SANDIA REPORT

SAND2016-3046 O
Unlimited Release
Printed April 4, 2016

Sierra Structural Dynamics–User's Notes

Sierra Structural Dynamics Development Team

Latest Software Release:
4.40.1-Release 2016-04-04

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 U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Unlimited Release

Sandia National Laboratories

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

SAND2016-3046 O
Unlimited Release
Printed April 4, 2016

Sierra Structural Dynamics–User's Notes

Sierra/SD Development Team

Abstract

Sierra/SD provides a massively parallel implementation of structural dynamics finite element analysis, required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. This document provides a users guide to the input for Sierra/SD. Details of input specifications for the different solution types, output options, element types and parameters are included. The appendices contain detailed examples, and instructions for running the software on parallel platforms.

Contents

1 Introduction 3

2 Sections of a **Sierra/SD** Input File 6

2.1 SOLUTION 6

2.1.1 Multicase 6

2.1.2 A Note On Time Stepping In Multicase Solutions 9

2.1.3 Checkout 9

2.1.4 CJdamp 9

2.1.5 Craig-Bampton Reduction 10

2.1.6 Dynamic Design Analysis Method (DDAM) 13

2.1.7 Directfrf 16

2.1.8 Directfrf-LoadId 19

2.1.9 DirectFrf-MaterialId 21

2.1.10 Dump 22

2.1.11 Eigen 22

2.1.12 Eigen-MaterialId 25

2.1.13 Direct Eigen Solution 25

2.1.14 AEigen 27

2.1.15 Blk_eigen 28

2.1.16 Eigenk 29

2.1.17 Largest_Ev 29

2.1.18 Buckling 30

2.1.19 Modal Participation Factor 31

2.1.20 Modalfrf 34
2.1.21 Modalranvib 37
2.1.22 Modalshock 41
2.1.23 Modaltransient 42
2.1.24 QModaltransient 44
2.1.25 QEVP 45
2.1.26 QModalfrf 53
2.1.27 NLStatics 53
2.1.28 NLTransient 55
2.1.29 Receive_Sierra_Data 56
2.1.30 Statics 58
2.1.31 Subdomain_Eigen 58
2.1.32 Tangent 59
2.1.33 Transhock 59
2.1.34 Transient 60
2.1.35 Transient-LoadId 64
2.1.36 TSR_Preload 65
2.1.37 Explicit Solver 67
2.1.38 Geometric Rigid Body Modes 69
2.1.39 Waterline of Rigid Body 71
2.1.40 Gap Removal 75
2.2 Solution Options 76
2.2.1 Restart – option 77
2.2.2 Solver 80
2.2.3 Lumped – option 83
2.2.4 Constraintmethod – option 83

2.2.5 Scattering – option 83
2.2.6 no_symmetrize_struc_acous – option 84
2.2.7 transfer – option 84
2.3 PARAMETERS 85
2.4 FETI 95
2.4.1 Corner Algorithms 97
2.4.2 Solves within Solves 97
2.4.3 Levels of Diagnostic Output 99
2.5 CLOP 100
2.6 GDSW 102
2.7 Solver_Options 108
2.8 ECHO 108
2.8.1 Mass Properties 109
2.8.2 Mpc 111
2.8.3 ModalVars 111
2.8.4 Subdomains 112
2.8.5 Memusage 112
2.9 OUTPUTS 113
2.9.1 Maa 113
2.9.2 Kaa 114
2.9.3 Faa 114
2.9.4 ElemEigChecks 114
2.9.5 Elemqualchecks 114
2.9.6 Displacement 115
2.9.7 Velocity 116
2.9.8 Acceleration 116

2.9.9 Strain 116
2.9.10 Stress 116
2.9.11 VonMises 117
2.9.12 Stress = GP 118
2.9.13 VRMS 118
2.9.14 Energy 118
2.9.15 GEnergies 118
2.9.16 Mesh_Error 120
2.9.17 Harwellboeing 120
2.9.18 Mfile 121
2.9.19 Force 121
2.9.20 rhs 121
2.9.21 EForce 121
2.9.22 Residuals 123
2.9.23 Resid_only 123
2.9.24 TIndex 123
2.9.25 EOrient 124
2.9.26 Pressure 125
2.9.27 NPressure 125
2.9.28 APressure 126
2.9.29 APartVel 126
2.9.30 Slave_Constraint_Info 126
2.9.31 Statistics 127
2.9.32 KDiag 127
2.9.33 ADiag 129
2.9.34 Warninglevel 129

2.9.35 Precision 129
2.9.36 ddamout 130
2.10 HISTORY 130
2.11 FREQUENCY 134
2.12 FILE 135
2.12.1 geometry_file 136
2.12.2 sierra_input_file 137
2.12.3 Additional Comments About Output 137
2.13 Linesample 137
2.14 BOUNDARY 138
2.14.1 Prescribed Displacements and Pressures 140
2.14.2 Prescribed Accelerations 141
2.14.3 Node_List_File 142
2.14.4 Nonreflecting Boundaries 142
2.14.5 Impedance Boundary Conditions 143
2.14.6 Slosh Boundary Conditions 144
2.14.7 Infinite Elements 144
2.15 LOADS 147
2.15.1 Scale Factors for the Load 148
2.15.2 Sideset Loading 149
2.15.3 Spatial Variation 151
2.15.4 Required Section 151
2.15.5 Follower Stiffness 151
2.15.6 Acoustic Loads 152
2.15.7 Thermal Loads 155
2.15.8 Energy Deposition Input and Loads 158

2.15.9 Consistent Loads 159
2.15.10 Pressure_Z 159
2.15.11 Static Loads 159
2.15.12 Time Varying Loads 160
2.15.13 Random Pressure Loads 160
2.15.14 Frequency Dependent Loads 164
2.15.15 Rotational Frames 164
2.15.16 Rigid Body Filter for Input 168
2.16 Load 169
2.17 INITIAL-CONDITIONS 170
2.18 RanLoads 171
2.19 Contact Data 173
2.20 Tied Surfaces 173
2.20.1 Mortar Methods 174
2.20.2 Node to Face 175
2.21 Contact Normals 176
2.22 RigidSet 178
2.23 RrodSet 181
2.24 Tied-Joint 181
2.24.1 Input Specification 182
2.24.2 Output Specifications 185
2.25 BLOCK 187
2.25.1 Block Parameters 187
2.25.2 General Block Parameters 189
2.26 Macroblock 191
2.27 MATERIAL 193

2.27.1 Isotropic Material 194
2.27.2 Anisotropic Material 195
2.27.3 Orthotropic Material 195
2.27.4 Stochastic Material 196
2.27.5 Linear Viscoelastic Material 197
2.27.6 Acoustic Material 200
2.27.7 Temperature-Dependent Material Properties 200
2.27.8 Density 201
2.27.9 Specific Heat 201
2.27.10 CJetaFunction 203
2.28 COORDINATE 203
2.29 FUNCTION 205
2.29.1 Linear Functions 207
2.29.2 Functions using Tables 208
2.29.3 Polynomial Functions 209
2.29.4 LogLog Functions 210
2.29.5 Random Functions 210
2.29.6 SamplingRandom Functions 211
2.29.7 RandomLib Functions 212
2.29.8 SpatialBC Functions 214
2.29.9 ReadNodal Functions 215
2.29.10 ReadNodalSet Functions 216
2.29.11 ReadSurface Functions 216
2.29.12 User Defined Functions 217
2.29.13 Plane Wave 219
2.29.14 Planar Step Wave 220

2.29.15 Spherically Spreading Wave 220
2.29.16 Shock Wave 222
2.29.17 FSI 224
2.30 MATRIX-FUNCTION 224
2.30.1 Alternate Table Interface 226
2.31 Table 228
2.32 CBModel 229
2.32.1 Sensitivity Analysis for Craig-Bampton models 233
2.33 ModalFilter 236
2.34 Integer List 238
2.35 SENSITIVITY 238
2.35.1 Attune 239
2.35.2 Sensitivity Output 240
2.35.3 Sensitivity Limitations 242
2.36 Element Level Interface for UQ 242
2.37 DAMPING 244
2.37.1 Nonlinear transient solutions with damping 246
2.37.2 Nonlinear Distributed Damping 246
3 Elements 249
3.1 Hex8 249
3.2 Hex20 250
3.3 Wedge6 250
3.4 Wedge15 250
3.5 Tet4 251
3.6 Tet10 251
3.7 QuadT 251

3.8 QuadM 252
3.9 Quad8T 254
3.10 Nquad/Ntria 255
3.11 TriaShell 256
3.12 Layered Shell 257
3.13 Tria3 260
3.14 Tria6 261
3.15 Offset Shells 261
3.16 HexShell 262
3.17 Beam2 266
3.18 Nbeam 270
3.19 OBeam 274
3.20 Truss 274
3.21 Ftruss 274
3.22 ConMass 275
3.23 Spring 277
3.23.1 Spring Parameter Values 278
3.24 RSpring 278
3.25 Spring3 - nonlinear cubic spring 279
3.26 Dashpot 280
3.27 SpringDashpot 281
3.28 Hys 281
3.29 Shys 283
3.30 Iwan 285
3.31 Joint2G 285
3.31.1 Specification 286

3.31.2 Constitutive Behavior . . . 287
3.32 Gap . . . 293
3.33 Gap2D . . . 297
3.34 GasDmp . . . 297
3.35 Nmount . . . 299
3.36 MPC . . . 299
3.37 RROD . . . 302
3.38 RBar . . . 303
3.39 RBE2 . . . 304
3.40 RBE3 . . . 304
3.41 Superelement . . . 306
3.42 Interface Elements . . . 312
3.43 Dead . . . 312
3.44 Offset Elements and Lumped Mass . . . 313
4 Stress/Strain Recovery . . . 314
4.1 Stress/Strain Truth Table . . . 314
4.2 Solid Element Stress/Strain . . . 314
4.3 Shell Element Stress/Strain . . . 314
4.4 Line Element Stress/Strain . . . 316
5 Inverse Problems . . . 317
5.1 Optimization . . . 319
5.2 Inverse-Problem . . . 323
5.2.1 Regularization Parameters . . . 323
5.2.2 Inverse Data Files . . . 323
5.3 BLOCK section for Material Identification . . . 327
5.4 MATERIAL section for Material Identification . . . 328

5.5 LOADS section for Load Identification 331
5.6 Limitations for Inverse Load Problems 332
5.7 ROL Output for Inverse Problems 333
6 Troubleshooting 335
6.1 Stand-Alone Tools 335
6.1.1 Grope 335
6.1.2 Cubit 335
6.2 Using Yada to identify disconnected regions 336
6.3 Using Sierra/SD To Troubleshoot 336
6.3.1 Modal Analysis 337
6.3.2 Evaluating Memory Use 338
6.3.3 Debugging RBMs with the Node_List_File 338
6.3.4 Identifying Problematic Subdomains 339
6.3.5 Problematic Elements and Connectivity 339
6.4 Troubleshooting FETI Issues 341
6.4.1 Introduction 341
6.4.2 Standard FETI Block 341
6.4.3 Memory 341
6.4.4 Local Rigid Body Modes 343
6.4.5 Global Rigid Body Modes 344
7 Acknowledgments 346

References **347**

Appendix

1 Sierra/SD Example Input Files 351
1.1 An Eigenanalysis Input File 351

1.2 An Anisotropic Material Input File 352
1.3 A Multi-material Input File 354
1.4 A Modaltransient Input File 358
1.5 A Modalfrf Input File 360
1.6 A Directfrf Input File 362
1.7 A Statics Input File 364
2 Running Sierra/SD on serial UNIX platforms 365
3 Running Sierra/SD in Parallel 367
3.1 Number of Processors Needed 367
3.2 Use "yada" to load balance the model 368
3.3 Running Yada on serial UNIX platforms 369
3.4 Parallel Machine Work Space 369
3.5 Using Nem_spread 370
3.6 Sierra/SD FILE Section 370
3.7 Running Sierra/SD 371
3.8 Joining Result Files 371
4 CF FETI 373
4.1 Features of CF solver 373
4.2 Limitations of the Solver 373
5 GDSW Solver Parameters for Older Version 377
Index 381

List of Figures

1 DDAM Example Input . . . 17

2 Padé Expansion Input Example . . . 19

3 Direct frf load id input . . . 20

4 Eigen Material Id Example . . . 26

5 Aeigen example . . . 28

6 Blk_eigen Example . . . 29

7 Modal Participation Factor (MPF) Example . . . 33

8 SA_Eigen Example . . . 50

9 Transhock Example Input . . . 60

10 Transient/Transfer Example for Gemini type Coupling. . . . 64

11 Transient Load Identification Example . . . 65

12 Explicit Time Step Control Example . . . 69

13 Waterline Example . . . 72

14 Waterline Coordinate Definition . . . 73

15 Net Force vs depth for a Rigid Body . . . 74

16 Notes on Eigen Restart . . . 80

17 Example MFile Format Results . . . 94

18 Multiple Solver Options Example . . . 108

19 Example *KDIAG* output. . . . 128

20 Example Boundary Section . . . 139

21 Coordinate Frame Projection for Tractions . . . 151

22 Depth Dependent Pressure Load Example . . . 159

23 RandomPressure Loading Approximations 161
24 Statics LOADS entry for Rotation 165
25 Example using Tangent Update 167
26 Example of using qevp for Tangent Update 168
27 Search Tolerance definition 176
28 Normal Definitions on Faceted Geometry 177
29 Smoothing Parameters for Surface Normals 177
30 Shell Normal in Contact or Tied Interactions 179
31 RigidSet/TiedJoint Centernode Connection 180
32 RrodSet Constraints 181
33 Tied Joint Surface Normal Definition 183
34 Tied Joint Geometry 185
35 Tied Joint Example 186
36 Example Block input 188
37 Coordinate System Definition Vectors 204
38 Linear function #3. "illegal_fun" 207
39 Linear function #5. "extrap_fun" 208
40 Example *Gaussian* output. 211
41 Example **RandomLib** Function Specification 213
42 RandomLib Temporal Interpolation 214
43 Example **ReadNodal** Function Specification 215
44 Example **ReadNodalSet** Function Specification 216
45 Example **ReadSurface** Function Specification 217
46 Example **PlaneWave** Function Specification 220
47 Spherical Wave Geometry 221
48 Spherical Wave Example 222

49 Example **Shock Wave** Function Specification 223
50 Fluid-Structure Interaction (FSI) Infrastructure 225
51 Example Input for a Matrix-Function using Tables 227
52 Craig-Bampton Reduction 234
53 Example ModalFilter Input 236
54 Eigen Sensitivity Example Data 241
55 UQ element interface 243
56 QuadT Element 252
57 Quad8T Element 254
58 Function for nquad_eps_max 256
59 Shell Rotation Process 258
60 Stacking arrangement for a multilayer shell element. 259
61 Tria6 Element 261
62 Example HexShell Input 264
63 HexShell Autolayer Example 265
64 Beam Orientation and Local Coordinate System. 268
65 Beam Offset and Local Coordinate System. 269
66 NBeam Orientation, Offset and Local Coordinate System 272
67 **Hys** element parameters 284
68 **Iwan** Constitutive Model 289
69 Hysteresis Microslip Variation with β 290
70 Hysteresis Macroslip Variation with β 291
71 Eplas Model 292
72 Gap element Force-Deflection Curve 295
73 Mass bouncing off a Gap 296
74 Gap2D force diagram 298

75 **Sierra/SD** Mount Interface 299
76 Nmount Orientation 300
77 Tria3 Stress Recovery 317
78 Sample "directfrf-materialid" Problem Input 318
79 Optimization Section Example for source inversion 319
80 Optimization Section Example 322
81 Inverse-Problem 324
82 Sample Data Truth Table Input for Acoustic Problem 324
83 Example data truth table for Structural Acoustics 325
84 Example Data Truth Table for Structures 325
85 Sample Real Data File Input 326
86 Block Section Example for Material Inversion 328
87 Block Section Example for Material Inversion 329
88 Material Section Example for Material Inversion 330
89 Loads Section Example for Acoustic Force Inversion 332
90 Loads Section Example for Structural Force Inversion 333
91 Example of *ROL_Messages.txt* file for Inverse Problem Solution 334
92 Problem Decomposition 337
93 Single *Spring* element 340
94 Truss Decomposition Issues 340

List of Tables

1 Comment String Options . . . 3

2 **Sierra/SD** Solution Types . . . 7

3 Multicase Parameters . . . 8

4 DirectFRF Parameters for Padé Expansion . . . 18

5 Fluid Loading Parameters . . . 23

6 AEigen optional parameters . . . 27

7 AEigen Verbosity Table . . . 28

8 MPF Parameters . . . 32

9 MPF Summary data . . . 33

10 ModalRanVib Output to Exodus File . . . 40

11 Comparison of Quadratic EigenProblem Methods . . . 47

12 Parameters for QEVP Anasazi Solutions . . . 48

13 Ceigen Tests . . . 49

14 SA_Eigen Parameters . . . 51

15 Verification Summary for SA_Eigen . . . 52

16 Projection_Eigen Parameters . . . 53

17 Receive_Sierra_Data Parameters . . . 56

18 Explicit Transient Solution Parameters . . . 68

19 Waterline Parameters . . . 71

20 **Sierra/SD** Solution Options . . . 76

21 Supported restart capabilities for transient integrators in Sierra/SD. . . . 79

22 Restart File Format and Names . . . 81

23 Available keywords in the Parameters section 86
24 Developer keywords in the Parameters section 86
25 Some useful combinations of units. 87
26 Beam Attribute Ordering 87
27 Eigenvector Normalization Methods 93
28 FETI Section Options 96
29 Corner Options 98
30 Linear Solver Options 99
31 Prt_Debug Options 100
32 CLOP Section Options 101
33 GDSW Section Options (Basic) 102
34 GDSW Section Options (Advanced) 103
35 GDSW Section Options (Supplemental Output) 104
36 GDSW Section Options (Helmholtz) 107
37 ECHO Section Options 110
38 Hex20 Gauss Point Locations 119
39 Data Files Written Using the Mfile Option 122
40 TIndex parameters 124
41 Element Orientation Outputs 125
42 Element Orientation Interpretation 125
43 Supported Statistical Data types 127
44 Selected Dynamic Matrix Definitions 129
45 Warning Diagnostic Options 129
46 Output Exodus Precision Options 130
47 OUTPUT Section Options 131
48 Variables that are output from ddam analysis 132

49 Frequency Value Specification Methods 135
50 Boundary Enforcement Keywords 140
51 Limitations for Prescribed Boundary Conditions 142
52 Available parameters for the infinite element section 145
53 Load Specification Keywords 150
54 Random Pressure Inputs 162
55 Rotating Frame Parameters 165
56 Tied Surface Parameters 177
57 RigidSet Parameters 178
58 Tied Joint Parameters 183
59 Tied Joint, “Normal” and “Side” dependencies 185
60 General Block Parameters 189
61 Non-Structural Mass Units 191
62 Element Attributes 192
63 Default Parameters for Viscoelastic Materials 198
64 Material Stiffness Parameters 202
65 Coordinate Names for history files 206
66 Random function parameters 211
67 SamplingRandom function parameters 212
68 RandomLib function parameters 213
69 ReadNodal function parameters 215
70 Predefined RTC variables 219
71 Planar Step Wave Parameters 221
72 Spherical Wave Parameters 221
73 Shock Wave Parameters 223
74 Free Surface Flag Options 223

75 TABLE Section Options 228
76 CBModel Parameters 230
77 Data output for Craig-Bampton Reduction 232
78 Modal Filter Keywords 237
79 Sensitivity Analysis Keywords 238
80 Sensitivity Analysis Solution Type Availability 242
81 DAMPING Section Options 244
82 QuadM attributes 253
83 TriaShell attributes 257
84 HexShell Verification Summary 266
85 Attributes for Beam2 270
86 Attributes and Parameters for Nbeam 273
87 Ftruss Attributes and Parameters 276
88 SpringDashpot Parameters 282
89 Older Iwan 4-parameter model 288
90 Revised Iwan 4-parameter model 288
91 Rbar Exodus Attributes 303
92 Element Stress Truth Table 315
93 Optimization Section Parameters 322
94 Optimization Section Parameters for Line Search 322
95 Optimization Section Parameters for Trust Region 322
96 Inverse-Problem parameters 323
97 Block Section Parameters for Material Inversion 328
98 Block Section Parameters for Material Inversion 329
99 Material Section Parameters for Material Inversion 330
100 Table of Supported Material Parameters for Inverse Methods 331

101 Loads Section Parameters for Force Inversion 332

1 Determining Number Of Processors Needed 368

2 CF FETI Parameter Modifications 374

3 Solver options and defaults specific to older GDSW solver 378

4 Solver options and defaults specific to older GDSW Solver 379

This page intentionally left blank.

