
description of, 531
in Contact Definition – in Interaction Defaults, 488, 525
description of, 526
SYMMETRY
in J Integral, 686
SYNCHRONIZE OUTPUT
in Heartbeat Output, 610
description of, 617
in History Output, 601
description of, 607
in Restart Output, 622
description of, 635
in Results Output, 577
description of, 590
SYSTEM
in Orientation, 69

T 2DERIV PSIC
in NLVE 3D Orthotropic material model, 262
T AXIS
in Beam Section, 312
T DERIV GLASSY C11
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C12
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C13
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C22
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C23
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C33
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C44
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C55
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY C66
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY CTE1
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY CTE2
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY CTE3
in NLVE 3D Orthotropic material model, 262
T DERIV GLASSY HCAPACITY
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 11
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 12
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 13
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 22
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 23
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 33
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 44
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 55
in NLVE 3D Orthotropic material model, 262
T DERIV PSIA 66
in NLVE 3D Orthotropic material model, 262
T DERIV PSIB 1
in NLVE 3D Orthotropic material model, 262

T DERIV PSIB 2
in NLVE 3D Orthotropic material model, 262
T DERIV PSIB 3
in NLVE 3D Orthotropic material model, 262
T DERIV PSIC
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C11
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C12
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C13
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C22
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C23
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C33
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C44
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C55
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY C66
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY CTE1
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY CTE2
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY CTE3
in NLVE 3D Orthotropic material model, 262
T DERIV RUBBERY HCAPACITY
in NLVE 3D Orthotropic material model, 262
T DISPLACEMENT FUNCTION
in Line Weld, 471
T DISPLACEMENT SCALE FACTOR
in Line Weld, 471
T FUNCTION
in Orthotropic Rate material model, 245
T ROTATION FUNCTION
in Line Weld, 471
T ROTATION SCALE FACTOR
in Line Weld, 471
TANGENT DIAGONAL SCALE
in Full Tangent Preconditioner, 141
description of, 143
TANGENT DIAGONAL SHIFT
in Full Tangent Preconditioner, 141
description of, 143
TANGENTIAL CAPACITY
in Contact Definition – in Area Weld Model, 488
description of, 516
in Contact Definition – in Surface Weld Model, 488
description of, 515
in Contact Definition – in Threaded Model, 488
description of, 519
TANGENTIAL DISPLACEMENT FUNCTION
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Spot Weld, 466
TANGENTIAL DISPLACEMENT SCALE FACTOR
in Contact Definition – in Spring Weld Model, 488
description of, 514
in Spot Weld, 466
TANGENTIAL TOLERANCE
in Contact Definition – in Interaction, 488, 530
description of, 534
in Contact Definition – in Search Options, 488, 554
description of, 555
TANGENTIAL TRACTION FUNCTION
in Contact Definition – in Junction Model, 488
description of, 518
in Contact Definition – in Threaded Model, 488
description of, 519
TANGENTIAL TRACTION GAP FUNCTION
in Contact Definition – in Threaded Model, 488
description of, 519
TANGENTIAL TRACTION GAP SCALE FACTOR
in Contact Definition – in Threaded Model, 488
description of, 519
TANGENTIAL TRACTION SCALE FACTOR
in Contact Definition – in Junction Model, 488
description of, 518
in Contact Definition – in Threaded Model, 488
description of, 519
TARGET AXIAL FORCE INCREMENT
in Control Stiffness, 167
description of, 168
TARGET E
in Incompressible Solid material model, 256
in Mooney-Rivlin material model, 259
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
TARGET E FUNCTION
in Mooney-Rivlin material model, 259
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
TARGET ITERATIONS
in Adaptive Time Stepping
description of, 186
TARGET PRESSURE INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RELATIVE AXIAL FORCE INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RELATIVE CONTACT RESIDUAL
in Control Contact, 153
description of, 157