Sierra Structural Dynamics

Sierra/SD provides a massively parallel implementation of structural dynamics finite element analysis. This capability is required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. General capabilities for modal, statics and transient dynamics are provided.

This document describes the input for the **Sierra/SD** program. Examples of input specifications are scattered throughout the document. Appendix A provides several full input files. Appendix B provides instructions on invoking **Sierra/SD** on a serial UNIX platform.

The **Sierra/SD** software evolved from the "Salinas" package. The name for "Salinas" is taken from a series of ancient Tewa Indian pueblos to the east of Albuquerque, New Mexico. These pueblos have been a source of culture and of salt for centuries. They were among the first settlements for Spanish explorers in the region.

This page intentionally left blank.

1 Introduction – Input File

The input file contains all the directives necessary for operation of the program. These include information on the type of solution, the name of the **Exodus** file containing the finite element data, details of the material and properties within the element blocks, which boundary conditions to apply, etc. Details of each of these sections are covered below.

Typically, the input file has an extension of "`.inp`", although any extension is permitted. If the "`.inp`" extension is used, **Sierra/SD** may be invoked on the input without specifying the extension.

The input file is logically separated into sections. Each section begins with a keyword (**Solution**, **BLOCK**, etc), and ends with the reserved word **end**. Words within a section are separated with "white space" consisting of tabs, spaces, and line feeds. Comments are permitted anywhere within the file, and follow the C++ convention, i.e. a comment begins with the two characters "//" and ends with the end of the line.[1]

Comments

Several options are available for a comment specifier. These are listed in Table 1. Whatever the string used to specify a comment, it behaves in the same way, namely all characters following the comment string are skipped. The comment string is specified by starting a the file with one of these special strings, followed by white space. Thus if the first line in a file is,

```
# this is my input file for Sierra/SD
```

then the "#" character will be used as the comment character for that entire file. If this file includes another file using the "`include`" directive (see below), that file will begin processing using the previous comment string, unless the included file starts with one of the special comment strings.

Table 1: Comment String Options

String	Descriptor
//	C++ style comments (default)
#	Hash character. Used in scripts and Sierra input
;	Semicolon.
%	percent. Used in Matlab, LaTex, etc.

[1] To be safe, define comments as "//" followed by a space.

Skipping Sections

Occasionally an entire section may need to be commented out. This may be done using "//" on each line of the section. An other way to comment out an entire section is to begin it with double "$$" characters, followed by a space. In the following, block 1 is commented out, and block 4 is active.

```
$$ BLOCK 1       // this section skipped
   material 1
END

BLOCK 4          // this section valid
   material 1
END
```

Except for data within quotes, the input file is case insensitive. The software converts everything to lower case unless it is enclosed in quotes. Either the single quote `'` or the double quote `"` may be used. Quotes may be nested by using both single and double quotes, as in either `'a string with "embedded" quotes'` or `"a string with 'embedded' quotes"`.

Preprocessing with Aprepro

Sierra employs a powerful preprocessor, "`aprepro`", which can be run either stand alone, or as part of the analysis. Aprepro can be used for a variety of purposes.

1. To define variables on the command line. This is especially useful for automated runs such as optimization and uncertainty quantification.

2. To simplify input by allowing algebraic expressions, e.g. `Y={ 4 * 3}.`

3. To automatically include text of other files.

4. To manage various systems of units, e.g. `Y={ 10 * psi }.`

For details on aprepro in general, and for stand alone documentation, please refer to the seacas documentation.[1] All the rules for command line substitution apply to the built-in capability. Definition of command line variables during analysis requires specification of command line arguments, **--aprepro** and **--define**, as used in the example.

```
sierra salinas --aprepro --define "Eval=10E6 NuVal=0.30" -i example.inp
```

In this example, the text "Eval" in the input file, *example.inp* will be replaced with "10E6". Likewise "NuVal" will be replaced by 0.30.

Including Files

The input parser supports nested includes[2]. This is done using the **include** command. This is the only command the parser recognizes. Files may be included to any depth. As an example,

```
include english_materials
```

The **include** may occur anywhere on the line (though for readability and consistency we recommend that it be the start of the line). The file name must immediately follow and need NOT be enclosed in quotes. Case sensitivity will be preserved. No white space is allowed in the file names.

Files may also be included using Aprepro processing (see the previous section). The syntax is slightly different, but more consistent with the parser used in other modules.

```
{include("english_materials")}
```

Input Summary

Summarizing, a minimum of two files are needed to run **Sierra/SD** , namely, a text input file, e.g. *example.inp*, and an **Exodus** input file,[2] e.g. *example.exo*, which contains the finite element model. The finite element model is specified in *example.inp* as the geometry file (see section 2.12).

Each of the **Sierra/SD** input sections is described in the following section.

[2] Prior to release 2.5 the command for including a file was "#include". That syntax will continue to be supported, but in release 2.5 we introduced user specified comment delimiters including the # character. If "#" is used to start a comment, it becomes impossible to include a file using the old syntax.

2 Sections of a Sierra/SD Input File

2.1 SOLUTION

The **solution** section determines which solution method, and options are to be applied to the model. The available solution types are shown in Table 2. Relevant options are shown in Table 20, and are described in section 2.2.

2.1.1 Multicase

All of the solution methods of table 2 may be a part of a multicase solution. This allows the user to specify multiple steps in a solution procedure. For example, there can be a static preload, a computation of the updated tangent stiffness matrix, and a linearized eigen analysis. The syntax for multicase solutions is similar to that for single cases, but each solution step is delineated by the "case" keyword. In addition, any of the modal solutions must be preceded by an eigen analysis and eigen keywords are no longer recognized as part of the solution.

In a multicase solution, the system matrices (mass, stiffness and damping) will typically be computed only once. Matrix updates between solutions may be specified by selecting the **tangent** keyword (see section 2.1.32).

2.1.1.1 Multicase Parameters. Many of the solution parameters are specific to a particular solution type. For example, time step parameters are meaningless in a modal solution. However, some options apply more generally. These parameters, listed in Table 3, may be specified either above the case control sections, or within the section. The specification above the case control section is the default value. Specifications within the case sub-blocks apply only to that sub-block. In the example below, the `restart` options are thus "none" for most sub cases, but "read" for the eigen analysis and "auto" for the linear transient.

2.1.1.2 Multicase Example. In the example which follows, a nonlinear statics computation is followed by a tangent stiffness matrix update. The updated matrix is then used in an eigen analysis. Two sets of **Exodus** output files will be written. Output from the statics calculation will be in files of the form *'example-nls.exo'*. Eigen results will be in the form *'example-eig.exo'*. The **tangent** solution normally produces no output in the **Exodus** format.

```
Solution
   restart=none
   title='example multicase'
```

Table 2: **Sierra/SD** Solution Types

Solution Type	Description	Parameters
buckling	buckling eigensolution	nmodes, shift
cbr	Craig-Bampton reduction	nmodes, shift
ceigen	complex eigen	
checkout	skip large matrix and solves	
cjdamp	modal damping contributions	
directfrf	direct frequency response	
directfrf-loadid	inverse frf load id	
directfrf-materialid	inverse frf material id	
ddam	dynamic design analysis method	U.S. navy methods imaginary_data_file
dump	form matrices only	
eigen	real eigensolution	nmodes, shift, untilfreq
eigen-materialid	inverse eigen material id	nmodes, shift, untilfreq
eigenk	real eigensolution of K *(seldom useful)*	nmodes
gap_removal	gap removal debugging	
largest_ev	largest eigenvalue of K,M	*seldom used directly*
modalfrf	frequency response using modal displacement or modal acceleration	nmodes, usemodalaccel, nrbms, complex
modalranvib	random vibration using modal superposition check correlation matrix	eigen parameters noSVD CheckSMatrix
modalshock	shock response spectra using modal approximate implicit transient analysis (unimplemented)	nmodes, time_step, nsteps, nskip, flush srs_damp
modaltransient	transient analysis using modal superposition	nmodes, time_step, nsteps, nskip, start_time, flush
NLstatics	nonlinear statics	max_newton_iterations,tolerance num_newton_load_steps, update_tangent
NLtransient	implicit nonlinear transient analysis	time_step, nsteps, nskip, start_time, rho, flush, max_newton_iterations,tolerance
old_transient	implicit transient analysis (acceleration based)	time_step, nsteps, nskip, rho, flush *(can include sensitivity analysis)*
Receive_Sierra_Data	coupling to Sierra	
statics	static stress	
subdomain_eigen	subdomain eigenanalysis *(ONLY for debug)*	nmodes
tangent	compute tangent matrices	*(multicase only)*
transhock	shock response spectra using direct implicit transient analysis	time_step, nsteps, nskip, flush srs_damp
transient	implicit transient analysis	time_step, nsteps, nskip, start_time, rho, flush
transient-materialid	transient material id	time_step, nsteps, nskip, start_time, rho, flush
tsr_preload	thermal structural response	file *(multicase only)*

Table 3: Multicase Parameters

These parameters may be specified as defaults above the case specifications, or they may be specified for each subcase to which they apply.

Parameter	Description	Options
restart	Restart options	see section 2.2.1
solver	selection of solver	see section 2.2.2

```
  case 'nls'
    nlstatics
    load=10
  case 'tangent'
    tangent
  case 'eig'
    eigen
    restart=read
  case 'trans1'
    transient
      restart=auto
      time_step 1e-8 1e-6
      nsteps    100  4000
      flush 50
      rho=0.9
      load=20
  case 'trans2'
    transient
      restart=auto
      time_step 1e-4
      nsteps    200
      flush 10
      load=20
END
```

The **case** keyword must always be followed by a label. The label is used in the output file name. The **case** keyword is also used to divide parameters of each solution type.

The **load** keyword is used within a solution step to indicate which loads to apply during a solution. In the example above, load '10' will be applied during the nonlinear statics calculation. During a multicase solution the **loads** section (found elsewhere in the file) will be ignored. See paragraph 2.15 for information on the **loads** section or paragraph 2.16 for information on the **load** section of the input file.

2.1.2 A Note On Time Stepping In Multicase Solutions

In the multicase example provided above, compare cases 'trans1' and 'trans2'. It is important to note that case 'trans1' will step through 100 steps of time at a step size of 1e-8, then step through 4000 steps at a step size of 1e-6. Assuming the calculation starts at time=0, the final time value of case 'trans1' will be 1e-8*100 + 1e-6*4000 = 0.004001. Case 'trans2' will start at 0.00400099 and run an additional 200 time steps at a step size of 1e-4. This will end at a time value of 0.024001. (NOTE: This was not the default behavior for **Sierra/SD** versions 1.2.1 or earlier).

2.1.3 Checkout

The **checkout** solution method tests out various parts of the code without forming the system matrices or solving the system of equations. This solution method may be used to check input files for consistency and completeness on a serial platform before allocating expensive resources for a full solution.

2.1.4 CJdamp

The **CJdamp** solution provides a method of computation of the equivalent modal damping terms introduced from material damping in lightly damped visco elastic materials. It is based on a development by Conor Johnson *et al.*[3] It is an approximate method which assumes that the mode shapes and frequencies are not modified by the damping. The modal damping is simply related to the fraction of energy in block.

The **CJdamp** method is effectively a post processing step following an eigen analysis. For each of the modes in the eigen analysis, a strain energy is computed on an element basis. These are summed at the block level.

$$SE_j^i = \sum_{elem}^{\text{in block j}} \phi_i^T K^{elem} \phi_i \tag{1}$$

The total strain energy TSE^i is just the sum of the strain energy contributions in mode i from all blocks. We define the block strain energy ratio for mode i as,

$$R_j^i = SE_j^i / TSE^i \tag{2}$$

The **CJdamp** contribution for the modal damping of mode i, is given by,

$$\zeta_i = \frac{1}{2} \sum_j R_j^i \eta_j(f_i) \tag{3}$$

Where $\eta_j(f_i)$ is the **CJetaFunction** contribution from block j evaluated at the natural frequency of mode i (see section 2.27.10).

Note that cases following the CJdamp solution will include this damping as part of their damping calculation.

Example,

```
SOLUTION
   case eig
       eigen nmodes=30
   case cjd
       cjdamp
   case frf
       modalfrf
END
```

2.1.5 Craig-Bampton Reduction

It can be advantageous to reduce a model to it's interface degrees of freedom. This is very important in satellite work, where the model of the satellite may be much larger than the model of the remainder of the missile. Reduction of the satellite model to a linearized, Craig-Bampton model makes it possible to share the dynamic properties of the model without requiring details of the interior. There are many types of component mode synthesis techniques (or **CMS**), of which the Craig-Bampton approach is one of the more popular. In this approach the model is reduced to a combination of fixed interface and constraint modes. The fixed interface modes are eigenvectors of the system with all interface degrees of freedom clamped. A constraint mode is the deformation that results if one interface degree of freedom receives a unit displacement, and all other interface degrees of freedom are zero. There is one constraint mode per interface degree of freedom.

The **CBR** solution reduces an entire structural model to its reduced system and transfer matrices. Parameters are listed in the table below, and correspond to the parameters required for an eigen analysis (section 2.1.11). In addition, a **CBModel** section must be defined elsewhere (see section 2.32). Any boundary conditions specified are applied before reducing the model.

We note that sensitivity analysis can be performed in **CBR** analysis, though the process is somewhat different than other types of sensitivity analysis. Section 2.32 contains more information about sensitivity analysis in Craig-Bampton models.

Parameters are listed below.

Parameter	Type	Argument
nmodes	*integer*	number of fixed interface modes
shift	*Real*	negative shift
correction	*string*	correction method for rigid body modes
RbmDof	*string*	zem selector
ModalFilter	*string*	optional modal filter

The method will write system matrices and general information. Each of the parameters is described below.

nmodes: The CB model is composed of fixed interface modes and constraint modes. The number of constraint modes is determined by the interface. The number of fixed interface modes is set by this parameter. The fixed interface modes are eigen modes of the interior of the structure, and provide a basis for internal deformation. Any number of these modes may be specified. Typically frequencies up to about twice the system frequency are required for good accuracy.

correction: As shown in the theory manual, the null space of the stiffness matrix is determined by the sum of two large terms: $\kappa_{cc} = K_{cc} + K_{cv}\psi$. With parallel iterative solvers, it may be difficult to determine this quantity as accurately as desired. In particular, it is possible for errors in the solver to render the reduced matrices negative definite, which can cause instability in subsequent transient analysis. It is strongly recommended that low solution tolerances be used in developing CB models. In addition, the matrix may be post processed to correct these errors. The post processing options are as follows:

none no correction will be applied.

values (default) no corrections will be made to the eigen vector space, but the negative eigenvalues will be adjusted to zero.

vectors Zero energy eigen vectors are determined geometrically (which is typically very accurate), and these are used to correct both the eigenvalues and the eigenvectors. This is more involved than correcting the eigenvalues alone, but it is not a significant computational cost, and can greatly improve the usefulness of the resulting model.

If *correction=vectors* is selected, one may also optionally determine which zero energy modes are required. This is done with the *RbmDof* parameter. The parameter is followed by a string indicating which dofs are active on the interface. The string contains the numbers 1 through 6, where 1 represents translation in the x coordinate direction. These specifications apply in the basic coordinate frame.

shift: Parallel solvers require a large negative shift. This is required to ensure that all subdomains are non singular.

ModalFilter: The optional **ModalFilter** keyword provides a means of reducing the modes retained for output and for subsequent analysis. For more details, see section 2.33. You can also put the modal filter into a separate case, called **preddam**, 2.1.6.

An example follows.

```
SOLUTION
  case cbr
    cbr
      nmodes=20 shift=-4e6
      correction=vectors
      RbmDof='123'
END
```

Inertia Tensor for Craig-Bampton Reduction. The capability to output a reduced inertia matrix, **I** from a Craig-Bampton Reduction (CBR) analysis is also available. The input file syntax is described in the **CBModel** section, 2.32. **I** is defined by

$$\mathbf{I} = \mathbf{\Phi}^T \mathbf{R},$$

where $\mathbf{\Phi}$ is the matrix of mode shapes used for the CBR analysis and consists of both fixed-interface modes and constraint modes. The number of rows in **I** is the number of (a-set) degrees of freedom in the model and the number of columns is the number of CBR modes. **R** has the same number of rows and one column per rigid body mode. For example, for just the three translational rigid-body modes and assuming just three degrees-of-freedom per finite element node,

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \end{bmatrix}.$$

Note the following limitations for the CBR method.

- *In serial, no MPCs may share nodes with interface nodes. Otherwise the MPCs may eliminate the dofs that should be retained. For parallel domain decomposition solvers (such as FETI), this restriction is relaxed.*
- *The entire reduced order model and associated transfer matrix must fit into memory. On a parallel machine, this memory is required on every processor. The model dimension is the sum of the number of constraint and fixed interface modes.*
- *The reduction process involves static solutions of the system with all interface degrees of freedom clamped. Such a solve may be singular if the interface dofs do not properly constrain the system. In such cases the solution is not reliable. It is a good idea to verify the reduced order eigen solution against the full system. One may also compare the retained mass.*

2.1.6 Dynamic Design Analysis Method (DDAM)

The U.S. Navy Dynamic Design Analysis Method (DDAM) is an established procedure employed in the design for ship equipment and foundations for shock loading requirements. The details of the formulation, specific procedures for application, acceptance criteria, etc., are documented in NAVSEA Report 250-423-30 and NAVSEA 0908-LP-000-3010. Support for performing DDAM calculations, as implemented in the Sierra/SD Finite Element Code, is documented both in the Sierra/SD Users' manual as well as in the DDAM Primer. The user is expected to be fully familiar with both cited NAVSEA publications.

DDAM is focused on five main phases, problem formulation phase, mathematical modeling phase, coefficient computation phase, dynamic computation phase, and the evaluation phase. DDAM as implemented in Sierra/SD, will focus primarily on the *evaluations* phase. This phase assesses modal analysis, modal filtering, displacement, velocity, acceleration, force, Von Mises stress calculations and all NRL sums.

DDAM does have a few limitations. The equipment to be analyzed must be represented as a linear elastic system with discrete modes. Damping is neglected. For very low frequency systems, DDAM may not be appropriate, and, where closely spaced modes exist, DDAM may produce excessive responses.

Current DDAM capabilities in Sierra/SD are as follows.

1. Select modes by frequency limits.

   Sierra/SD doesn't have the capability to select modes by frequency limits. The user must choose the number of modes, nmodes, in case 1, eigen, and continually increase

nmodes and restart or rerun eigen analysis until the desired frequency level is reached. This step is recommended to be completed before case 2: preddam and case 3: ddam are included and analyzed.

2. Include modal masses that include at least 1% individually of the total modal mass

   User must manually add any extra needed modes using "add" in modalfilter block.

3. The user may verify PREDDAM (case 2) and DDAM (case 3) by examining filtered modes, participation factors, modal weights, shock design coefficients and values found in the following text files.

   (a) preddam → PREDDAM_RESULTS.txt

   (b) ddam → DDAM_RESULTS.txt

4. Incorporate a 6*g minimum load requirements. This is hard coded and not a user option.

5. DDAM with complex models that consist of multiple element types.

6. DDAM Analysis using Super Element Capability

7. Displacements, Velocities, Accelerations, Forces output, by node, to and accessible in the output exodus file.

8. Von Mises stresses output, by element, to and accessible in the output exodus file.

9. Ship directions must match coordinate direction (Vertical → Z-direction, Athwartship → Y-direction, Fore and Aft → X-direction)

   Directions must match between preddam inputs modalfilter block, load block, and ddam input analysis_direction

10. DDAM Analysis using symmetric boundary conditions

11. DDAM Analysis of all models in parallel.

12. Post processing available for all DDAM modal variables and all DDAM NRL summed variables (Paraview - graphical, grope, blot, exo2mat - data extraction)

Future capabilities will include:

1. Analyses of closely space modes

2. Verification of antisymmetric boundary conditions

3. Demonstrate ability to perform DDAM analysis where input direction is arbitrary to orthogonal axis of the model (Currently the x,y,z directions must line up with the vertical, athwartship and for and aft ship directions)

4. Constraint forces
5. Elemental forces
6. All output written to the exodus file
7. Automatic inclusion of all modes which contributing 1% or greater to mass, that aren't included in the first filtering.
8. Output DDAM beam bending and shear stresses.

The DDAM solution is divided into three cases, case 1, eigen, case 2, preddam, and case 3, ddam.

Case 1, eigen: is modal analysis already implemented into Sierra/SD. The parameters, as they pertain to DDAM, are shown in the table below. More information on the keyword eigen and its inputs may be obtained in section 2.1.9.

Parameter	**Argument**	**Default**
nmodes	Integer	10
shift	Real	required for parallel analysis of floating structures

Note: The standard Sierra/SD specification for eigen analysis is "nmodes=<number>", where <number> is an integer value for the requested number of modes. This capability is limited to compute approximately half the modes of the system. For verification purposes a limited capability exists to compute all the eigenmodes. This capability runs only in serial, and only on very small models. It is selected by "nmodes=all".

Case 2, preddam: utilizes the eigenvectors and system mass matrix produced in case 1, eigen, to calculate and filter the modal participation factors, modal weights, individual modal weight percentage, cumulative modal weight, and cumulative modal weight percentage. Preddam requires two inputs shown below.

Parameter	**Argument**	**Default**
Modalfilter	String	None
Load	Integer	None

Modalfilter is implemented as a part of Sierra/SD and may be used as a part of other solution methods. Modalfilter provides a means of filtering data taken from the modal analysis and the participation factors. Modalfilter parameters as they pertain to DDAM are:

More information on the keyword modalfilter may be found in section 2.32. Finally, a load block should be defined. This load block (in Sierra/SD DDAM analysis) specifically applies to the value of gravitational loading (-386.4) and its direction must match

Parameter	Argument
Remove	Integer list
Cumulative mef	6 fractions
Add	Integer list

Section	Keyword	Parameter
Load	Gravity	val1 val2 val3
Scale factor multiplier	Scale	Val1

DDAM analysis direction defined in the MODALFILTER block and in case 3: DDAM. The parameters for the load block pertaining to preddam are as follows:

NOTE: preddam must be run following eigen in order to obtain necessary modal data for preddam calculations. Preddam may be run without case 3, ddam.

Case 3, ddam: uses filtered eigenvalues and mode shapes from case 1: eigen and filtered modal participation factors and modal weights from case 2: preddam, to calculate shock design coefficients and values, filtered modal displacements, velocities, accelerations, forces, Von Mises stresses, and NRL sum. The inputs are as follows:

Parameter	Argument	Choices
analysis_direction	String	VERTICAL or ATHWARTSHIP or FORE_AFT
ship_type	String	SURFACE_SHIP or SUBMARINE
mount_type	String	HULL or DECK or SHELL_PLATING
response_type	String	ELASTIC or ELASTICPLASTIC
velocity_coeffs	Values	v1 v2 v3 v4
acceleration_coeffs	Values	a1 a2 a3 a4 a5 (a5 case specific)

An example is shown in Figure 1.

NOTE: Case DDAM may only be run following Preddam and Eigen.

2.1.7 Directfrf

Option **directfrf** is used to perform a direct frequency response analysis. In other words, we compute a solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right)\overline{u} = \overline{f}(\omega) \tag{4}$$

where $\overline{u}$ is the Fourier transform of the displacement, u, and $\overline{f}$ is the Fourier transform of the applied force. The method used is to compute the frequency dependent matrix $A(\omega) = K + i\omega C - \omega^2 M$, and frequency component of the force at each frequency point at the output. The matrix equation is then solved once per frequency point. When a direct solver is used, this means that a complex factorization must be performed once per output. This can be

```
SOLUTION
    case 1
        eigen
            nmodes = 4
    case 2
            preddam
                modalfilter vertical
                load 1
    case 3
        ddam
            analysis_direction VERTICAL
            ship_type SURFACE_SHIP
            mount_type HULL
            response_type
            velocity_coeffs 1.4 5.2 220.1 12.2
            acceleration_coeffs 1.0 2.0 3.0 4.0 5.0
END
MODALFILTER vertical
    remove 1:500
    //                x     y     z     Rx    Ry    Rz
    cumulative mef    0.0   0.0   1.0   0.0   0.0   0.0  //VERTICAL
END
LOAD 1
    body
       gravity
          0.0        0.0  1.0
          scale -386.4
END
```

Figure 1: DDAM Example Input

very time consuming, and the **modalfrf** may be a better option for many situations (see section 2.1.20).

The force function must be explicitly specified in the load section, and MUST have a "function" definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the directfrf. They are identified in the **frequency** section along with an application region (see section 2.11). The range of the computed frequency response is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution.

In addition to the output that is sent to the .frq file, output may also be written to the **Exodus** file during a directfrf, provided that the keywords are specified in the **output** section. If nothing is specified in the **output** section, then nothing is written to the **Exodus** output files.

The expression "frf" is often interpreted as the ratio *of output/input. There are very good reasons for using that ratio, including the confusion that can come from scaling the Fourier transform. The Sierra/SD code computes the output and does not compute a ratio. If the ratio is required, use a function with unit load as the input.*

2.1.7.1 Padé Expansion: Computation of each frequency response is expensive because the system matrices must be computed, factored and solved once at each frequency. A cost effective approach is to use a much coarser computational grid for full computation, and use a rational function (or Padé) expansion for intermediate points.[3] The two additional parameters are required for the expansion, are listed in table 4. They are described below, and an example is shown in Figure 2. The theory is described in reference 4.

Parameter	Default	Description
interpolate points	0	Number of additional points to interpolate. If zero, no interpolation is performed.
interpolate order	20	Order of the rational function expansion.

Table 4: DirectFRF Parameters for Padé Expansion

[3] A rational function expansion is similar to a Taylor series expansion, but is capable of approximating resonant behavior.

```
solution
    case out
        directfrf
        INTERPOLATE POINTS = 50
        INTERPOLATE ORDER = 18
end
```

Figure 2: Padé Expansion Input Example. In this example, each exactly computed direct frequency response point will be separated by 50 interpolated values. These values will be determined using a Padé expansion of order 18.

2.1.8 Directfrf-LoadId

This solution method applies inverse methods (see section 5) in the frequency domain to determine a *load* as an output when provided with acoustic pressures or structural displacements and a finite element model. The finite element model determines matrices K,C,M. In general the user provides complex displacement and/or pressure at a set of nodes in the model [4]. This solution method determines the sideset load which corresponds best to the measured displacements and/or acoustic pressures.[5] Note that for structures, inversion is based on the signed magnitude of the traction, and the direction of the tractions is fixed.

The forward problem is defined in equation (4). The inverse equation is identical, but must be solved with optimization subject to regularization because measurements are available only at a subset of the analysis degrees of freedom.

An example input deck is given in Figure 3. As in a forward solution, most of the parameters of an inverse frequency method are found in other sections.[6] In particular look in the following sections of chapter 5:

Truth Table: The truth table (**data_truth_table** from the **inverse-problem** description) is a list of the indices of the global node numbers (a.k.a. target nodes) where displacements are measured. See section 5.2.2 for details of the file formats.

Data File: Experimentally determined "target" displacements are read from the **data_file** specified in the **inverse-problem** section (5.2.2).

Loads: (section 5.5) describes the inverse parameters for the **LOADS** block.

[4]For acoustics, the acoustic pressure is provided at a set of nodes, for structures the displacements are provided on a set of nodes.

[5]For structures the load is a pressure or traction on the sideset. Moments and point forces are not currently supported. For acoustics, the load is the acoustic acceleration on the sideset.

[6]The forward solution supports a Padé expansion. This is not supported for the inverse methods.

```
SOLUTION
    directfrf-loadid
END
INVERSE-PROBLEM
    data_truth_table = ttable.txt
    real_data_file = dataReal.txt
    imaginary_data_file = dataImag.txt
END
OPTIMIZATION
    ROLmethod = linesearch
    LSstep = Newton-Krylov
    LS_curvature_condition = null
    Max_iter_Krylov = 20
    Use_FD_hessvec = false
    Use_inexact_hessvec = false
    opt_tolerance = 1e-12
    opt_iterations = 2
END
LOADS
    sideset 301
        inverse_load_type = SPATIALLY_CONSTANT
        pressure=10
        function = 1
END
```

Figure 3: Direct frequency response load identification example input

Frequency: (section 2.11) describes the frequencies where the problem is computed. For inverse loads, these frequencies are essentially independent, and a load identification is performed at each frequency.

Optimize: (section 5.1) provides controls over the optimization problem.

Figure 3 reproduces example `inverse-problem` and `optimization` text input.

Section 5.6 contains a discussion of the current limitations with inverse load methods.

Sierra/SD uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 5.7 contains a discussion of the output file that is written by ROL.

2.1.9 DirectFrf-MaterialId

The **directfrf-materialid** method applies inverse methods in the frequency domain to determine *material parameters* when provided with both loads and displacements applied to a finite element model with unknown material parameters. Please refer to section 5 for an overview of the process.

The forward problem is defined in equation (4). The inverse equation is identical, but must be solved with optimization subject to regularization because measurements are available only at a subset of the analysis degrees of freedom. The solution provides the material parameters at the unknown element locations in the model.

As in a forward solution, the parameters of an inverse frequency method are found in other sections. In particular look in the following sections.

Truth Table: The truth table (**data_truth_table** from the **inverse-problem** description) is a list of the indices of nodes (a.k.a. target nodes) where displacements or acoustic pressures are measured. See section 5.2.2 for details of the file formats.

Block: (section 5.3) provides controls over the **Block** specifications for material inverse problems.

Material: (section 5.4) provides controls over the **Material** specifications for material inverse problems.

Frequency: (section 2.11) describes the frequencies where the problem is computed. For inverse material id, these frequencies are essentially independent, and a load identification is performed at each frequency.

Optimize: (section 5.1) provides controls over the optimization problem.

Inverse-Problems: (section 5) provides the connection to the measurement data.

There is no support for Padé expansion in inverse methods.

Sierra/SD uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 5.7 contains a discussion of the output file that is written by ROL.

2.1.10 Dump

The keyword **dump** will cause **Sierra/SD** to form matrices only and no solution will be obtained.

2.1.11 Eigen

The **eigen** keyword is needed to obtain the eigenvalues and mode shapes of a system. The parameters which can be specified for an eigensolution are shown in the table below, and described immediately following.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0
untilfreq	*Real*	0
ModalFilter	*string*	none
fluidloading	*yes/no*	no

nmodes This parameter specifies the number of modes to compute. With the exception of the direct eigen method (section 2.1.13), the modes are computed beginning with the lowest frequency. The calculation continues until **nmodes** have converged. An iterative Lanczos type method is used.

Specifying "nmodes=all" calls the direct eigen solver, which disables many of the other options, and is limited to small, serial models. See section 2.1.13.

Shift The shift parameter provides a means for solving singular systems. See the discussion below.

UntilFreq The **untilfreq** keyword provides an additional method of controlling the eigenspectrum to be computed. If this value is provided, then the analysis will be automatically (and internally) restarted until the frequency of the highest mode is at least

the value of the **untilfreq**. This restart capability is somewhat crude. There are always **nmodes** new modes computed on each calculation. Also, because there can be inaccuracies associated with restarting the eigensolver,[7] we restart a maximum of **5** times.[8]

ModalFilter The optional **ModalFilter** keyword provides a means of reducing the modes retained for output and for subsequent analysis. For more details, see section 2.33. You can also put the modal filter into a separate case, called **preddam**, 2.1.6.

fluidloading A coupled structural/acoustic eigen problem is quadratic eigen system. However, under certain approximations, the fluid may be represented as an added mass on the structure, and a real eigen problem results. See the "wet modes" section of the theory manual (1.23). The **fluidloading** enables that added mass calculation. The additional parameters associated with calculation of wet modes are shown in Table 5. In addition, all material blocks that use an acoustic material are treated alike as a fluid domain for mass loading. Fluid domains are determined by the material properties of the block or blocks.

Keyword	**value**	**Description**
fluidloading	yes/no	yes enables fluid mass loading
nmodes_acoustic	*int*	number of acoustic modes in expansion

Table 5: Fluid Loading Parameters

2.1.11.1 Eigenanalysis of singular systems

The eigenvalue problem is defined as,

$$(K - \omega^2 M)\phi = 0. \tag{5}$$

Where K and M are the stiffness and mass matrices respectively, and ω and ϕ are the eigen values and vectors to be determined. The problem may be solved using a variety of methods - the Lanczos algorithm is used in **Sierra/SD** . In this method, a subspace is built by repeated solving equations of the form $Ku = b$. For floating structures, or structures with mechanisms, K is singular and special approaches are required to solve the system. The two approaches used in **Sierra/SD** are described below.

Deflation. If it is possible to identify the singularity in K, then the null vectors of K are eigenvectors (with $\omega = 0$), and the system can be solved by insuring that no component

[7] We use the **ARPACK** Lanczos solver for the eigen problem. This solver maintains the orthogonality of the eigenvectors for a single batch of modes. However, when we restart it, we must deflate out the previously computed modes. There can thus be a slight loss of orthogonality. When we repeatedly restart, the effect can be significant.

[8] We anticipate that in the future, this keyword will be retired when better control methods are provided.

of the null vectors ever occurs in b. This approach is equivalent to computing the pseudo inverse of K.

The strength of deflation is that if the eigenvectors can be determined exactly, the Lanczos algorithm is unaltered and the remaining vectors can be determined somewhat optimally. The difficulty is ensuring that we have correctly determined the eigenvectors, especially when mechanisms or multipoint constraints exist in the model. Determination of the eigenvectors is often a tolerance based approach that has not been as robust as we would like.

Shifting. The second method involves solution of a modified (or shifted) eigenvalue problem.

$$((K - \sigma M) - \mu M)\,\phi = 0. \tag{6}$$

This system has the properties that the eigenvectors, ϕ, are unchanged from the original equation, and the eigenvalues, μ, are simply related to the original values. Namely, $\mu = \omega^2 - \sigma$.

The shifted problem benefits from the fact that $K - \sigma M$ can be made nonsingular (except in very rare situations). This is done by choosing σ to be a large negative value. Unfortunately, the Lanczos routine convergence is affected if σ is chosen to be too far from zero[2]. A reasonable value is $\sigma = -\omega_{elas}^2$, where ω_{elas} is the expected first nonzero (or elastic) eigenvalue.

On serial platforms we support only the shifted method. Because of the higher accuracy of direct solvers, a small negative shift is normally sufficient to solve the problem. This shift (usually -1) is computed automatically. We do not recommend that you override the defaults.

When using the FETI solver on parallel platforms both methods are available. If deflation is used, user input (and careful evaluation) may be required to ensure that all global rigid body modes have been properly identified. The relevant FETI parameters are `rbm` and `grbm_tol` as described in appendix 6.4. The shifted eigenvalue problem has proven to be more robust for many complex problems. Set the `grbm_tol` to a small value (e.g. 1e-20), and manually enter a negative shift. The output should still be examined to ensure that no global rigid body modes are detected.

If the model is not floating and has no mechanisms, the system is not singular, and no shift should be used (as it may slow convergence).

Example

[2] If σ is too large a negative value, many solves will be required to determine the eigenvalues (which consequently slows convergence). Another consequence is that often not all redundant, zero eigenvalues may be found. They may be found by reducing the shift, tightening tolerances or by restarting.

A **SOLUTION** section for an eigenanalysis with a **shift** of -10^6 , will look like the following, if 12 modes are needed. This shift would be appropriate for a system where the first elastic mode is approximately 150Hz.

```
Solution
  eigen
  nmodes 12
  shift -1.0e6
end
```

2.1.12 Eigen-MaterialId

Inverse methods may be applied in modal space to identify material parameters on a block or on an element. The user specifies some of the lowest modes of the structure. Currently only the modal frequencies are included in the objective function, but in the future the mode shapes will be allowed as additional data. The result (but not the methods) are similar to those using the Attune product (see section 2.35.1). The initial beta capability computes sensitivity with respect to the eigenvalues. The analyst provides the **eigen-materialid** keyword along with the standard parameters for modal analysis. See section 2.1.11. Modal filters (2.33) may not be applied in **eigen-materialid** analysis. The analysis requires input for measurement data, and for control of various optimization parameters. See the following sections for details:

Optimize: (section 5.1) provides controls over the optimization problem.

Inverse-Problems: (section 5.2) provides the connection to the measurement data. Of particular importance are the **modal_data_file** and **modal_weight_table** which are described in section 5.2.2.

An example input is shown in Figure 4.

Sierra/SD uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 5.7 contains a discussion of the output file that is written by ROL.

2.1.13 Direct Eigen Solution

The standard methods for eigen analysis are based on iterative methods, which build a Krylov subspace from which the solution is determined. These methods are optimized to find a few a the eigen modes of the system, which is typically what is needed for structural

```
SOLUTION
    eigen-materialid
        nmodes=12
        shift=-1e5
END

INVERSE-PROBLEM
    modal_data_file = modal_data.txt
    modal_truth_table = mtruth.txt
END

OPTIMIZATION
    optimization_package = ROL_lib
    ROLmethod = linesearch
    LSstep = secant
    scaleDesignVars = yes
    LS_curvature_condition = null
    Max_iter_Krylov = 50
    Use_FD_hessvec = false
    Use_inexact_hessvec = true
    opt_tolerance = 1e-10
    opt_iterations = 30
END
```

Figure 4: Eigen Material Id Example

analysis. The direct solution provides a means of computing all the modes of the system. It is limited to small, serial solutions. No shift is available. The method is provided for support of the need for small verification problems. It is selected using **"nmodes=all"** in a standard eigen analysis. Note: this option does not work if any multipoint constraints are present in the model.

2.1.14 AEigen

The standard eigenvalue methods of section 2.1.11 use the `ARPACK` eigensolver to arrive at the solution (see reference 5). This is a powerful, public domain solver and has been very successful. However, new solver approaches are being developed. The **AEigen** keyword selects the Krylov Schur method in the Anasazi package (see reference 6).[1] to solve the symmetric generalized eigenvalue problem of equation 5 for the lowest frequency modes. Anasazi is a module in Trilinos.

Parameter	Argument	Default
nmodes	*Integer*	10
anverbosity	*Integer*	1
shift	*Real*	
implicit_restarts	*Integer*	5
anblocksize	*Integer*	1
subspace_size	*Integer*	-1
aneigen_tol	*Integer*	10^{-16}

Table 6: AEigen optional parameters. If parameters are supplied which do not apply to a particular solver, the redundant parameters are silently ignored.

The Krylov-Schur method, BKS, is similar manner to the `ARPACK` solver, discussed in section 2.1.11. A subspace is built by repeatedly solving systems of the form $(K - \sigma M)$, where σ is the shift specified by keyword **shift**; see 2.1.11 for more discussion on shifting. The keyword **implicit_restarts** allows the specification of the number of restart steps. The default is 5, as with **eigen**. Unlike ARPACK, Anasazi employs block solvers. That is, the subspace generated by the BKS iteration adds multiple vectors at each step, where ARPACK adds only a single vector. The number of vectors is specified by the keyword **anblocksize**. The default block size for the BKS solver is 1, producing a similar iteration as with **eigen**.

The level of verbosity is controlled by the keyword **anverbosity**. Setting **anverbosity** allows us to specify what information is printed by the solver. The default value is 1, implying that the Anasazi solvers will output only errors and warnings. Each verbosity type is controlled by a single bit in the integer **anverbosity**. These types are listed in

[1] Eigenvalue problems are solved using a linear solver. The iterative solution of the eigenvalue problem is a nonlinear iteration that requires multiple linear solves.

Table 7. Each combination is valid, the combinations being formed by adding different verbosity values. For example, setting **anverbosity** to $25 = 0 + 1 + 8 + 16$ requests output for Errors, Warnings, Final Summary and Timing Details.

The **subspace_size** dictates the maximum size of the basis for BKS. A value of -1 signifies the default, which is computed based on the number of eigenmodes requested.

Verbosity type	Value
Errors	0
Warnings	1
Iteration Details	2
Orthogonalization Details	4
Final Summary	8
Timing Details	16
Status Test Details	32
Debug	64

Table 7: AEigen Verbosity Table

Figure 5 illustrates the "`SOLUTION`" section of an input file using the Krylov-Schur method, "`ansolver BKS`". A dozen of the lowest frequency modes are requested. The **shift** is -10^6. The shift is appropriate for a floating system where the first elastic mode is approximately 150 Hz. This produces an eigenanalysis equivalent to the example given in 2.1.11 for **eigen**. The verbosity level specifies that after finishing the eigenanalysis, the solver will print status information (number of iterations, current eigenvalues) and timing statistics.

```
Solution
   aeigen
     nmodes 12
     shift -1.0e6
     ansolver BKS
     anblocksize 1
     anverbosity 25
end
```

Figure 5: Aeigen example

2.1.15 Blk_eigen

The **blk_eigen** is used to provide the analyst with the ability to do an eigenanalysis on only a subsystem of the model defined by blocks. This is convenient if the analyst is concerned with only a certain part of the system. It is also used to implement nonlinear distributed damping as discussed in section 2.37.2. The parameters **shift** and **nmodes**

are supported in this solution and are defined in section 2.1.11. An example input file is shown in Figure 6.

```
SOLUTION
  case 'blockeig'
    blk_eigen
      block 1:3, 5, 20
      shift -1e6
      nmodes 10
    block 4, 6:19
      shift -1e5
      nmodes 6
  case 'nonlinear'
    nltransient
      nsteps = 200
      time_step = 5.0e-3
      rho = 0.8
END
```

Figure 6: Blk_eigen Example

The **Blk_eigen** method has some limitations. As in any eigenvalue problem, nonlinear element blocks use the currently defined tangent stiffness matrix. It is meaningless to compute eigen solutions on combinations of blocks that are over-constrained. For example, computing the eigen solution of a block of RBARS alone will fail.