TARGET RELATIVE PRESSURE INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RELATIVE RESIDUAL
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
TARGET RELATIVE SDEV INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RELATIVE STRAIN INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RELATIVE STRESS INCREMENT
in Control Stiffness, 167
description of, 168
TARGET RESIDUAL
in CG, 130
description of, 132
in Control Contact, 153
description of, 157
in Control Stiffness, 167
description of, 168
TARGET SDEV INCREMENT
in Control Stiffness, 167
description of, 168
TARGET SMOOTHING RELATIVE RESIDUAL
in Full Tangent Preconditioner, 141
description of, 145
TARGET SMOOTHING RESIDUAL
in Full Tangent Preconditioner, 141
description of, 145
TARGET STRESS INCREMENT
in Control Stiffness, 167
description of, 168
TARGET TIME STEP
in Mass Scaling, 114
description of, 115
TEMPERATURE TYPE
in Prescribed Temperature
description of, 451
TEMPO
in BCJ material model, 223
TENSILE STRENGTH
in Karagozian and Case concrete material model, 230
TENSILE TEST FUNCTION
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
TENSION
in Wire Mesh material model, 240
TENSION RELEASE
in Contact Definition – in Interaction
description of, 536
TENSION RELEASE FUNCTION
in Contact Definition – in Interaction
description of, 537
TERMINATION TIME
in Heartbeat Output, 610
description of, 616
in History Output, 601
description of, 607
in Output Scheduler, 637
description of, 639
in Restart Data, 622
description of, 633
in Results Output, 577
description of, 590
in Time Control, 89, 182
THERMAL CONSTANT
in Power Law Creep material model, 225
THERMAL EXPANSION FUNCTION
in Mooney-Rivlin material model, 259
in Swanson material model, 268
in Viscoelastic Swanson material model, 271
THERMAL STRAIN AA FUNCTION
in Elastic 3D Orthotropic material model, 237
THERMAL STRAIN BB FUNCTION
in Elastic 3D Orthotropic material model, 237
THERMAL STRAIN CC FUNCTION
in Elastic 3D Orthotropic material model, 237
THERMAL STRAIN FUNCTION
description of, 199
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245

in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Thermoelastic material model, 204
in Wire Mesh material model, 240
usage of, 201
THERMAL STRAIN X FUNCTION
description of, 199
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Thermoelastic material model, 204
in Wire Mesh material model, 240
usage of, 201
THERMAL STRAIN Y FUNCTION
description of, 199
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Thermoelastic material model, 204
in Wire Mesh material model, 240
usage of, 201
THERMAL STRAIN Z FUNCTION
description of, 199
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Thermoelastic material model, 204
in Wire Mesh material model, 240
usage of, 201
THETA
in Elastic Laminate material model, 248
in Periodic, 405
THETA OPT
in BCJ material model, 223
THICKNESS
in Membrane Section, 308
in Shell Section, 302
THICKNESS MESH VARIABLE
in Membrane Section, 308
in Shell Section, 302

THICKNESS SCALE FACTOR
in Membrane Section, 308
in Shell Section, 302
THICKNESS TIME STEP
in Membrane Section, 308
in Shell Section, 302
THREADED MODEL
in Contact Definition, 488
description of, 519
TIED MODEL
in Contact Definition, 488
description of, 513
TIED NODE SET
in MPC, 478
TIED NODES
in MPC, 478
usage in, 481
TIME
in Initial Condition, 380
description of, 384
in Pore Pressure, 454
description of, 456
in Prescribed Acceleration, 400
description of, 403
in Prescribed Displacement, 389
description of, 392
in Prescribed Rotation, 409
description of, 412
in Prescribed Rotational Velocity, 414
description of, 417
in Prescribed Temperature, 449
description of, 451
in Prescribed Velocity, 395
description of, 398
in Pressure, 428
description of, 431
TIME CONTROL
example of, 92, 189
in Adagio Procedure, 76
contents and description of, 182
overview, 182
in Presto Procedure, 76
contents and description of, 89
general layout of, 87
overview, 86
usage of, 87
in Procedure
about, 77
TIME INCREMENT
in Parameters For Adagio Region, 183, 184
TIME INCREMENT FUNCTION
in Parameters For Adagio Region, 183, 184
TIME INTEGRATION CONTROL
in Implicit Dynamics, 190
description of, 191
TIME STEP INCREASE FACTOR
in Parameters For Presto Region, 90
description of, 91
usage of, 88
TIME STEP INITIALIZATION, 731
TIME STEP LIMIT
in Node Based Time Step Parameters, 112
TIME STEP RATIO FUNCTION
in Control Modes Region, 117
usage in, 119
TIME STEP RATIO SCALING
in Control Modes Region, 117
usage in, 119
TIME STEP SCALE FACTOR
in Parameters For Presto Region, 90
description of, 91
usage of, 88
with Element Numerical Formulation, 295
with Peridynamics, 693
TIME STEPPING BLOCK
in Time Control, 89, 182
description of, 89, 183
TIMESTEP ADJUSTMENT INTERVAL
in Heartbeat Output, 610
description of, 615
in History Output, 601
description of, 606
in Output Scheduler, 637
description of, 638
in Restart Data, 622
description of, 632
in Results Output, 577
description of, 589
TIMESTEP FORMAT
in Heartbeat Output, 610
TIP RADIUS
in Cavity Expansion, 462
TITLE, 60
in Heartbeat Output, 610
in History Output, 601
in Results Output, 577
TL FUNCTION
in Orthotropic Rate material model, 245
TORQUE
in Reference Axis, 420
in Reference Axis Rotation
description of, 422
TORSIONAL SPRING MECHANISM, 337
TORSIONAL STIFFNESS
in Torsional Spring Mechanism, 337
TRACTION, 434
TRACTION DISPLACEMENT FUNCTION