2.1.16 Eigenk

The **eigenk** keyword is used to obtain the eigenvalues and eigenvectors of the stiffness matrix of the model. This is equivalent to **eigen** if the mass matrix is equal to the identity matrix. The same parameters apply.

IT IS CURRENTLY ONLY AVAILABLE ON SERIAL PLATFORMS.

2.1.17 Largest_Ev

The **Largest_Ev** keyword is used to obtain the largest eigenvalue of the system, i.e.

$$(K - \lambda M)\phi = 0$$

This eigenvalue is seldom of use in practical analysis. It is typically used in calculation of the stable time step for explicit time integration, where the analyst does *not* need to call this step directly. There are no arguments to the solution method.

2.1.18 Buckling

The **buckling** keyword is used to obtain the buckling modes and eigenvalues of a system. The parameters which can be specified for a buckling solution are shown in the table below. By default, if **nmodes** is not specified, a value of 10 is used.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0

The **shift** parameter indicates the shift desired in a buckling analysis. The shift value represents a shift in the eigenvalue space (i.e. ω^2 space). The value to select is problem dependent.

The **nmodes** parameter specifies the number of requested buckling modes. Most commonly, only the critical (lowest) buckling mode is of interest, and in that case **nmodes** would be specified to be 1. However, there are cases when the first few buckling modes are of interest, and thus this parameter can be specified in the same way as in eigenanalysis.

Unlike eigenanalysis, buckling solution cases require a **loads** block. This is because buckling is always specified with respect to a particular loading configuration. For example, for a pressure load applied on a sideset, the buckling analysis would indicate the critical amplitude of the applied pressure needed to cause buckling. The critical buckling load is computed as the product of the first (lowest) eigenvalue times the amplitude of the applied load. Thus, for the case

```
LOADS
  sideset 1
  pressure = 10.0
END
```

and a lowest obtained eigenvalue of 100.0, the critical buckling pressure would be computed as $P_{cr} = 100.0 \times 10.0 = 1000.0$. This would indicate that buckling would occur if the loading were applied as,

```
LOADS
  sideset 1
  pressure = 1000.0
END
```

Similar conclusions can be drawn about force loads on nodesets.

Buckling solutions cannot be computed for floating structures. If there are global rigid body modes, the solution may not be correct. Also, for meshes with MPCs, only parallel

solution is possible. Serial buckling solutions with MPCs cause a fatal error in the constraint transformations. This error will be eliminated in future versions.

One additional constraint on buckling is that currently beams and shells cannot be used in buckling solutions. We expect to eliminate this restriction in future releases.

Example

A **SOLUTION** section for buckling analysis with a **shift** of -10^6 , will look like the following, if only 1 mode is needed (i.e. if only the critical buckling load is of interest).

```
Solution
  buckling
  nmodes 1
  shift -1.0e6
end
```

2.1.19 Modal Participation Factor

The *modal effective mass* or *modal participation factor* are means of determining the nature of the eigenvectors of a solution. More particularly, the modal participation factor measures the fraction of an eigenvector that has the character of a rigid body mode of the system. This is used to determine the interaction of these modes with gravity loads. Any vector (including the eigen modes of a *constrained* system) may be expressed in terms of the eigen modes of an *unconstrained* system.[2] Thus,

$$v = \sum_{i}^{6} \gamma_i R_i + \sum_{i'=7}^{N} \beta_{i'} \bar{\phi}_{i'} \tag{7}$$

where R_i represents a rigid body mode, and $\bar{\phi}_{i'}$ represents the remaining (non-zero energy modes) of the unconstrained structure. When v is an eigenvector of the constrained system, the modal participation factor is defined as follows.[3]

$$\Gamma_{ij} = \frac{R_i^T M v_j}{\sqrt{\left(R_i^T M R_i\right)\left(v_j^T M v_j\right)}} \tag{8}$$

[2] All the eigen modes of an unconstrained system fully span the space of the constrained system, but the system may not converge rapidly. Likewise, the eigen modes of the constrained system together with the constraint modes span the same space.

[3] One important detail is the space on which these calculations are performed. In **Sierra/SD** we expand vector v to the unconstrained space, and we used a lumped representation for the mass matrix, M.

Parameter	argument	Description
write_table	Y/N	*"Yes"* write Γ table (default) *"No"* summary only
RCID	*string*	A string representing the rigid body modes to include in the calculation. The string "123" represents the translational degrees of freedom only (default). The value "123456" includes all 6. No embedded spaces or commas.
Blockwise	Y/N	if *No*, no blockwise data is reported.

Table 8: MPF Parameters

Obviously, Γ_{ij} is a mass normalized measure of the contribution of a given rigid body term, γ_i, to the vector, v_j. A summary term which represents the total fraction of a vector that is spanned by all rigid body modes is also useful.

$$\mathrm{MPF}_j = \sum_i^6 \Gamma_{ij}^2 \tag{9}$$

The **MPF** method computes these participation factors for the eigenvectors of a system. This method *must* be used as part of a multicase solution, and the previous case *must* be an eigenvalue problem (see section 2.1.11). Further, this method (by default) computes the modal participation factor on a block by block basis. Thus, those portions of the model that most contribute to the rigid body motion may be determined.[4] Then,

$$\Gamma_{ij}^k = \frac{R_i^T M^k v_j}{\sqrt{(R_i^T M R_i)\,(v_j^T M v_j)}} \tag{10}$$

Parameters for the **MPF** method are listed in Table 8.

Summary data from the calculation is written to the results file as described in Table 9. In addition, unless *write_table=no*, data will be written to an external text file. The format for the file is specified in the results file. It contains the block wise modal participation factors, Γ_{ij}^k of equation 10. An example is provided in Figure 7.

The external text file is intended to be easily read by external programs such as the matlab "load" command. It therefor has no header information. The data ordering is exactly the same as the table written to the echo file (which contains that header information). Each column is grouped first by block (in the order of the blocks in the genesis file), and then by degree of freedom. Usually there are either 3 or 6 dofs per block entry. Each row corresponds to a single mode.

[4]The overall modal contribution is *not* the sum of the block wise contributions, and contributions from individual blocks may cancel other blocks. See Table 9.

Data	**Value**	Description
MPF	$\sum_i (\Gamma_{ij})^2$	Overall mode$_j$ MPF
MPF-B$_k$	$\sum_i (\Gamma_{ij}^k)^2$	MPF for block k, mode j
MPF by RBM$_i$	$\sum_j (\Gamma_{ij})^2$	MPF for direction i

Table 9: MPF Summary data. Each mode, v_j, has contributions from each of these summary values.

```
SOLUTION
   case eig
     eigen nmodes=10
     shift=-1e5
   case out
     mpf
     blockwise=yes
     RCID=123
     write_table=yes
END
```

Figure 7: Modal Participation Factor (MPF) Example

The optional external text file (*.mpf) contains block-wise modal participation factors. The data is presented in tabular format, separated by white space. Most data analysis software tools can easily import this type of data for analysis and plotting (e.g., Microsoft Excel, OpenOffice Calc, Python, MATLAB, Octave, etc.). The MPF file contains no header information, so it is important to understand what each column and row represents. Each row of data corresponds to a mode. Columns represent modal participation factors calculated for each block and requested coordinate (controlled with "rcid"). The columns are grouped first by block, and then by degree of freedom. Blocks are written out in the order they are found in the genesis file (note: they are not sorted by Block ID or by the order they appear in the input file). Hence, if the exodus file contains two blocks and rcid=123 (default), the *.mpf file will contain six columns in the following order: Block1_x, Block1_y, Block1_z, Block2_x, Block2_y, Block2_z. If rcid=123456, then six columns per block (x, y, z, Rx, Ry, Rz) will be written out and there will be 12 columns.

Lumped or Consistent Mass: We always use the lumped mass for computation of the geometric rigid body vectors used in the modal participation factor calculation. These vectors are mass orthogonalized, and use of the consistent mass matrices for these efforts, especially when there are MPCs can be quite complicated in parallel. There is a small error introduced when the modes are computed using a consistent mass, and the rigid body vectors use a lumped mass. Refining the mesh reduces the problem, but most accurate results are obtained when the lumped mass is used (see section 2.2.3).

2.1.20 Modalfrf

Option **modalfrf** is used to perform a modal superposition-based frequency response analysis. In other words, the modalfrf provides an approximate solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right)\overline{u} = \overline{f}(\omega)$$

where $\overline{u}$ is the Fourier transform of the displacement, u, and $\overline{f}$ is the Fourier transform of the applied force.

If the damping matrix is zero, or if it can be diagonalized by the undamped modes, then the modalfrf solution uses the undamped modes for the superposition. Otherwise, for general damping matrices C, complex modes are used for the superposition. In either case, the modalfrf is performed in a multicase approach, where the modes (real or complex) are computed in a first case, and then the modalfrf is computed in a subsequent case.

Modal damping can be applied regardless of whether the modes are real or complex-valued.[5] However, proportional damping is currently only available when the modes are real-valued. For more details we refer to the section on damping 2.37.

2.1.20.1 ModalFrf with Real-valued Modes In the case where the undamped real modes are used for the superposition, two options are available for the modalfrf solution: the modal displacement method, and the modal acceleration method. In the case when complex modes are used, only the modal displacement method is available. In both the modal displacement and modal acceleration methods, the approximate solution is found by linear modal superposition. Once the modes have been computed, there is little cost in computation of the frequency response. The solution does suffer from modal truncation of course, but in the case of the modal acceleration method a static correction term partially accounts for the truncated high frequency terms. Thus, in general that method is more accurate than the modal displacement method. The most accurate, but also the most computationally expensive approach is the **directfrf** method described in section 2.1.7. The **qmodalfrf** solution (2.1.26) is an even faster modal method, for solutions not requiring a large amount of output.

For real modes using the modal displacement method, the relation used for modal frequency response is given below.

$$\overline{u}_k(\omega) = \sum_j \frac{\phi_{jk}\phi_{jm}\overline{f}_m(\omega)}{\omega_j^2 - \omega^2 + 2i\gamma_j\omega_j\omega}$$

Here $\overline{u}_k$ is the Fourier component of displacement at degree of freedom k, ϕ_{jk} is the eigenvector of mode i at dof k, and ω_j and γ_j represent the eigenfrequency and associated fractional

[5] One can use the "eigen" or the "qevp" solution methods.

modal damping respectively. In the case of complex modes, the equations need to be linearized and are more complex. We refer to that section of the theory manual 1.13.

For the modal acceleration method, the procedure for computing the modal frequency response is more complicated. The response is split into the rigid body contributions, and the flexible contributions. The number of global rigid body modes must be specified in the input file. For details on the theory, we refer to section 1.9 of the theory manual.

The modal acceleration method is typically much more accurate at finding the zeros of a function, but only slightly more accurate in finding the poles (or peaks) of the response. The cost is an additional factor and solve. It can be used on floating structures, but the additional factor involves only the stiffness terms (which are singular) and has no mass terms to stabilize the solution. Thus, it may be much more difficult to perform that solve than the other solves involved in the eigen analysis. In eigen analysis we recommend a negative shift for floating structures to remove the singularity associated with rigid body modes. No such approach is possible if you are using the modal acceleration method. Thus, significant "tweaking" of the FETI parameters may be required to accurately determine the global rigid body modes required for success of this method.

The force function must be explicitly specified in the load section, and MUST have a "function" definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time.

The following table gives the parameters needed for **modalfrf** section.

Parameter	Argument
nmodes	*Integer*
usemodalaccel	-
nrbms	*Integer*
complex	*yes/no*
lfcutoff	*Real*

The **nmodes** parameter controls the eigenanalysis (see section 2.1.11). The optional keyword, **usemodalaccel**, is used to determine whether to use the modal displacement or the modal acceleration method. If this keyword is specified, modal acceleration is used, otherwise the modal displacement method is invoked. If **usemodalaccel** is used, then

the number of global rigid body modes must be specified using **nrbms**. The keyword **complex** specifies to **Sierra/SD** if the modes to be used in the superposition are real or complex. Note that it is possible that both types of modes could be stored in the database, and thus the user would need to specify which set of modes to use in the superposition. The **lfcutoff** keyword provides a low frequency cutoff to filter modes. It is typically used to remove rigid body modes from the calculation. Modes with a frequency below this value are not included in the calculation. By default, all modes are retained.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.11). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

We note that, in addition to the output that is sent to the .frq file, output is also written to the **Exodus** file during a modalfrf, provided that the keywords are specified in the **output** section. If nothing is specified in the **output** section, then nothing is written to the **Exodus** output files.

In the case of undamped modes, the following is a multicase example of how the modalfrf could be specified.

```
SOLUTION
   case eig
     eigen nmodes=7
     shift=-1e5
   case out
     modalfrf
END
FREQUENCY
   freq_step=300
   freq_min=100
   freq_max=2500
   nodeset=12
   acceleration
END
```

2.1.20.2 ModalFrf with Complex Modes In the case when complex modes are used, only the modal displacement method is available. In this case the **qevp** solution case is used to compute the modes. There are currently three methods that can be used with the **qevp** solution case, and they are the **sa_eigen** method, the **anasazi** method, and the **ceigen** method. For more details, we refer to section 2.1.25.1. We note that in the case of complex modes, modal superposition is currently implemented for the **sa_eigen** method and the **anasazi** method. The **ceigen** method is currently not set up to work with a

subsequent modal superposition.

Also, when computing the complex modes in preparation for a modal superposition, we recommend using the **reorthogonalization** flag. When turned on, this flag searches for repeated modes and reorthogonalizes the eigenvectors of those modes. In many cases, repeated modes coming out of the eigensolvers are linearly independent, but not orthogonal. For more details, we refer to section 2.1.25.1.

In the case of complex modes, the following is an example.

```
SOLUTION
  case qevp
    qevp
    method = sa_eigen
    reorthogonalize = Y
    nmodes=20
    nmodes_acoustic = 5
    nmodes_structural = 5
  case out
    modalfrf
    complex = y
END
FREQUENCY
   freq_step=300
   freq_min=100
   freq_max=2500
   nodeset=12
   acceleration
END
```

2.1.21 Modalranvib

Option **modalranvib** is used to perform a modal superposition-based random vibration analysis in the frequency domain. The solution computes the root mean square (RMS) outputs (including the von mises stress) for a given input random force function. The resulting power spectral density functions may also be output for locations specified in the "frequency" section. The forcing functions (one for each input) must be explicitly specified in the ranload section (2.18), which MUST reference a "matrix-function" definition (see section 2.30).

The following table gives the **solution** parameters needed for **modalranvib** analysis.

Parameter	Argument	Comment
nmodes	*Integer*	do not use in multicase
noSVD	N/A	selection of reduction method
lfcutoff	*Real*	to eliminate zero energy modes
TruncationMethod	*string*	"none", "displacement" or "acceleration"
keepmodes	*Integer*	
CheckSMatrix	*true/false*	default is true

The **nmodes** parameter controls the eigenanalysis (see section 2.1.11). All keywords associated with eigen analysis are appropriate and available. It is recommended that the eigenanalysis be performed as the first step of a multicase solution. If used in a multicase analysis, "nmodes" should not be specified.

The *optional* keyword **noSVD** determines the method used to compute the RMS von Mises stress output. If **noSVD** is specified, then the simpler method which does not use a singular value decomposition is used. Additionally,that simpler method causes the second and fourth moments associated with von Mises stress to be computed and to be written to Exodus output. (The RMS von Mises stress and these two moments, along with the appropriate material properties, can be used in an manner suggested in[7] and discussed in[8] to estimate fatigue life in broad-band random excitation.) However, this method provides no information about the statistics of the stress. Only the RMS value and moments are reported.

The *optional* keyword **lfcutoff** provides a low frequency cutoff for random vibration processing. Usually, rigid body modes are *not* included in this type of calculation if RMS stress is computed. The **lfcutoff** provides a frequency below which the modes are ignored. The default for this value is 0.1 Hz. Thus, by default rigid body modes are not included in random vibration analysis. A large negative value will include all the modes.

The *optional* keyword **TruncationMethod** provides control over selection of the retained modes. By default modes are retained if they have any contribution to the stress. As stresses are proportional to displacement, the default method is "DISPLACEMENT". Rarely, one may want to avoid all truncation (NONE) or truncate based on acceleration contributions (ACCELERATION), which are much more heavily weighted to higher frequencies. Often zero energy modes contribute to a bad truncation, and a preferred means of controlling the truncation is to use the "lfcutoff" parameter and to ensure the integration does not go to zero frequency.

The *optional* keyword **keepmodes** is a method of truncating modes. By default, its value is **nmodes**. If a value is provided, the modes with the lowest modal activity will be truncated until only **keepmodes** remain. Note that this is a much different truncation procedure than simply truncating the higher frequency modes. Modal truncation is important because all of the operations compute responses that require order N^2 operations. Even if **keepmodes** is not entered, modes with modal activity less than 1 millionth of the highest active mode will be truncated.

The *optional* keyword **CheckSMatrix** can be used to turn off evaluation of the correlation matrix, S. This matrix is generated at each frequency, and must be positive semidefinite. A correlation matrix that is not positive semidefinite results in negative PSD results, which are not physically meaningful. Note, matrix evaluations are only enabled if PSD output is requested.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the random vibration spectra. They are identified in the **frequency** section along with an optional application region (see section 2.11). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra *does* depend on the magnitude of **freq_step** since it is used in the frequency domain integration.

In random vibration, the **frequency** block serves two purposes. First, it is used for the integration information for the entire model. Thus Γ_{qq} for the referenced papers[9] is integrated over frequency and used for all output. In addition, if an output region is specified in the **frequency** block, output acceleration and displacement power spectra may be computed for the given region at the required frequency points. At this time, only "acceleration" and/or "displacement" may be specified in the frequency block for random vibration analysis. This output is described in more detail below.

Random vibration analysis is a little trickier than most input. A number of blocks must be specified.

1. The **solution** block must have the required input for eigen analysis, and the keyword **modalranvib**.
2. The **RanLoads** block contains a definition of the spectral loading input matrix and the loadings. Note that the input, S_{FF} is separated into frequency and spatial components. The spatial component is specified here using **load** keywords. See section 2.18. The spectral component is referred to here, but details are provided in the matrix-function section.
3. The **matrix-function** section contains the spectral information on the loading. It references functions for the details of the load. The real and imaginary function identifiers for this input are specified here (2.30).
4. There must be a **function** definition for each referenced spectral function. Functions of time or frequency are further described in section 2.29.
5. There must be a **frequency** block that is used for integration and optionally also for output of displacement and acceleration output. See section 2.11.
6. As an undamped system is singular, some type of **Damping** block information needs to be provided. Modal damping terms are required.[6] See section 2.37.

[6]Proportional damping, such as is applied with the ALPHA and BETA terms, will NOT work in modalranvib.

7. **Boundary** conditions are supplied in the usual way, but the standard **loads** block is replaced by the input in the ranloads section. The loads block will be quietly ignored in random vibration analysis.

8. The **outputs** and **echo** sections will require the keyword **vrms** for output of RMS von mises stress. If the *stress* keyword is also found, then the natural stresses for solid elements will be output.[7] Quantities output are listed in Table 10.

All other input should remain unchanged.

Keyword	Output Variable	Description
Vrms	vrms	Root Mean Squared Von Mises Stress
	D1...D5	Von Mises Stress moments. Details in 10.
	Xrms	X component of RMS displacement
	Yrms	Y component of RMS displacement
	Zrms	Z component of RMS displacement
	Axrms	X component of RMS acceleration
	Ayrms	Y component of RMS acceleration
	Azrms	Z component of RMS acceleration

Table 10: ModalRanVib Output to Exodus File. The stress moments are not computed nor output if "noSVD" is selected. The RMS values of displacements and acceleration are not truly vectors.

2.1.21.1 Power Spectral Densities. One output from the random vibration analysis is a power spectral density or PSD (for displacement or acceleration). The power spectral density is a measure of the output content over a frequency band, and usually measured in units of cm^2/Hz or some similar unit. Acceleration PSDs are often measured in units of g^2/Hz.[8]

Like the input cross spectral forces, the output quantities are Hermitian, with 9 independent quantities at each frequency, at each output node for each type of output. Details of how these quantities are transformed in alternate coordinate systems are outlined in section 1.9 of the theory manual. The matrix quantities are diagrammed below. Quantities are output in the order A_{xx}, A_{yy}, A_{zz}, A_{zx}, A_{zy}, A_{xy}, A_{zxi}, A_{zyi}, A_{xyi}.

$$\begin{bmatrix} A_{xx} & A_{xy} + iA_{xyi} & A_{xz} + iA_{xzi} \\ A_{xy} - iA_{xyi} & A_{yy} & A_{yz} + iA_{yzi} \\ A_{xz} - iA_{xzi} & A_{yz} - iA_{yzi} & A_{zz} \end{bmatrix}$$

[7] The natural stresses are output in the following order: σ_{xx}, σ_{yy}, σ_{zz}, σ_{yz}, σ_{xz}, σ_{xy}. These stresses are linear functions of the displacement.

[8] Power spectral density output is requested in the **frequency** block. A collection of nodes is indicated and the *displacement* or *acceleration* keyword is entered. PSDs of displacement or acceleration are available.

Because the inputs are specified in terms of force cross-correlation functions, the standard procedure for applying loads often involves application of a large concentrated mass at the input location. The force may then be applied to the mass and the acceleration determined from $a = f/m$, where we assume that m is much larger than the mass of the remainder of the structure. Some confusion can arise in the scaling of the force.

The output PSD for acceleration is defined as follows.

$$G_{ij} = H^{\dagger}_{ki} S_{kl} H_{lj} \tag{11}$$

$$\langle a_i a_j \rangle = H^{\dagger}_{ki} \langle f_k f_l \rangle H_{lj} \tag{12}$$

where H_{lj} is the transfer function giving a_j/f_l.

Consider a single input, i.e. $k = l$, and with $f_k = m_k a_k$.

$$G_{ij} = H^{\dagger}_{ki} \langle m_k a_k a_k m_k \rangle H_{lj} \tag{13}$$

$$= (m_k^2) H_{ki} \langle a_k a_k \rangle H_{kj} \tag{14}$$

Thus, the acceleration PSD must be multiplied by the square of the mass to get the force PSD. Note that **Sierra/SD** uses the scale factor in the spatial force distribution, so the scale factor in **Sierra/SD** should be m_k.

2.1.22 Modalshock

The **modalshock** solution method is used to perform a modal superposition-based implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set. The following table gives the parameters needed for **modalshock**.

Parameter	Argument
nmodes	*Integer*
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
srs_damp	*Real*

The **nmodes** parameter controls the modal solution described in section 2.1.11. The time stepping parameters **time_step**, **nsteps** and **nskip** are described in the transient section (2.1.34).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.11). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The optional parameter **srs_damp** is a damping constant used for the shock response spectra calculation. Its default value is 0.03. Damping for the model is defined in section 2.37.

2.1.23 Modaltransient

Option **modaltransient** is used to perform a modal superposition-based implicit transient analysis. The following table gives the parameters needed for **modaltransient**. Damping for the model is defined in section 2.37.

Parameter	Argument	default
time_step	*Real*	none
nsteps	*Integer*	none
start_time	*Real*	0
nskip	*Integer*	1
load	*Integer*	*sec 2.16*
lfcutoff	*Real*	none
flush	*Integer*	50

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The optional **start_time** allows the analyst to define the start time of the transient simulation. It defaults to zero. The optional parameter **nskip** controls how many integration steps to take between outputting results. (It defaults to 1, which is equivalent to outputting all time steps). Time dependent loadings are applied by referencing the appropriate **load** and **function** sections (see 2.16 and 2.29).

The *optional* keyword **lfcutoff** provides a low frequency cutoff for processing the modes. The analyst may or may not wish to include rigid body modes in this type of calculation. The **lfcutoff** provides a frequency below which the modes are ignored in the modal superposition. The default behavior is to include all of the modes - if this parameter is present, modes below the cutoff will be ignored. A large negative value will include all the modes.

Modal transient should normally be executed as a later step of a multicase solution, where previous steps computed the eigenvalue response. However, for compatibility with earlier formats, **modaltransient** can be called as a single step solution (see section 2.1.11). In

that case the following eigen value parameters are also required. Note that in a single step solution (with no case structure), no **load** keyword is required, but a **loads** section must exist in the file (see section 2.15).

Parameter	Argument
nmodes	*Integer*
shift	*Real*

The parallel solution of modal transient may be slower than expected because while the eigen solution parallelizes very well, there is not enough computation to parallelize the modal calculation. In addition, **Sierra/SD** *computes the displacements at all locations in the model before subsetting to those nodes in the history file.*
If output is only required at a few locations, you may want to consider a qmodal transient solution (see 2.1.24) or a Matlab integration. Integration in Matlab will require the eigenvalues and vectors from the history file, and the modal generalized forces. These forces are written to `'ModalFv.m'`.

Modaltransient solutions do support restart. The format for a restart in **modaltransient** is given in detail in section 2.2.1. We note that one can restart both in the **eigen** part of the analysis, the **modaltransient** part, or both. In the latter case, **Sierra/SD** would read in the modes from the modal restart file, as well as the time history data from any previous transient restart files (direct or modal), and would then continue stepping in time. Modal transient analyses may be restarted from some other transient integrators. See Table 21.

An example of restart with the **modaltransient** solution is given below. In this case, the eigen solution is restarted prior to the modaltransient solution. The eigensolution would proceed as follows

```
SOLUTION
case 'eigen'
  eigen
  nmodes 10
  restart=write
END
```

and, subsequently, the eigen restart and modal transient would look follows

```
SOLUTION
```

```
case 'eigen'
  eigen
  nmodes 20
  restart=read
case 'modaltrans'
  modaltransient
  nsteps 100
  time_step 1.0e-3
  restart=write
END
```

We note that write option in the **modaltransient** case in the previous example would be needed to restart the modaltransient with additional subsequent time steps. For example, one could then do the following

```
SOLUTION
case 'eigen'
  eigen
  nmodes 20
  restart=read
case 'modaltrans'
  modaltransient
  nsteps 100
  time_step 1.0e-3
  restart=write
case 'modaltrans'
  modaltransient
  nsteps 200
  time_step 1.0e-3
  restart=read
END
```

2.1.24 QModaltransient

Option **qmodaltransient** is used to perform a fast modal superposition-based implicit transient analysis on a subset of the mesh. Instead of performing a full solution and then reducing the data to a subset, as in **modaltransient**, displacements are only calculated for the nodes in the history file. The parameters are identical to those specified for the Modal Transient solution and are discussed in *section 2.1.23*. Of course, a history section is also required or no output will be written.

2.1.24.1 Limitations:

1. The entire reduced model will fit in memory. The reduced model includes the eigenvectors of all desired modes on all the required output locations. Obviously, it also includes either the reduced mass and stiffness matrices, or the eigenvalues from which such can be constructed.[9]

2. Limited output is required. In particular, we don't know anything about the elements any more, so we could not compute stresses if they are not available as stress modes, Φ_σ. Applied force is also unavailable.

3. The loading is simple, i.e. there are no follower loads or user functions that depend on spatial data.

4. If a qmodal method is run and the files qDisp and qForce are created, ignore these files.

2.1.24.2 Example: The following example demonstrates the required components of a **qmodaltransient** input.

```
SOLUTION
   case eig
     eigen nmodes=10
     shift=-1e5
   case out
     qmodaltransient
     time_step=0.001
     nsteps=20
     load=1
END

HISTORY
   block 101
   disp
END
```

We note that, unlike **transient** and **modaltransient**, restarts are not currently supported for **qmodaltransient**.

2.1.25 QEVP

[9]Note that the reduced model could be either a pure modal model, a Craig Bampton model or some other such system.

2.1.25.1 Quadratic EigenValue Methods Comparison The quadratic eigenvalue problem is defined as,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{15}$$

The solution of the quadratic eigenvalue problem (eq. 15), has applications in a variety of physics solutions including coupled structural acoustics, general eigenvalue systems with damping, and gyroscopic systems for rotating structures. Various methods have been developed to address the solution to these problems. The solution to the problem is difficult, and knowledge of the types of systems encountered can help significantly in addressing the robustness of each of the methods. The methods are listed and described in the following paragraphs. Table 11 lists recommended procedures for different problem sets.

ANASAZI: We have recently introduced the Anasazi method. This can be used to address two problem areas, 1) the coupled structural acoustics problem, and 2) gyroscopic systems from rotating frames. Currently it requires that both the mass and the stiffness matrix be nonsingular. The Anasazi method is an ongoing development effort. Previous versions of **Sierra/SD** used the solution case qevp with no method keyword to denote the Anasazi method, and it is the default method to keep consistent with this syntax.

A couple of the parameters for the Anasazi solver for quadratic eigenvalue problems are described in the Table 6. Here shifts are not supported, and a warning message may be avoided by setting `shift` to zero. Restarts are not supported either; set `implicit_restarts` to 1. Restarts make the capability easier to use. Without restarts the user is required to set the subspace size, `subspace_size`, to be sufficiently large. The way that the algorithm works is to compute all of the modes, and then compare a kind of relative residual to `aneigen_tol`. If the residuals are large, the modes are not returned to the user. It can be helpful to use a larger value of `aneigen_tol` than the default, say 10^6 or 10^10. In this situation, it is helpful to set `anverbosity` to a large value, say 10. And then examine other diagnostic information there to ascertain the accuracy of the modes.

CEIGEN: The **Ceigen** method uses methods in ARPACK to solve the quadratic eigenvalue problem. Of methods in **Sierra/SD** , it is the oldest, and probably the least robust.

SA_EIGEN: The **SA_eigen** method solves a coupled structural acoustics problem by solving a linear, uncoupled eigenproblem on each of the domains, and using them as a basis to reduce the coupled equations to a dense system. The dense system is solved using LAPACK routines. The method is *only* applicable to structural/acoustic systems. It is fairly robust, but modal truncation can introduce significant errors. Some solutions can fail (or convergence may be very slow) because the decomposition tools know nothing about the two domains.

PROJECTION_EIGEN: The **projection_eigen** method solves the quadratic eigenvalue problem by projecting the problem into a subspace corresponding to the real-valued modes. This smaller subspace is constructed by neglecting the damping matrix,

symmetrizing the stiffness matrix, and solving a standard eigenvalue problem of the form

$$Ku = \lambda Mu \tag{16}$$

This smaller problem is then used as a basis for solving the original quadratic eigenvalue problem, which takes the form

$$Ku + \lambda Cu + \lambda^2 Mu = 0 \tag{17}$$

The original quadratic eigenvalue problem is then pre and post multiplied by the eigenvectors obtained from the subspace eigenvalue problem. This results in a very small quadratic eigenvalue problem which is then solved with a LAPACK method. Finally, the modes from the reduced space are projected back out to the space corresponding to the original quadratic eigenvalue problem.

As with the **sa_eigen** method, truncation error is a concern with the **projection_eigen** method. The more modes one takes, the smaller the truncation error.

Problem	Ceigen	SA_eigen	Anasazi	Projection_eigen
Damped Systems	Good	OK	Fails	Good
structural acoustics	Fails	Good	Good	Good
Rotational systems	N/A	N/A	Good	Good
Damped str/acoustics	Fails	OK	Fails	Good

Table 11: Comparison of Quadratic EigenProblem Methods

The various QEVP methods can be chosen using the keyword "method" followed by the name of the method (Anasazi, ceigen, sa_eigen, Projection_eigen). Below is a more detailed description of each QEVP method, their parameters, and examples of how to use them.

2.1.25.2 Anasazi The **Anasazi** method is one of a family of methods for addressing the quadratic eigenvalue problem. See section 2.1.25.1 for a comparison of these methods. The quadratic eigenvalue problem is defined as,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{18}$$

This method uses tools in Trilinos/Anasazi to develop a solution of this highly nonlinear problem. As currently implemented, the Anasazi method applies only to systems with a nonsingular mass and stiffness matrix and where the damping matrix, C, is asymmetric. Parameters for input are described in Table 12. An example is given below.

```
SOLUTION
   case qevp
      qevp
```

Table 12: Parameters for QEVP Anasazi Solutions

Parameter	**Argument**	**Default**	**Comment**
nmodes	*Integer*	10	number of modes
shift	*Real*	0	ignored
reorthogonalize	Y/N	"Y"	Reorthogonalize vectors
check_diagonal	Y/N	"Y"	Check that vectors diagonalize linearized system
ANverbosity	*Integer*	17	Anasazi verbosity
ANblocksize	*Integer*	1	Anasazi Block Size
ANeigen_tol	*Real*	1.0e-16	Eigen tolerance

```
        method=anasazi
        nmodes=14
        anverbosity=27
  END
```

2.1.25.3 Ceigen The "qevp" solution with "method=ceigen" is used to select complex eigen analysis using the ARPACK package. This computes the solution to the quadratic eigenvalue problem,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{19}$$

Note that two other solution methods may also be used to evaluate the quadratic eigenvalue problem. Each of these methods has its strengths and weaknesses. A comparison of these methods is provided in section 2.1.25.1.

The following table gives the parameters needed for complex eigen analysis.

Parameter	Argument	Default
nmodes	*Integer*	100
viscofreq	*Real*	1e-6
eig_tol	*Real*	0

The **nmodes** keyword indicates the number of modes to compute in the quadratic eigenvalue analysis. These modes are computed (and reported) as complex conjugate pairs.

The optional **viscofreq** keyword indicates the frequency at which the damping properties of visco elastic materials will be computed. It must be non-negative. The **viscofreq** parameter can be very confusing. In particular, visco elastic materials typically have high damping at lower frequencies, and lower damping at high frequencies. The **viscofreq** parameter sets a frequency from which we estimate *all* of the visco elastic damping. Thus, if **viscofreq** is small, the damping is large. In particular, if **viscofreq** is below the glass transition frequency, then damping appropriate to the low frequency modes will be

Table 13: Ceigen Tests

Name	**Description**
ceig	stiffness proportional damping
ceig_visco	visco elastic damping
ceig_dash	dashpot damping
steel_in_foam	complex mixed materials

used. *This high value of damping is applied to the entire spectrum.* It is generally better to over-estimate **viscofreq** than to underestimate it.

The reason for this difficulty is that even linear visco elastic materials generate a more complex equation than that shown in equation 19. With a single term in the Prony series, the equation of motion for a damped visco elastic structure can be written in the frequency domain.

$$\left(K + D\frac{s}{s+\omega_g} + Ms^2\right)u = f(s) \tag{20}$$

Where s is the *Laplace* transform variable and $\omega_g = 1/\tau$ is the reciprocal of the relaxation constant. Clearly this system is not a simple quadratic in s. Effectively, **viscofreq** approximates this system with the linearized system below.

$$\left(K + D\frac{s}{2\pi \cdot \mathbf{viscofreq} + \omega_g} + Ms^2\right)u = f(s) \tag{21}$$

Computation of quadratic eigenmodes is much more difficult than real eigen analysis. The system of equations is more difficult, and more "tricks" must be used to resolve issues that are generated, such as decreasing `eig_tol`. Even the post processing can be complicated. Like real eigen analysis, one must request displacement output in the `output` section (see 2.9.6). Now the output file contains 12 separate fields (six real and six imaginary) for the complex results. Few post processing tools know what to do with these results. More details are provided in section 1.11 of the theory manual.

Because of the difficulties with complex eigen analysis, it is important to understand the problems for which we have evaluated and tested it. The tests in the test suite are listed in Table 13.

2.1.25.4 SA_eigen The "qevp" procedure with method SA_eigen provides a means of computing the modal response of a coupled structural acoustic system, using a modal truncation basis. The quadratic eigenvalue problem describing this system can be written

```
SOLUTION
    case saeig
        qevp
            method=sa_eigen
            nmodes=20
            nmodes_acoustic=50
            nmodes_structure=26
            acoustic_lfcutoff=-1
            structural_lfcutoff=-1
            sort method = frequency
END
```

Figure 8: SA_Eigen Example

as follows.

$$\left(\begin{bmatrix} K_s & 0 \\ 0 & K_a \end{bmatrix} + \lambda \begin{bmatrix} C_s & L \\ -\rho_a L^T & C_a \end{bmatrix} + \lambda^2 \begin{bmatrix} M_s & 0 \\ 0 & M_a \end{bmatrix} \right) \begin{bmatrix} \phi_s \\ \phi_a \end{bmatrix} = 0 \tag{22}$$

Here the subscripts refer to structural or acoustic domains, ρ_a is the density of the fluid and L is a coupling matrix. Note that for this formulation, ϕ_a represents the acoustic velocity potential, which relates to the time derivative of the acoustic pressure, $\phi_a = \nabla \dot{u}_a$. See section 1.12 in the theory manual for more details.

The **SA_eigen** method solves this system by solving for the uncoupled eigen modes in the two domains, using them as a basis to reduce the coupled equations to a dense system, and solving the dense system. Thus, it uses a modal reduction technique similar to the Craig-Bampton methods (section 2.1.5) to generate a dense system of equations that are solved and results propagated back to the physical space. More details are available in the theory manual.

Parameters of the analysis are provided in Table 14, and an example is provided in Figure 8.

Boundary conditions are applied exactly as for the generalized eigenvalue problem. Exterior, non-reflecting boundary conditions may be applied, but modal convergence is poorer. Loads are irrelevant. Output is now complex, just as for the **ceig** case (2.1.25.3).

Limitations: This is a modal superposition method. The QEVP method is a more complete (but less robust) method which does not depend on modal truncation. The SA_eigen method works reasonably well for a variety of structural acoustic environments. Damping may be provided, but does tend to slow convergence. The method also depends on the solution to separate structural and acoustic subregion eigen problems. These solutions are not as robust as full system eigen analysis. Please see the notes in the verification manual for convergence details. Table 15 summarizes the status of this procedure.

Parameter	Args	Description
nmodes	*int*	Number of requested eigen modes
nmodes_acoustic	*int*	Number of free-free acoustic modes in the reduction. Defaults to 2·(nmodes).
nmodes_structure	*int*	Number of free-free structural modes in the reduction. Defaults to 2·(nmodes).
acoustic_lfcutoff	*Real*	Low frequency cutoff to filter acoustic modes. By default all modes are retained
structural_lfcutoff	*Real*	Low frequency cutoff to filter structural modes. By default all modes are retained. Used to eliminate negative modes
shift	*Real*	Eigen shift used in computation of the subregion modes. See 2.1.11.
sort method	*string*	**magnitude:** complex magnitude of λ **frequency:** Sort by frequency and then damping. **damping:** Sort by damping and then frequency. **truefreq:** Sort by frequency... avoiding zero energy round off. **none:**
linearization	*int*	**1** A = [0 I; -K -C]; B = [I 0; 0 M] **2** A = [-K 0; 0 M]; B = [C M; M 0]; **4** A = [0 -K; M 0]; B = [M C; 0 M]; These follow the linearizations in Tisseur
reorthogonalize	*string*	**no:** no reorthogonalization **yes:** reorthogonalize all modes
check_diagonal	*string*	**no:** no check for orthogonalization **yes:** check only redundant modes **all:** check all modes

Table 14: SA_Eigen Parameters

Low Frequency Cutoff: The parameters acoustic_lfcutoff and structural_lfcutoff remove low frequency modes before initiating the QEVP. This will reduce the numer of modes (nmodes_acoustic and nmodes_structure) in the analysis. Negative cutoff frequencies are allowed.

Analytic Reference	Verification Section	Tested	Parallel Test	User Test
11	8.6	Y	Y	some

Table 15: Verification Summary for SA_Eigen

Specialized Output: There are a few items that are output specifically for the *sa_eigen* procedures that can be very helpful in assessing the solutions.

StructuralFraction It is useful to know which modes participate in which regions. This is computed as follows.

Let ϕ be the right eigenvector computed on the reduced space. We subset ϕ into its structural and acoustic components. i.e.,

$$\phi = \begin{bmatrix} \phi_s \\ \phi_a \end{bmatrix}$$

We compute,

$$F_{structure} = \frac{\phi_s^\dagger \cdot \phi_s}{\phi_s^\dagger \cdot \phi_s + \phi_a^\dagger \cdot \phi_a} \tag{23}$$

where $\phi^\dagger$ represents the transpose and complex conjugate of ϕ. Note that these products are computed in the reduced space which has coordinates associated with each structural or acoustic eigen mode. In the reduced space, the mass matrix is identity, and the vector product, $\phi^\dagger \cdot \phi$ represents an energy norm.

AcousticFraction The acoustic fraction is the analogue of the structural fraction (eq. 23) applied the acoustic domain. It represents the portion of the system level complex eigenmode that is associated with the acoustic domain.

ErrorNorm We define a normalized modal energy residual.

$$E^n_{resid} = \frac{|\phi^\dagger(k + \lambda c + \lambda^2 M)\phi|}{\phi^\dagger K \phi} \tag{24}$$

Where ϕ and λ are the estimates of the eigenpairs computed using the modal approximation technique. The matrices, k, c and m are the fully assembled stiffness, coupling and mass matrices. This residual norm is a measure of the relative accuracy of the eigenvalue solution. It is available in both the text results files and the output **Exodus** files, and should be consulted to determine the convergence.