in Contact Definition – in Cohesive Zone Model, 488
description of, 517
TRACTION DISPLACEMENT SCALE FACTOR
in Contact Definition – in Cohesive Zone Model, 488
description of, 517
Transfers, 756
TRANSFORMATION TYPE
in Procedural Transfer, 757
description of, 758
TRANSVERSE SHEAR HOURGLASS STIFFNESS
in Finite Element Model – in Parameters For Block, 289
description of, 292
TRANSVERSE SHEAR HOURGLASS VISCOSITY
in Finite Element Model – in Parameters For Block, 289
description of, 292
TRUSS SECTION, 317
TX
in Orthotropic Rate material model, 245
TY
in Orthotropic Rate material model, 245
TYPE
in Coordinate System, 73
in Definition For Function, 62
in Loadstep Predictor, 178
in User Variable, 734
TZ
in Orthotropic Rate material model, 245

UNLOAD BULK MODULUS FUNCTION
in Karagozian and Case concrete material model, 230
UPDATE ALL SURFACES FOR ELEMENT DEATH
in Contact Definition, 488
UPDATE ON TIME STEP CHANGE
in Lanczos Parameters, 102
in Power Method Parameters, 109
UPDATE STEP INTERVAL
in Lanczos Parameters, 102
in Power Method Parameters, 109
USE FINITE ELEMENT MODEL, 80
in Control Modes Region, 117, 175
usage in, 118, 176
USE HHT INTEGRATION
in Implicit Dynamics, 190
description of, 191
USE OUTPUT SCHEDULER
example of, 639
in Heartbeat Output, 610
description of, 617
in History Output, 601
description of, 608
in Restart Data, 622
description of, 635
in Results Output, 577
description of, 591
USE WITH RESTART
in User Variable, 734
USER INTEGRATION RULE
in Shell Section, 302
USER OUTPUT, 592
example of, 742
USER SEARCH BOX
in Contact Definition, 488
description of, 557
USER SUBROUTINE FILE, 61
example of, 737
usage in context, 728
USER SUBROUTINE MODEL
in Contact Definition, 488
description of, 523
User Subroutines
aupst_check_elem_var, 708
aupst_check_global_var, 717
aupst_check_node_var, 708
aupst_cyl_transform, 751
aupst_evaluate_function, 707
aupst_get_elem_centroid, 726
aupst_get_elem_nodes, 721
aupst_get_elem_topology, 721
aupst_get_elem_var, 708
aupst_get_elem_var_offset, 708
aupst_get_face_nodes, 721
aupst_get_face_topology, 721
aupst_get_global_var, 717
aupst_get_integer_param, 703
aupst_get_node_var, 708
aupst_get_point, 726
aupst_get_proc_num, 726
aupst_get_real_param, 703
aupst_get_string_param, 703
aupst_get_time, 707
aupst_local_put_global_var, 717
aupst_put_elem_var, 708
aupst_put_elem_var_offset, 708
aupst_put_global_var, 717
aupst_put_node_var, 708
aupst_rec_transform, 752
copy_data, 753
ELEMENT BLOCK SUBROUTINE, 728
HEARTBEAT OUTPUT, 610
HISTORY OUTPUT, 601
NODE SET SUBROUTINE, 728
RESULTS OUTPUT, 577
SUBROUTINE DEBUGGING OFF, 728

SUBROUTINE DEBUGGING ON, 728
SUBROUTINE INTEGER PARAMETER, 728
SUBROUTINE REAL PARAMETER, 728
SUBROUTINE STRING PARAMETER, 728
SURFACE SUBROUTINE, 728
USER OUTPUT, 592
USER VARIABLE, 734
USER VARIABLE, 734

VALUES
in Definition For Function, 62
VARIABLE TYPE
in Initial Condition, 380
description of, 382
VECTOR
in Coordinate System, 73
VECTOR NODE
in Coordinate System, 73
VECTOR SCALE
in Lanczos Parameters, 102
in Power Method Parameters, 109
VELOCITY DAMPING COEFFICIENT
in Pressure, 428
description of, 430
VELOCITY DECAY
in Contact Definition – in PV_Dependent Model, 488
description of, 520
VISCOUS DAMPING, 474
VMIN
in Orthotropic Crush material model, 242
Void Elements, 298
VOLUME
in Point Mass Section, 320
VOLUME REPULSION OLD, 476
VOLUMETRIC SEARCH TOLERANCE
in MPC, 478