2.1.25.5 Projection_eigen The *Projection_Eigen* method is the most robust of all the solvers available for quadratic eigenvalue problems. Parameters of the *Projection_Eigen* solver are provided in Table 16. These parameters are identical to those for the *sa_eigen* method.

Parameter	Args	Description
nmodes	*int*	Number of requested eigen modes
shift	*Real*	Eigen shift used in computation of the subregion modes. See 2.1.11.
reorthogonalize	*string*	**no:** no reorthogonalization **yes:** reorthogonalize all modes
check_diagonal	*string*	**no:** no check for orthogonalization **yes:** check only redundant modes **all:** check all modes
sort method	*string*	**magnitude:** complex magnitude of λ **frequency:** Sort by frequency and then damping. **damping:** Sort by damping and then frequency. **truefreq:** Sort by frequency... avoiding zero energy round off. **none:**

Table 16: Projection_Eigen Parameters

2.1.26 QModalfrf

Option **qmodalfrf** is used to perform a fast modal frf analysis on a subset of the mesh. The parameters are identical to those specified for the Modal FRF solution and are discussed in *section 2.1.20*. The only exception is that the option **usemodalaccel** has no affect on a **qmodalfrf** solution; it is always displacement based. A history section is also required or no output will be written. A **qmodalfrf** solutions has the same limitations as the **qmodaltransient** solution discussed in the previous section.

The example in section 2.1.20 applies to the **qmodalfrf** method. Simply replace the *modalfrf* keyword in the solution block with *qmodalfrf*.

2.1.27 NLStatics

The **NLstatics** keyword is required if a nonlinear static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$, where K is now a function of u. The following table gives the parameters needed for nonlinear static analysis.

Parameter	Argument	Default
max_newton_iterations	*Integer*	100
tolerance	*Real*	1e-6
num_newton_load_steps	*Integer*	1
update_tangent	*Integer*	101

Four parameters control the conventional Newton method. Newton methods are nonlinear solution algorithms employed to solve the residual force equations. The residual vector, r, is the difference between the internal force vector, p, and the external force vector, f. The strategy drives the residual to zero.

$$r = p - f \tag{25}$$

The internal force vector is a function of the structural displacements (and possibly velocities). External forces can also be a function of the structural displacements in the case of follower loads such as surface pressure loads.

The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *displacement* decreases below **tolerance**, the loop completes successfully. If the iteration count exceeds **max_newton_iterations**, the Newton loop is considered to have failed.

The **num_newton_load_steps** keyword controls the number of load steps used to incrementally step up to the final equilibrium position. Large loads may cause the Newton algorithm to diverge. If this occurs, increase the number of load steps applied. Displacements will be output after each load step which may be animated similar to transient dynamics simulations.

The **update_tangent** keyword controls how often the tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to update the tangent stiffness matrix at the beginning of a load step only. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the tangent stiffness matrix is rebuilt after each Newton iteration. For highly nonlinear (difficult) problems, this option may be optimal, but for most problems the extra cost incurred in recomputation and refactoring of the tangent stiffness matrix should be amortized over several solves. Note, for this option to improve Newtons method, the element types in the model have to have the tangent stiffness method implemented.

An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a nonlinear statics solution'
  nlstatics
  tolerance             = 1e-6
  max_newton_iterations = 100
  num_newton_load_steps = 10  // split load into 10 increments
```

```
  update_tangent        = 1   // full-newton algorithm
end
```

2.1.28 NLTransient

The **NLtransient** solution method is used to perform a direct implicit nonlinear transient analysis. The following table gives the parameters needed for nonlinear transient analysis.

The nonlinear transient analysis is performed according to methods described in Hughes. A projector, corrector step is used. Note that for a linear system the NLtransient analysis will require two solves per time step.

Parameter	Argument	Default
time_step	*Real*	-
nsteps	*Integer*	-
nskip	*Integer*	1
start_time	*Real*	0
flush	*Integer*	50
rho	*Real*	Newmark beta
max_newton_iterations	*Integer*	100
tolerance	*Real*	1e-6
update_tangent	*Integer*	101

The time step control parameters, **time_step**, **nsteps**, **nskip**, **start_time**, and **flush** are described in the **transient** section above, section 2.1.34. The parameter **rho** is the same as described in the previous section. We note that, as in the case of linear transient analysis, multiple time steps can be specified in nonlinear transient analysis. The syntax for this is the same as described in the section on linear transient analysis.

Four parameters control the conventional Newton method used to solve the residual force equations. The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *acceleration* decreases below **tolerance**, the loop completes successfully. If the iteration count in a given time step exceeds **max_newton_iterations**, the Newton loop is considered to have failed. Thus, note that **max_newton_iterations** is not the limit for the total number of Newton iterations, but the limit on the number of iterations per time step.

In a nonlinear statics analysis, load stepping can be used to help the convergence of the Newton loop by cutting the total load into a series of incremental steps. This is controlled with the **num_newton_load_steps** keyword. However, in nonlinear transient analysis, load stepping makes no sense since the dynamic response of a structure subjected to a

total load is different than if it were subjected to a series of incremental loads. In effect, the load stepping is replaced by time stepping in the case of nonlinear transient analysis. Thus, the keyword **num_newton_load_steps** is inactive for nonlinear transient analysis.

For nonlinear transient problems, if Newtons method diverges, either the tangent stiffness matrix has to be updated more often (see **update_tangent**) or the time-step should be decreased.

The **update_tangent** controls how often the dynamic tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to 101, and thus unless a given Newton loop takes more than 101 iterations, the tangent matrix will not be updated by default. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the dynamic tangent stiffness matrix is rebuilt after each Newton iteration. Note that currently there is no option for forcing a tangent update at the beginning of each time step, unless the **update_tangent** keyword is set to exactly the number of Newton iterations taken per time step. For highly nonlinear problems, some control of this option is recommended. Note, for this option to improve Newtons method, the element types in the model have to have the dynamic tangent stiffness method implemented.

2.1.29 Receive_Sierra_Data

Coupling of **Sierra/SD** *Through The Sierra Framework*

Calculations in Sierra codes such as Sierra/SM may be transferred to Sierra/SD. This provides the ability to compute very nonlinear responses in an explicit code, and follow that by a mildly nonlinear, implicit linear or modal calculation in Sierra/SD. There is a detailed discussion of SM/SD handoff in section 8.2 of the HowTo document.

A solution method named **Receive_Sierra_Data** facilitates the transfer of data, which may occur either through the sierra framework or an Exodus input file written from the sierra application. Table 17 gives a summary of the available parameters for this method.

Table 17: Receive_Sierra_Data Parameters

Parameter	Description
read_from_file	read data from a file instead of an in-core transfer
equilibrium	See description below
no_geom_stiff	See description below
transfer iterate	for Gemini coupling (see description below)

The method used for the transfer depends on the executable built. As currently configured, the standard Sierra/SD executable must use the file transfer. Specially linked executables, such as *eagle* can be used for the in-core transfer of data. These executables contain linkage for Sierra/SM in addition to Sierra/SD.

The Receive_Sierra_Data solution makes sense only in the context of a multi-case solution. An example is given below, where preload data is received from a file, and a modal (eigen) solution is computed after receiving the preload data.

```
SOLUTION
    case xfer
       receive_sierra_data
          read_from_file
          equilibrium
          no_geom_stiff
    case eig
       eigen nmodes=40 shift=-3e6
END
```

When the transfer is to occur via an exodus file, the option **read_from_file** should be specified. This tells Sierra/SD to read the transfer data off of the incoming exodus file (as specified by the **geometry_file** specification in the file section (see section 2.12)), rather than expecting data from a transfer. To be meaningful, that file will contain data from some previous analysis.

The **receive_time_step** parameter may be applied *only* to a file transfer. It controls which time step in the input exodus file contains the data. By default, the data corresponding to the first time step is read in by Sierra/SD.

The **equilibrium** and **no_geom_stiff** keywords have complementary uses. The **equilibrium** keyword is used when the data coming into Sierra/SD (either through a file or a transfer) is in static equilibrium. When this parameter is used, no force terms will be added to the right hand side in Sierra/SD. The only effect of the preload in such a case would be to change the stiffness matrix through the contribution of a geometric stiffness matrix. When the **equilibrium** keyword is not used, the preload has two effects; it contributes a geometric stiffness matrix, and it also contributes a forcing term. For a modal analysis, the **equilibrium** keyword will have no effect, since it only effects the right hand side term.

The **no_geom_stiff** keyword can be used to ignore the contribution of the geometric stiffness matrix when data is being read into **Sierra/SD** (either through a file or a transfer). This can be useful for debugging purposes, if for example it is suspected that the geometric stiffness matrix is contributing to negative eigenvalues. Otherwise, this parameter should be avoided, since it changes the stiffness of the system.

The *transfer iterate* keywords are used in coupling of Sierra/SD with the fluids code Gemini. We don't go into details here, but instead point the reader to the Gemini Interface Users Guide. That guide can be found in the Sierra/SD HowTo document, which can be found on the Sierra/SD website.

2.1.30 Statics

The **statics** keyword is required if a static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$. An example **SOLUTION** section is shown below.

```
Solution
   title 'Example of a statics solution'
   statics
end
```

2.1.31 Subdomain_Eigen

The **subdomain_eigen** keyword is used to obtain the eigenvalues and eigenvectors of the mass and stiffness matrix of the model on a subdomain basis. This is useful mainly for debugging distributed solutions. It is obviously decomposition dependent, and has no physical meaning. The parameters are listed below.

Parameter	Argument	Default
nmodes	*Integer*	10
shift	*Real*	0

Many domain decomposition tools (such as FETI-DP) depend on non-singular subdomain stiffness matrices. Running subdomain_eigen on these systems reveals the condition of the system that is to be solved. For FETI-DP, the system of interest is the subdomain defined with the corner nodes clamped. This can be determined using the following procedure.

1. Set the FETI parameter **prt_debug**=3 in the FETI section (see section 2.4.3). Running a standard analysis (i.e. statics, transient analysis or eigen) will output the "`corners.data`" file. This file should normally be written properly even if the analysis fails.

2. Copy the file to a new name, and modify it to contain only the global node ids. This is the first column of the file.

3. Use the **node_list_file** option to clamp the corner nodes in the file (see section 2.14.3).

4. Run **Sierra/SD** using the **subdomain_eigen** option. Ask for 14 modes or so. A very small first mode indicates a singular system for which our corner selection algorithm has not properly constrained the subdomain.

2.1.32 Tangent

The **tangent** solution step is only relevant as part of a multicase solution (see paragraph 2.1.1). It forces an update of the tangent stiffness matrix. It is typically used following a nonlinear solution step to ensure that the following step begins using the tangent stiffness matrices computed from the previous result. However, it may also be used following a linear solution step, in which case the stiffness matrix is recomputed based on the current value of displacement.

The tangent stiffness matrix is assembled at the subdomain level from computations at the element level. It represents the partial derivative of the force with respect to the displacement, i.e.

$$K_{tangent} = \frac{\partial f}{\partial u} \tag{26}$$

In eigen analysis, the tangent stiffness matrix replaces the linear stiffness matrix in the eigenvalue equation. This permits computation of modal response following a preload. In nonlinear transient dynamics, the tangent stiffness matrix is used in the Newton (or other) iteration scheme used to reduce force residuals.

2.1.33 Transhock

The **transhock** solution method is used to perform a direct implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set (all node sets are defined in the Exodus file). The following table gives the parameters needed for transient shock analysis.

Parameter	Argument
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
srs_damp	*Real*

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.11). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls

the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The keyword **srs_damp** is a damping constant used for the shock response spectra calculation and is optional. It represents the damping for each single degree of freedom oscillator in the shock spectra computation. Its default value is 0.03. Figure 9 provides an example.

The shock spectrum procedure will only compute acceleration results. The options specified in the OUTPUT and ECHO blocks are used in the transient portion of the analysis, but are ignored for the post-processing of the transient results into shock spectra. Thus, if displacement, velocity, and/or acceleration is selected in the OUTPUT and/or ECHO sections for a shock spectra analysis, the results echoed to the output listing or the Exodus output file will be time history results as requested, but the only shock spectra results will be for acceleration response for the nodes in the specified node set. *The calculated shock spectra are written only to the frequency file (*.frq); they are not output to the Exodus results file.*

```
SOLUTION
    transhock
        time_step .00005
        nsteps 500
        nskip 1
        srs_damp .03
END

FREQUENCY
    freq_min 100.
    freq_max 10000.
    freq_step 100.
    nodeset 3
    acceleration
END
```

Figure 9: Transhock Example Input

2.1.34 Transient

The **transient** solution method is used to perform a direct implicit transient analysis. The following table gives the parameters needed for transient analysis.[2]

[2] In addition to the displacement based linear transient dynamics driver, there is an older, acceleration based driver. The old driver may be selected using the **old_transient** keyword. This driver is not recommended unless sensitivity analysis is required. It is no longer fully maintained, and will be removed in future releases.

Parameter	Argument	Default	Purpose
time_step	*Real*	1	set the time step
nsteps	*Integer*	100	set the number of steps
nskip	*Integer*	1	set output frequency
start_time	*Real*	0	start time for transient analysis
flush	*Integer*	50	control file buffering
rho	*Real*	none - see below	select time integrator
transfer	*string*	none	Use sierra transfer with Gemini

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The optional **start_time** allows the analyst to define the start time of the transient simulation. It defaults to zero. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameter **flush** controls how often the **Exodus** output file buffers should be flushed. Flushing the output ensures that all the data that has written to the file buffers is also written to the disk. This parameter also controls the frequency of output of restart information if requested. Too frequent buffer flushes can affect performance. However, in a transient run, data integrity on the disk can only be assured if the buffers are flushed. A **flush** value of -1 will not flush the **Exodus** output file buffer until the run completes. The default value is to flush the buffers every 50 time steps.

We note that multiple time step values, along with the corresponding number of steps, can be specified for transient analysis. This can be useful for separating the simulation into a section of small time steps followed by a section of larger time steps, or vice versa. The following provides an example of the use of multiple time steps.

```
solution
  time_step     1e-5     1e-3
  nsteps        100      500
  nskip         10       1
end
```

In this case, the user requested 100 time steps of $\Delta t = 1E-5$, followed by 500 time steps of $\Delta t = 1E-3$. There is no practical limit on the number of such regions that may be specified.

Integrator selection

Two time integrator schemes are available for direct time integration. The method and the parameters of the integrator are selected using the keyword **rho**. If this keyword is not

found, the time integrator defaults to a standard Newmark-Beta integration scheme[3]. If the **rho** parameter is used, then the Generalized Alpha method[12][13] is used, and the value of the numerical damping is controlled by **rho**.

**** IMPORTANT ****

Because of limited accuracy in the solvers, the Newmark-Beta integrator is conditionally unstable. If no damping is provided, it occasionally diverges as time progresses. This is described in a little more detail in section 1.1 of the theory manual. Therefore it is strongly recommended that either proportional damping or numerical damping be used in all time integration.

The parameter **rho** defines the Numerical damping of the Generalized Alpha method. **Rho** varies from 0 (maximal damping case) to 1 (minimal damping case). **If rho is not specified in the input file, the integrator defaults to the Newmark beta method.** Otherwise, the code uses the value of **rho** given by the user to compute the parameters needed for the Generalized Alpha method. Therefore, there is no value default for **rho**, as shown in the table above, since if it is not specified the code uses the Newmark beta method instead. If **rho** is specified to be greater than 1 or less than 0 an error message is printed. The three parameters **newmark_beta**, α_f, and α_m in the Generalized Alpha method are computed automatically, given the value of **rho**, and thus these need not be specified by the user. More detailed information on the implementation, and references can be found in the description of the method in the **Sierra/SD** program_notes and theory manual.

In order to achieve second order accuracy and unconditional stability, we must satisfy the following conditions.

$$\begin{aligned} \alpha_m < \alpha_f <= \frac{1}{2} \\ \gamma_n = \frac{1}{2} - \alpha_m + \alpha_f \\ \beta_n \geq \frac{1}{4} + \frac{1}{2}(\alpha_f - \alpha_m) \end{aligned} \tag{27}$$

The code automatically computes these parameters such that they meet these criteria.

[3]The Newmark-Beta integration is described in detail in most finite element text such as Cook or Hughes.

Specifically,

$$\begin{aligned}
\alpha_f &= \rho/(1+\rho) \\
\alpha_m &= (2\rho-1)/(1+\rho) \\
\beta_n &= (1-\alpha_m+\alpha_f)\cdot(1-\alpha_m+\alpha_f)/4 \\
\gamma_n &= 1/2-\alpha_m+\alpha_f
\end{aligned}$$

We note some special cases of interest. If $\rho = 0$, we have that $\alpha_f = 0$ and $\alpha_m = -1$. This is the maximum damping case. If $\rho = 1$, we have that $\alpha_f = \alpha_m = \frac{1}{2}$, which yields $\beta_n = \frac{1}{4}$, and $\gamma_n = \frac{1}{2}$. This is similar to the classical undamped Newmark-beta method, although we note that it is a different algorithm since $\alpha_f = \alpha_m = \frac{1}{2}$ implies some lagging in the time-stepping procedure. The classical undamped Newmark-beta method has $\alpha_f = \alpha_m = 0$.

Unlike the proportional damping parameters, there is no direct relation between **rho** and an equivalent modal damping term. A value of `rho=0.9` is recommended for most analyses. The Generalized Alpha integrator imparts numerical damping to the solution that most strongly affects *high* frequency content. Users must to check that the damping in the frequency range of interest is physical. For example with a time step size of $1e-5$, damping has the most effect at frequencies above the Nyquist frequency $.5e+5$.

2.1.34.1 Transfers: **Sierra/SD** can couple at each time step with the *Sierra Transfer Services*. Loads and boundary conditions may be supplied by an external application, while **Sierra/SD** supplies displacements and velocities on the interface.[4] This coupling is implemented for the **Gemini** application for underwater applications. Using the transfer services may impact other parameters. For example, **Gemini** typically controls the time step of both the fluid and the structural regions. Such a transfer is indicated by the **transfer** keyword. See section 2.2.7 for more details. An example is shown in Figure 10. Use of the explicit integrator with Gemini is discussed in section 2.1.37.2 (page 69). Note that both the explicit and implicit integrators may be used with Gemini, but there are definite performance trade-offs.

Both explicit and implicit transient solutions in **Sierra/SD** may subcycle if they need to take a smaller time step than the one provided via the transfer services. Explicit transient calculates a stable time step based on the largest eigen value of the problem and must not exceed that step size regardless of the transferred step size.

Implicit transient can take arbitrarily large steps, but more accurate results might be obtained by telling to take smaller steps and subcycle similar to explicit transient. In this case the subcycle time step is specified through the user input time_step parameter. If no time_step is specified in the **Sierra/SD** input file, the transferred time step is always used. If the time_step value is larger than the transferred time step, the transferred step size is used. If it is smaller, **Sierra/SD** will take multiple steps of the specified size to get to the

[4] See the "receive_sierra_data" section (2.1.29) for hand-off operations with Sierra/SM.

transferred step. This could result in the last time step being smaller than the previous steps. To avoid large inconsistencies in the output file, if this final step is less than ten percent of the size of the previous steps, the time is amortized over all previous steps and all steps will be the same, slightly larger size.

```
SOLUTION
    case coupled
        transient
        // no loads or bc needed
        transfer iterate
END
```

Figure 10: Transient/Transfer Example for Gemini type Coupling.

2.1.35 Transient-LoadId

The **transient-loadid** solution provides a means of determining the input loading that results in a time domain solution. The material and model parameters do not change during the solution. The method is one of several inverse method capabilities which are described in general in section 5. The solution is described as an "inverse" method because the displacement variables are known on only a subset of the reference locations. The differential equations of the forward problem are discretized and solved repeatedly to minimize the residual error.

Inputs: With a few exceptions (see limitations below) the parameters for the forward **transient** solution case (2.1.34) apply to the **transient-loadid** as well. In addition, various parameters provide data for the inverse problem. Some of these are detailed below. Please refer to section 5 for a general description of inverse methods.

Loads: (section 5.5) describes the inverse parameters for the **LOADS** block.

Truth Table: The truth table (**data_truth_table** from the **inverse-problem** description) is a list of the indices of nodes (a.k.a. target nodes) where displacements are measured. See section 5.2.2 for details of the file formats.

Data File: Experimentally determined "target" displacements are read from the **data_file** specified in the **inverse-problem** section (5.2). (section 5.5) describes the inverse parameters for the **LOADS** block.

Optimization: (section 5.1) provides controls over the optimization problem.

Inverse-Problems: (section 5) provides the connection to the measurement data and regularization parameters.

An example is shown in Figure 11.

Section 5.6 contains a discussion of the current limitations with inverse load methods.