W FUNCTION
in Orthotropic Rate material model, 245
WALL THICKNESS
in Beam Section, 312
WEIBULL
in Initial Condition, 380
description of, 382
WEIBULL MEDIAN
in Initial Condition, 380
WEIBULL SCALE
in Initial Condition, 380
WEIBULL SCALING EXPONENT SCALE
in Initial Condition, 380
WEIBULL SCALING FIELD NAME
in Initial Condition, 380
WEIBULL SCALING FIELD TYPE
in Initial Condition, 380
WEIBULL SCALING REFERENCE VALUE
in Initial Condition, 380
WEIBULL SEED
in Initial Condition, 380
WEIBULL SHAPE
in Initial Condition, 380
WIDTH
in Beam Section, 312
WLF C1
in NLVE 3D Orthotropic material model, 262
WLF C2
in NLVE 3D Orthotropic material model, 262
WLF COEF C1
in Viscoelastic Swanson material model, 271
WLF COEF C2
in Viscoelastic Swanson material model, 271
WLF TREF
in Viscoelastic Swanson material model, 271
WT FUNCTION
in Orthotropic Rate material model, 245
WWBETA 1PSI
in NLVE 3D Orthotropic material model, 262
WWBETA 2PSI
in NLVE 3D Orthotropic material model, 262
WWBETA 3PSI
in NLVE 3D Orthotropic material model, 262
WWBETA 4PSI
in NLVE 3D Orthotropic material model, 262
WWBETA 5PSI
in NLVE 3D Orthotropic material model, 262
WWTAU 1PSI
in NLVE 3D Orthotropic material model, 262
WWTAU 2PSI
in NLVE 3D Orthotropic material model, 262
WWTAU 3PSI
in NLVE 3D Orthotropic material model, 262
WWTAU 4PSI
in NLVE 3D Orthotropic material model, 262
WWTAU 5PSI
in NLVE 3D Orthotropic material model, 262

X DISPLACEMENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 557
X DISPLACEMENT SCALE FACTOR
in Contact Definition – in User Search Box, 488, 557
description of, 557
X EXTENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 558

Y DISPLACEMENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 557
Y DISPLACEMENT SCALE FACTOR
in Contact Definition – in User Search Box, 488, 557
description of, 557
Y EXTENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 558
YIELD FUNCTION
in Wire Mesh material model, 240
YIELD STRESS
in Ductile Fracture material model, 214
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Johnson-Cook material model, 221
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
YIELD STRESS FUNCTION
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
YOUNGS MODULUS
in BCJ material model, 223
in Ductile Fracture material model, 214
in Elastic 3D Orthotropic material model, 237
in Elastic Fracture material model, 208
in Elastic material model, 202
in Elastic-Plastic material model, 210
in Elastic-Plastic Power-Law Hardening material model, 212
in Fiber Membrane material model, 251
in Fiber Shell material model, 254
in Foam Plasticity material model, 233
in Incompressible Solid material model, 256
in Johnson-Cook material model, 221
in Karagozian and Case concrete material model, 230
in Low Density Foam material model, 236
in Mooney-Rivlin material model, 259
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Neo Hookean material model, 206
in NLVE 3D Orthotropic material model, 262
in Orthotropic Crush material model, 242
in Orthotropic Rate material model, 245
in Power Law Creep material model, 225
in Soil and Crushable Foam material model, 227
in Stiff Elastic material model, 266
in Swanson material model, 268
in Thermoelastic material model, 204
in Viscoelastic Swanson material model, 271
in Wire Mesh material model, 240
YOUNGS MODULUS AA
in Elastic 3D Orthotropic material model, 237
YOUNGS MODULUS BB
in Elastic 3D Orthotropic material model, 237
YOUNGS MODULUS CC
in Elastic 3D Orthotropic material model, 237
YOUNGS MODULUS FUNCTION
in BCJ material model, 223
in Multilinear Elastic-Plastic Hardening Model material model, 216
in Multilinear Elastic-Plastic Hardening Model with Failure material model, 218
in Thermoelastic material model, 204

Z DISPLACEMENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 557
Z DISPLACEMENT SCALE FACTOR
in Contact Definition – in User Search Box, 488, 557
description of, 557
Z EXTENT FUNCTION
in Contact Definition – in User Search Box, 488, 557
description of, 558
ZOLTAN DEBUG LEVEL
in Zoltan Parameters, 365
as default, 364
ZOLTAN PARAMETERS
as command block, 365
with defaults, 364
as command line in Rebalance, 362
description of, 363

Distribution

1	0899	RIM-Reports Management, 9532 (1 electronic)