```
SOLUTION
    transient-loadid
    nsteps = 100
    time_step = 1e-3
    nskip = 1
END
INVERSE-PROBLEM
    data_truth_table = ttable.txt
    data_file = dataReal.txt
END
OPTIMIZATION
    ROLmethod = linesearch
    LSstep = Newton-Krylov
    LS_curvature_condition = null
    Max_iter_Krylov = 20
    Use_FD_hessvec = false
    Use_inexact_hessvec = false
    opt_tolerance = 1e-12
    opt_iterations = 2
END
LOADS
    sideset 301
        inverse_load_type = SPATIALLY_CONSTANT
        pressure=10
        function = 1
END
```

Figure 11: Transient Load Identification Example

Sierra/SD uses the Rapid Optimization Library (ROL) for solving optimization problems. During the optimization solution ROL writes an output file, *ROL_Messages.txt* that contains convergence information. Section 5.7 contains a discussion of the output file that is written by ROL.

2.1.36 TSR_Preload

The **tsr_preload** solution method reads an **Exodus** file with a previously computed Thermal Structural Response (TSR) into **Sierra/SD** for a subsequent statics or transient dynamics analysis. This is not a fully coupled calculation. Rather, stress results are read

from the file, an equivalent internal force is computed, and that internal force is combined with the applied force throughout the transient run. A **tsr_preload** may only be specified as part of a multicase solution, and it must be followed by a transient dynamics or statics solution (see paragraphs 2.1.1 and 2.1.34 respectively).

Note that since the stresses are actually converted into a force, and since there is no immediate deformation in transient dynamics, the elastic stresses output by **Sierra/SD** will be very small initially, i.e. they will not contain a contribution from the thermal stress. However, at large times, the deformation from the internal force will result in an elastic stress opposite to that of the thermal stress. The `linesample` method 2.13 recovers the input thermal stress as an output quantity (in either Matlab or **Exodus** format).

The **tsr_preload** solution method is considered to be a temporary solution to a more complicated problem. In the future, TSR analysis will involve coupling to other mechanics codes. In many cases a thermal load (section 2.15.7), may provide equivalent capability.

Stress data input must be stored in the geometry file, i.e. the **geometry_file** specified in the **FILE** section (see paragraph 2.12). Data in the **Exodus** file must strictly match these criteria. There must be only one time step in the result. That time step must have a number of different element fields defined. These correspond to the six stresses and up to 27 different integration points of a hex20. Other solid elements are also supported. For those elements only the number of integration points applicable to that element are used. Unused integration values will be ignored. If in doubt, provide the extra integration data as missing integration points do NOT provide an error - rather they set the value to zero. Shell and beam type elements are not supported in tsr_preload.

The labels for the stresses must be as shown in the table below. In each case, replace %d with an integer representing the integration point value (0 to 26). Do not zero pad.

Name	Definition
`SIGXX_%d`	σ_{xx}, the xx component of stress
`SIGYY_%d`	σ_{yy}, the yy component of stress
`SIGZZ_%d`	σ_{zz}, the zz component of stress
`SIGYZ_%d`	σ_{yz}, the yz component of stress
`SIGXZ_%d`	σ_{xz}, the xz component of stress
`SIGXY_%d`	σ_{xy}, the xy component of stress

The **linedata_only** keyword indicates that no system matrices should be computed, but the linedata specified in the **linesample** file should be computed (see section 2.13). This is for verification of data transfer. The following is an example solution section for a TSR preload followed by transient dynamics.

```
SOLUTION
   title 'Pure bending from initial stress'
   case tsr
```

```
          tsr_preload
          load 1
      case trn
         transient
         time_step 1.e-6
         start_time 1.0e-3
         nsteps 3
         nskip 1
         load 2
   END
```

If executed on a file with `geometry_file='example.exo'`, this will produce two output files, `example-tsr.exo` and `example-trn.exo`. The first of these has very little useful information. The second contains the displacements (or other variables) from the transient analysis.

Line Sample

One additional feature for thermal structural response is the ability to do line sampling 2.13 on the original **Exodus** file containing the element stresses. This is useful for debugging and verification. It allows the stresses along lines within the structure to be examined. Sampling occurs only for data stored on integration points using variables names described above. Line sample is used for energy deposition (see the *Two Element Exponential Decay Variation Hex20* problem[14]). Energy deposition is interchangeable with supplying an applied temperature.

In tsr_preload, the input **Exodus** file is required to contain at least one of the following fields: stress, temperature or energy deposition. Any field that is not found in the input **Exodus** file is reported as a zero field in the output line sample output file.

2.1.37 Explicit Solver

A preliminary explicit solver is now available in **Sierra/SD** . To use the explicit solver, the keyword `EXPLICIT` must appear within the `SOLUTION` section of the input. Note that the explicit integrator requires a lumped mass matrix, and such a matrix will be generated for this solution. However, in a multicase solution, other procedures may not require a lumped mass. It is strongly recommended that a lumped mass assembly be used for all elements of the solution if an explicit transient is part of the solution. See section 2.2.3.

Parameters for the **explicit** solution procedure are included in Table 18, and the purpose of the keywords are detailed below.

termination_time The solution will end at the termination time. This provides con-

Table 18: Explicit Transient Solution Parameters

Parameter	Argument	Default
termination_time	*Real*	0
time_step_scale_factor	*Real*	0.9999
update_step_interval	*integer*	500
initial_time_step	*Real*	0
time_step_increase_factor	*Real*	1
time_step_estimation	*String*	*global_max_eigenvalue*
nskip	*integer*	1

trol over the total length of the time integration, as the number of steps depends on the current time step, which may change during the analysis. The last step is adjusted to end the simulation precisely at the termination time.

time_step_scale_factor The scale factor provides a means of limiting the stable time step. By default, **Sierra/SD** uses the maximum eigenvalue of the system to compute the stable time step. Note that this is a very accurate measure of stability, and typical analyses should run with a time step somewhat less than the limit.

update_step_interval This determines the frequency at which the stable time step is computed. Computation of the global maximum is relatively expensive and would not normally be performed at every time step.

initial_time_step A user provided initial time step may be provided. It will be honored provided that the stable time step is not less than this initial step.

time_step_increase_factor The increase factor provides a means to gradually change the user provided initial time step. In no case can the time step exceed the stable time step.

time_step_estimation Currently the global maximum eigenvalue is the only means of computing the stable time step.

nskip By default, output is provided in the **Exodus** file at each time step. The **nskip** parameter lets the user reduce the volume of output to an exact multiple of this parameters. Note that there may be aliasing problems if too much data is skipped.

Most of the parameters addressing control of the time step are shown in Figure 12.

An example of input for the explicit solver within a solution context is as follows:

```
SOLUTION
  TITLE = Example of input for explicit solver.
  LUMPED
  EXPLICIT
```

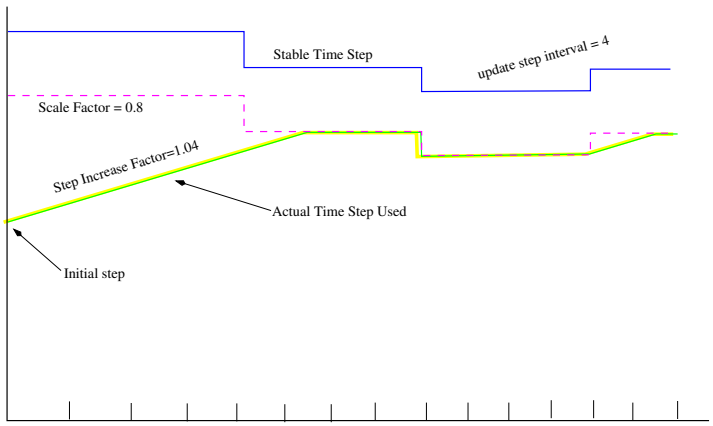


Figure 12: Explicit Time Step Control Example

```
    INITIAL_TIME_STEP = 2.13e-6
    TERMINATION_TIME = 5.0e-3
END
```

2.1.37.1 Explicit Integrator Limitations

In the preliminary implementation force boundary conditions may be applied, but no kinematic boundary conditions are available.

2.1.37.2 Explicit Integrator with Gemini

Time integration with **Sierra/SD** usually uses the standard implicit integrator. The explicit integrator is provided to address performance issues with coupled physics where the other domain may be controlling the time step. One such example is *Gemini*, a fluids code used by the Navy.

2.1.38 Geometric Rigid Body Modes

This section depends on section 2.15.16. In SierraSD, it's possible to use the geometric rigid body modes. There are three examples here. The first example just brings in the rigid modes. The second example uses the modes in solving an eigenvalue problem. The last example uses the Modal Transient capability 2.1.23. The third example uses the modes in a modal transient simulation to deflate out the rotations.

The rigid body modes are requested in the `SOLUTION` block.

```
SOLUTION
   geometric_rigid_body_modes
```

```
END
PARAMETERS
   num_rigid_mode 6
END
```

The rigid body modes can be incorporated into the modes computed in a modal analysis, and then used for other purposes.

```
SOLUTION
   case out
       geometric_rigid_body_modes
   case flexibleModes
       eigen
       nmodes 10
END
PARAMETERS
   num_rigid_mode 6
END
```

The rigid body modes are used as the first 6 eigenvectors. Then **Sierra/SD** computes 4 modes more eigenvectors, for a total of 10.

One use of the geometric rigid body modes is in a q-modal transient simulation to deflate out the rotational rigid body modes, and retain the translational rigid body modes. This is equivalent to use of the `FilterRbmLoad` for direct transient solutions (though accomplished in a very different way).

```
SOLUTION
   case out
       geometric_rigid_body_modes
   case vibration
       eigen
       nmodes 10
       modalfilter rotation
   case transient
       modaltransient
       time_step 1.e-5
       nsteps 62
       load 42
END
PARAMETERS
   num_rigid_mode 6
```

```
END
modalfilter rotation
   add all
   remove 4:6
END
```

2.1.39 Waterline of Rigid Body

It can be very advantageous to determine the waterline of a ship prior to commencing more complex analysis of the body. The **waterline** capability solves the nonlinear geometric equations of equilibrium for a rigid ship in water. An example is shown in Figure 13. Each keyword is described in Table 19.

Keyword	Description
waterline	selects the nonlinear rigid body waterline method
max_iterations	option maximum number of iterations (100)
tolerance_force	optional normalized force balance parameter (1e-6)
VizOption	write vizualization script. default="none".
point_a	coordinates of point 'A' on estimated water surface
point_b	coordinates of point 'B' on estimated water surface
point_c	coordinates of point 'C' on estimated water surface

Table 19: Waterline Parameters

The parameters `point_a`, `point_b` and `point_c` indicate the x, y, and z coordinates of three points on the estimated water surface. These three points define a plane, which serves as the initial guess of the waterline. The waterline normal is determined using the right hand rule with these points as in Figure 14. The Newton's method implementation then uses this plane as the initial guess, and begins iteration towards force and moment equilibrium. On completion, we write out the coordinates of three points on the final (converged) waterline surface, along with the Cartesian coordinate system defined by these points. This output appears in the result file in text format. A "grepos" script for repositioning the body may also be written.

The other parameters control the optimization.

max_iterations sets the maximum number of iterations.

tolerance_force is a normalized force residual. The norm is computed from the residual vector,

$$F_{residual} = [F_z/W, M_{\theta_1}/(LW),\ M_{\theta_2}/(LW)]$$

where $W = Mg$ is the total weight of the ship, and L is a characteristic length of the model.

```
SOLUTION
    case 'waterline'
       waterline
       max_iterations 100
       tolerance_force 1.0e-6 // absolute tolerance on force convergence
       delta = 1.0e-8 // regularization parameter used for Newton step
       point_a 0 0 0 // coordinates of point 'A' on estimated water surface
       point_b 1 0 0 // coordinates of point 'B' on estimated water surface
       point_c 1 1 0 // coordinates of point 'C' on estimated water surface
       load 1

    case 'transient'
...
END

LOAD 1
    sideset 1 // wetted sideset
    pressure = 1
    function = 1 // this defines rho*g*h

    body
       gravity = 0 0 9.8
END

// this assumes rho=1000, g=9.8
FUNCTION 1
    type LINEAR
    data 0.0 0.0
    data 1.0e6 9.8e9
END
```

Figure 13: Waterline Example

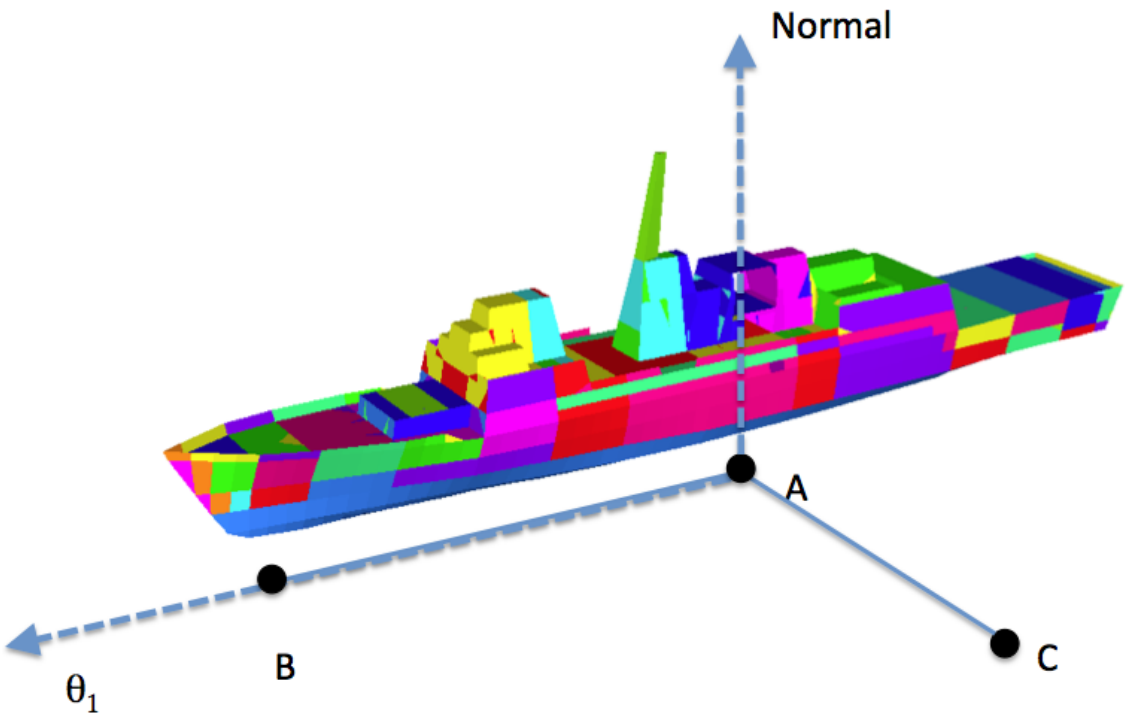


Figure 14: Waterline Coordinate Definition. The plane of the surface is defined by three points: A,B, and C. The θ_1 rotation is about the line from A to B, while the normal is defined using the right hand rule.

delta is currently unused.

VizOption may be "none" or "Ensight" to generate a visualization script.

In addition to the entries in the "solution" section of the input, this method requires two load entries and a function. The load entries define the sideset for the wetted surface and the gravity load.[5] The function defines the pressure as a function of depth. In the example of Figure 13, the argument to the function is the depth, h. The function returns $P = \rho g h$.

The waterline iteration may output nodal data during the iteration. Select "force" to output the buoyancy force. Select "npressure" to output the nodal pressure. See the "outputs" section, 2.9, for details.

2.1.39.1 Limitations:

There are a number of limitations to this method.

gradient based optimization These algorithms are based on a nonlinear gradient base optimization scheme. These are powerful tools, but exhibit limitations that may not be obvious at first blush. Some of these are listed below.

[5] Gravity is specified using the standard load keywords of a body load with a gravity vector. However, for the `waterline` solution only, only the magnitude of the gravity vector is relevant. The gravity direction is *always* directed opposite the normal to the surface for this solution type.

1. Singular tangent matrices are generated in various conditions, which cause the solution to terminate. A very common condition causing a singular tangent matrix is a body completely submerged in a constant density fluid. For simplicity, consider a unrotated cube[6] of edge length "S". The net force on the cube is,

$$F_{net} = (Area_{bottom}P_{bottom} - Area_{top}P_{top}) - mg \quad (28)$$
$$= \rho_f g S^2 (h_{bottom} - h_{top}) - mg \quad (29)$$
$$= \rho_f g S^3 - \rho_s g S^3 \quad (30)$$

where ρ_f and ρ_s are the densities of the fluid and solid respectively. What is significant, is that the net force does not depend on the average depth. Thus, $K_t = \frac{\partial F_{net}}{\partial z} = 0$. Obviously, $K_t = 0$ for a ship that is completely out of the water as well.

Real seawater is not constant density, so one may hope that an optimal solution may be found in this case. However, because the pressure is usually expressed as a piecewise linear function, the same problem occurs. Use of a runtime function may allow computation of higher order derivatives, but this has not been evaluated.

Figure 15 plots net force versus depth for a body. Only the partially submerged region has a nonzero tangent matrix that can be determined by a gradient based optimization scheme.

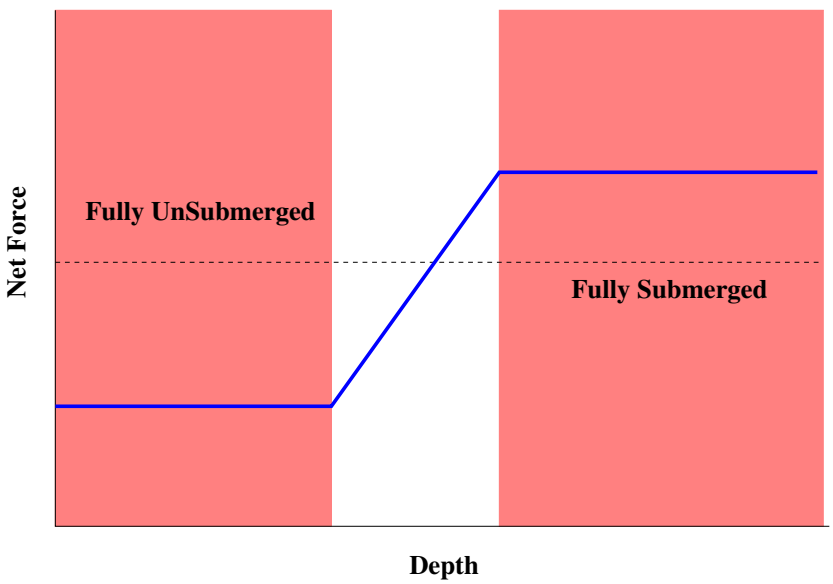


Figure 15: Net Force vs depth for a Rigid Body. Only the unshaded region, where the body is partially submerged, has a nonsingular tangent matrix.

2. Gradient based solution methods often have trouble with local minima. These can occur in the case of unstable systems, such as a very light, tall cylinder floating on a dense fluid. A local minimum occurs for the cylinder standing vertically. A global minimum is achieved when the cylinder is perturbed and falls to the side.

[6] The arithmetic is easier for a cube, but the arguments can be shown to be completely valid for any rigid body.

3. Gradients may also go to zero for symmetry reasons. A perfect cylinder floating on the water has no sensitivity to roll.

sideset orientation: The wetted surface defines the pressure surface. The surface does not need to be closed, but there can be no contribution to the net force from portions of the model that are submerged, but not part of the sideset.

The wetted surface is also defined as a single sideset. The outward direction of the sideset should be pointed into the water. There is no check for a reversal of the normals on the sideset. This must be evaluated by the analyst.

Z orientation: Current design requires that the initial configuration has gravity approximately aligned with the global Z coordinate.

2.1.40 Gap Removal

The removal of gaps in tied surfaces can occasionally result in distorted elements. By default gap removal is turned on for node/face interactions in tied surface (see section 2.20). While the modified mesh (with gaps removed) is typically output as part of the solution, badly deformed elements may make it impossible for the solver to converge; thus, no output is available for the analyst to see the effects of the gap removal.

The **gap_removal** solution method provides a capability whereby a user can see the effects of the gap removal process on **TIED DATA** blocks of interest. The **gap_removal** solution method simply reads in the mesh, applies gap removal to the specified **TIED DATA** blocks (all **TIED DATA** blocks by default), and writes out the new mesh with gap removed. Element quality and a boolean to represent element inversion are output as elemental variables.

An example is given below.

```
SOLUTION
  gap_removal
END

TIED DATA
  surface 1,2
  search_tolerance 1.0e-3
END
```

In this example, the **gap_removal** solution method will apply gap removal to the **TIED DATA** block corresponding to surfaces 1 and 2, and then write out the updated mesh with element quality measure. The suffix "-gap" will be appended to the output exodus file by default when gap removal inverts elements (fatal error). However, setting "ignore-gap-inversion" to true will suppress this output behavior.

We also note that the **gap_removal** solution method is a utility for visualizing the effects of gap removal. Since gap removal is applied by default to each **TIED DATA** block, in all solution methods (see section 2.20), it can at times cause undesirable mesh distortion. If the user suspects that gap removal is causing mesh quality issues, gap removal could be turned off in the **TIED DATA** blocks (see section 2.20), or the **gap_removal** solution method could be used to generate data to visually diagnose the problem.

Since the **gap_removal** solution method runs very quickly, the best strategy is to is to apply it to the **TIED DATA** blocks of interest, one at a time, and visualize each resulting mesh to check the mesh quality.

2.2 Solution Options

The options described in Table 20 and in the following paragraphs are part of the **Solution** section in the input file. None of the keywords are required. Note that in multicase solutions most of these parameters may be applied separately within the subcase (see section 2.1.1.1).

Table 20: **Sierra/SD** Solution Options

Option	**Description**	**Parameters**
restart	restart options	**none**, read, write or auto
lumped	Use lumped mass matrices	none
lumped_consistent	Use a linear combination of lumped and consistent mass matrices.	Real
solver	Identify solver used	`auto`
constraintmethod	method of applying MPCs	Lagrange or Transform
scattering	Separate incident and scattered acoustic loads.	
no_symmetrize_struc_acous	turn off symmetrization for structural acoustics	none
transfer	transfer options	send, receive, or iterate

2.2.1 Restart – option

Option **restart** controls restart file processing. Restart files permit the solution to be saved for later use. Only a limited capability is provided, but it is intended to meet most of the typical needs for structural dynamics. Note that except for eigen restarts, the restart files are independent of the **Exodus** output, but the restart options may significantly affect the **Exodus** outputs. Eigen restarts directly use the **Exodus** output. Application of restarts in specific sections is detailed in the following paragraphs. See Figure 16 for details on eigen restart.

There are four values associated with this option.

none indicates that restart files will be ignored. They will be neither read, nor written. Existing restart files will not be altered in any way. Restart=none is the default selection if no restart options are entered in the solution block.

read indicates that existing restart files will be read, but no output restart files will be written. If the restart files do not exist, a fatal error will result.

write indicates that existing restart files will be ignored, but restart files will be written.

auto is a combination of read and write. However, unlike read, the existence of previous restart files is optional, i.e. there will be no error message if there are no existing restart files. Invalid restart files will produce a warning, but not a fatal error.

Restarts are designed to ensure accuracy of the solution. However, restarts in **Sierra/SD** are not transparent in the sense that there will be small differences in two solutions to a problem when one solution involves a restart. Restarts may also have an expense. For example, the FETI solver uses an acceleration technique where the values of previous solutions are used as a starting place for new solves. The information associated with previous solutions is not stored in the file.

The precision of the **Exodus** *data used to store the results may also affect restarts. For eigen analysis where restarts are expected, it is strongly recommended that higher precision be specified for the* **Exodus** *file output. See section 2.9.35.*

For transient dynamics, the state of the machine at the most recent time step is recorded. To avoid problems with corruption of a database, the three nodal vectors (disp, velocity, acceleration) and applicable element data are recorded at each time step, but on alternate locations in the file. If previous **Exodus** files exist, they will be appended. Data is written at the same interval as the **Exodus** output.

When restarting a multicase solution, the current time is used to determine which case the restart will begin. For example, assume the following solution block is defined.

```
Solution
  case one
    transient
    restart=auto
    time_step 1e-6
    nsteps 200
  case two
    transient
    restart=auto
    time_step 1e-5
    nsteps 300
End
```

If restarting at `TIME=1E-4`, case "one" has a final time value of `T_F = T_0 + 200*1E-6 = 2E-4`, assuming `T_0=0`. Since `TIME < T_F`, case 'one' will restart the solution. If restarting at `TIME=2E-5`, then case 'one' will not perform any calculations. Case 'two' will then be tested to see if a restart will begin there.

An important note about restarts with multicase is that the case names in the restart input deck must match those from the original input deck. This is because the names of the exodus files that are written to disk in restart contain the case name strings. Thus, in the following example with case name "one", both the original and restart input decks must have the same case names. If the original (write) input deck had the following case description

```
Solution
  case one
    transient
    restart=write
    time_step 1e-6
    nsteps 200
End
```

Then the restart (read) input deck would look as follows

```
Solution
  case one
    transient
    restart=read
    time_step 1e-6
    nsteps 200
End
```

where the case name "one" is the same in both input decks.

2.2.1.1 Restart Solution Support. Restarts are not supported in all solutions types. They are supported for the following.

- Eigen
- transient
- nltransient
- modaltransient
- QEVP (with anasazi and sa_eigen methods)

Note that none of the modal solutions except modaltransient support restart. Typically most of the computation time for these solutions is in the eigen analysis. It is recommended that the multicase solution be used, with a restart in the eigen analysis portion of the solution.

Table 21 illustrates the current supported restart options for transient analysis. As shown here, one can restart a **modaltransient** analysis from a **transient** analysis, and vice versa. Restart is not supported for **qmodaltransient**.

Integrator	NLtransient	transient	modaltrans	explicit	qmodaltrans
NLtransient	TESTED	?	?	?	NA
transient	TESTED	TESTED	TESTED	TESTED	NA
modaltrans	TESTED	TESTED	TESTED	TESTED	NA
explicit	TESTED	TESTED	?	TESTED	NA
qmodaltrans	NA	NA	NA	NA	?

Table 21: Supported restart capabilities for transient integrators in Sierra/SD.

The restarts of the QEVP solution are currently limited to only read all of the modes requested or none. Therefore you cannot ask for more modes than are in the restart file and have **Sierra/SD** calculate only the additional modes. You can however save previously calculated modes for use in follow on solution methods.

As of 1/2010, all restart files are in the exodusII format. This permits evaluation and manipulation of the data using standard tools. Details of the file names and formats used for restart are included in Table 22.

2.2.1.2 Restarts with Virtual Nodes and Elements Currently, Sierra-SD creates virtual nodes and elements when a model has features like tied joints, infinite elements, and superelements. These virtual nodes and elements are created internally during execution of the code, and result in the analysis having more degrees of freedom than are specified in the **Exodus** file.

In eigen analysis, starting with release 2.7, no restart file is written – the analysis is restarted based on the **Exodus** output. Thus, the displacements must be written to the **Exodus** output or no restart is possible.
This type of restart has the advantage of simplification of file management (there are no extra files written). Because the files are written in **Exodus** format, it is possible to join the data and re-spread over a new domain decomposition. For example, one could compute modes on 20 processors, join the results and compute the modal based solutions on a single processor. Diagnosis of error is also much easier because the files are in a well understood format.
Reuse of the **Exodus** output does impact the options available as follows.

NONE ensures that no data will be read from existing **Exodus** files. However, *output, written in the normal fashion, will replace existing files.* As a consequence this option no longer ensures that the results of the previous analysis will not be modified.

AUTO Data is optionally read from existing files. They will be updated to include additional calculations. Thus, this option is not changed.

READ now operates *almost* identically to the AUTO option. The only difference is that the program will abort if no restart data is available. This is different from the original design in that the files are updated. Option READ no longer means read-only.

WRITE is unchanged. No restart files are read, but results will be written at the end of the analysis. This option is effectively identical to "NONE".

Figure 16: Notes on Eigen Restart

When the restart files described in Table 22 are written to disk, these virtual elements and nodes will also be written, resulting in **Exodus** files that have more nodes and elements than were in the original **Exodus** file. The restart **Exodus** output files can be visualized in Ensight and the data can be processed in matlab, just like any other output **Exodus** file. Thus, the restart files provide this additional benefit of being able to post-process the virtual nodes and elements. For example, one may wish to visualize the RBARS created for a Tied Joint in Ensight. This can be done by visualizing the restart files.

We note that currently, the infinite element output to the restart file only allows some of the infinite elements to be visualized. This will be corrected in future releases.

2.2.2 Solver

As **Sierra/SD** evolves, various solvers are available for computation of the solution. Each solver brings with it different capabilities and sometimes unwanted features. Currently available solvers are listed in the following.

Solution	**filename**	**Details**
eigen	*example-out.exo*	Use the standard **Exodus** output for restart. Displacements *must* be written or no restart is possible. Other variables (such as strain energy) may also be written.
qevp	*example-out.exo*	Uses standard **Exodus** output for restart. No additional modes may be computed, but the superposition is possible.
transient nltransient explicit	*example-out.rst_trans.exo*	The two most recent time steps are written. They are only written at the "flush" interval.

Table 22: Restart file format and names. Details of the restart files for a file named "example.exo" for various solutions.

AUTO Use the best known solver. Generally this is recommended. The matrix of solvers versus solution types is messy, and generally the best solution will be found by using this option. For example, there is no need to change the solver as you move from serial to parallel solutions.

CLOP This solver is based on a two-level overlapping Schwarz preconditioner with a partition of unity coarse space.[15] CLOP has historically been applied to problems with very large numbers of constraint equations which other solvers could not accommodate. Although no longer under development, the CLOP solver is currently supported by Clark Dohrmann.

GDSW The Generalized Dryja, Smith, Widlund (GDSW) solver is based on a domain decomposition preconditioner which combines overlapping Schwarz and iterative substructuring concepts.[16] Like CLOP, the GDSW solver is well suited to solving problems with large numbers of constraint equations. It has also been observed to be very competitive with other parallel solvers, even for problems with only a small number of constraints. The GDSW solver is currently under development and supported by the Sierra-SD team. The most recent development efforts for the GDSW solver have been focused on implementation and testing of a new Helmholtz solver for direct frequency response analysis.

FETI-DP This solver is the workhorse for parallel solutions. A full description of the solver is beyond the scope of this users manual (references are on the web). FETI-DP was developed by Charbel Farhat, Kendall Pierson and others.[17] It is very scalable, and robust. Multipoint constraints are handled using Lagrange multipliers. The parallel solution process must be used with the solver, but it can be reduced to a single subdomain. Care must be used to ensure that subdomains are mechanism free.

CF_FETI An evolution of FETI-DP, this solver adds the capability to compute nonlinear

constraints within the solver. This is an advanced method of computing gap and contact response. It is a development solver platform, and is not available on all machines. The CF (Charbel Farhat) solver is templated software that supports complex solves as well as real. Thus it can be used for direct frf calculations. The *CF* solver is provided for general availability on most platforms after release 2.0. Further details are available in section 4.

Sparsepak This solver is the workhorse for solutions using a serial executable and is the default solver for single processor runs when using a parallel executable. It is a direct solver, and is part of sparspak developed by Esmond Ng. The solver is fairly robust, but may fail for singular systems. It occasionally has problems for very small systems. Originally written as a Cholesky decomposition, it has been extended to compute LDL^T. Constraints are eliminated using a transformation matrix method. It may be abbreviated as "spak".

SuperLU This package, available from NERSC, provides both real and complex solutions. In **Sierra/SD** , the complex version of SuperLU is the default solver for computing the solution of direct FRFs when using a serial executable or for single processor runs when using a parallel executable.

Generally no user input is required for specification of a solver. Indeed, up to version 1.0.5 of **Sierra/SD** , only one solver was ever available at any time (i.e. we built separate executables if another solver was desired). Usually the specification can be left off, or specified as `auto`. If a solver is requested and unavailable, a warning will be issued, and `auto` will be selected.

The solver may be specified as a default (above the `case` keywords as detailed in section 2.1.1.1), or it may be individually specified within the case framework. The default value is `auto`. In the example shown below FETI-DP will be used for the eigen analysis, FETI-DPC for transient dynamics, and the `auto` selection for the direct frequency response. If "input" is specified in the "echo" section (see section 2.8) then the solver information will be echoed to the results file.

```
SOLUTION
  solver=auto
  case eig
    eigen nmodes=50
    solver=feti-dp
  case nlt
    nltransient
    solver=clop
    (other parameters)
  case frf
    directfrf
 END
```

2.2.3 Lumped – option

Option **lumped** in the **SOLUTION** section causes **Sierra/SD** to use a lumped mass matrix, and not a consistent mass matrix, in the analysis. The method used here is to scale the diagonal terms of the mass matrix so as to ensure the proper total mass, and set the off diagonal terms to zero.

The drilling degrees of freedom associated with beams and shells can generate spurious modes when they are lumped. As a consequence, **Sierra/SD** does NOT fully lump these degrees of freedom. They are lumped in the element coordinate frame, but transforming the mass matrix to the physical coordinates results in a 3x3 entry for the rotations.

Option **lumped_consistent** in the **SOLUTION** section causes **Sierra/SD** to use a linear combination of the lumped and consistent mass matrices in the analysis. A real number α is read in following the **lumped_consistent** keyword. Then, the mass matrix is formed as follows

$$M = \alpha * M_{lumped} + (1 - \alpha) M_{consistent} \tag{31}$$

This modified mass matrix typically gives better dispersion properties than either the lumped or consistent matrices alone.

2.2.4 Constraintmethod – option

The **constraintmethod** option is defined in the **SOLUTION** section to indicate how multipoint constraints (MPC) will be applied. The selections for applying MPCs are are **Lagrange** and **Transform**. These methods are explained in detail on pp. 272-278 in Ref. 18.

The **constraintmethod** is currently superfluous. When using the `FETI` solver, a Lagrange multiplier method is the only method available. When using the serial solvers, the only available method is **Transform**.

2.2.5 Scattering – option

For some acoustics and structural acoustics problems, it is advantageous to define the loads in terms of an incident pressure instead of a total pressure. The solutions for the scattered pressures follow the same differential equations as those of the total pressures. It may be necessary to combine the incident and scattered terms to compute a total pressure. See section 2.3 in the theory manual for details. Note that the **scattering** keyword applies

to *all* loads in the solution case. It is nonsensical to mix scattering pressure inputs with total pressure inputs.

Scattering solutions require this keyword in the solution block. In addition, loads should be applied properly in the LOADS block. The user must apply a load to *both* the structural and the acoustic side of a wet surface. This is typically done using a function tailored for that purpose.[7]

2.2.6 no_symmetrize_struc_acous – option

In structural acoustics problems, the default behavior is to symmetrize the structural acoustic matrices. This is possible by simply multiplying the acoustic equation by a -1. We refer to the theory manual for more details. There are cases where this symmetrization is not possible (such as infinite elements), and in those cases the code internally switches to a non-symmetric formulation.

However, in those cases where symmetrization is possible, the analyst may wish to abandon the symmetrization, and simply revert back to the original non-symmetric system of equations. This may be advantageous from a solver and/or conditioning standpoint, or it may just be of interest to see how the code performs in the non-symmetric case relative to the performance when symmetrization is applied. In these cases, the non-symmetric GDSW solver would be required for the solution. The keyword **no_symmetrize_struc_acous** turns off symmetrization in the code, and forces the solver to solve the nonsymmetric system directly. This keyword can be used in any solution case, and can vary from case to case.

2.2.7 transfer – option

Option **transfer** is used to request a transfer of data to or from another code. Typically this is used for a coupled computation with Gemini. Data hand-off from Sierra/SD will use the "receive_sierra_data" options of section 2.1.29. Transient/transfer may be specified differently for each solution case. Currently, **Sierra/SD** can only be coupled through the Sierra framework using this method. The type of data that is transferred and other control parameters reside in a separate, Sierra input file, which must be used when running these type of coupled analyses. There are three values associated with this option.

send indicates that data will be copied from **Sierra/SD** to another code after the current **Sierra/SD** solution case completes.

receive indicates that data will copied into from another code before the **Sierra/SD** solution case starts. This is identical to having a receive_sierra_data solution case

[7] the "plane_wave", "planar_step_wave" and "shock_wave" functions compute both appropriate pressures on the structure, and normal velocities on the acoustic medium. See sections 2.29.13, 2.29.14 and 2.29.16.

preceding the current one.

iterate indicates that data will be copied into **Sierra/SD** before a solution and out of **Sierra/SD** after the Solution. The solution can then repeated with different sets of transferred input data. This currently only works with implicit or explicit transient solutions in **Sierra/SD** and effectively hands over primary control of time parameters to the coupled code. In other words, the time step and number of steps are chosen by the other code and **Sierra/SD** runs until it gets a termination signal from Sierra. Note that even though **Sierra/SD** does not pick the time step size, it can sub cycle and perform many smaller steps before converging on the coupled code's time step.

2.3 PARAMETERS

This *optional* section provides a way to input parameters that are independent of the solution method or solver. Only one **parameter** section is recognized in each file. The parameters and their meanings are listed below and in Table 23. For reference, Table 24 are parameters occasionally used by developers, but not recommended for common use.

WtMass This variable multiplies all mass and density on the input, and divides out the results on the output. It is provided primarily for the english system of units where the natural units of mass are actually units of force. For example, the density of steel is $0.283\,lbs/in^3$, but "lbs" includes the units of g= 386.4 in/s^2. Using a value of wtmass of 0.00259 (1/386.4), density can be entered as 0.283, the outputs will be in pounds, but the calculations will be performed using the correct mass units.

Sierra/SD , like most finite element codes, does not manage the *units* of the analysis. The selection of a consistent set of units is left to the analyst. For example, if the analyst uses the SI system (Kg,m,s) the units of pressure must be Pascals. Frequencies are reported in Hz. For micromachines these units are quite awkward. It may be better to use units of grams, millimeters and microseconds. The analyst must ensure that all material properties and loads are converted to these units.

Some examples of useful units are shown in Table 25.

NegEigen Unconstrained structures have zero energy modes which may evaluate to small negative numbers due to machine round off. The eigenvalues and associated eigenfrequencies are reported as negative numbers in the results files. However, many post processing tools (such as *ensight*) require non-negative frequencies. By default, **Sierra/SD** converts all negative eigenvalues to near zero values in the output **Exodus** files[8]. To retain the negative eigenvalues in the output file, select parameter **NegEigen**.

[8]Because many postprocessing tools are written for transient dynamics, they expect monotonically increasing, positive values for the time. Since eigenvalues are written in the time columns of the output file, they are converted to be monotonically increasing, positive values. Note that the numerically computed eigen frequencies are also stored as global variables in the file

Keyword	**Arg**	**Default**	**Description**
WtMass	*Real*	1	Mass multiplier
NegEigen	*none*		negative eigenvalue flag
OldBeam	*none*		Beam attribute ordering flag
eig_tol	*Real*	auto	Eigen value tolerance
MaxResidual	*Real*	1	maximum residual for eigen
LinkStiffness	*y/n*	*yes*	Link stiffness for rigid elems
nonlinear_default	*yes/no*	yes	nonlinear element blocks
TangentMethod	*string*	element	method of computing tangent
Info	*int*	1	screen output control
syntax_checking	*int*	2	syntax checking control
SkipMpcTouch	*none*		control of MPCS
condition_limit	*Real*	1	element quality output control
badqual_limit	*Real*	10^{20}	element quality corner control
reorder_rbar	*none*		constraint reordering flag
thermal_time_step	*int*	last	input of thermal data
thermal_exo_var	*string*	TEMP	**Exodus** temperature variable name
energy_time_step	*int*	last	input of energy data
energy_exo_var	*string*	TEMP	**Exodus** energy variable name
FilterRbmLoad	*string*	nofilter	control for filtering of rigid body components in loads
RbmTolerance	*Real*	1e-10	tolerance for rigid body zero
MatrixFloor	*Real*	0	control of matrix fill
MaxMpcEntries	*int*	10^6	maximum # entries in any mpc
Mpc_Scale_Factor	*Real*	varies	multiplier for MPC
patch negative elements	*none*		check and fix element matrices
eigen_norm	*string*	mass	“visualization” or “unit”
constraint_correction	*yes/no*	no	orthogonalize constraints to RBMS
Mfile_format	*string*	sparse_function	control output format for Matlab
RandomNumberGenerator	*string*	“rand”	or “test”
RemoveRedundancy	*yes/no*	yes	automatically remove redundancy
MortarMethod	*string*	dual	dual or standard mortar method
ComplexStress	*yes/no*	no	output complex stress in FRF solutions
num_rigid_mode	*int*	0	number of system rigid body modes
ignore_gap_inversion	*bool*	false	Ignore the change in quality of elements that are caused by gap removal

Table 23: Available keywords in the Parameters section

Keyword	**Arg**	**Default**	**Description**
SharedAcousticStiffness	yes/no	no	share stiffness across processors

Table 24: Developer keywords in the Parameters section

Developer keywords in the Parameters section. These keywords may be used, but their use is discouraged, and they are not fully tested.

Table 25: Some useful combinations of units.

length	mass	time	wtmass	density	force	modulus	intrnl mass
m	Kg	sec	1	Kg/m^3	N	N/m^2 or Pa	Kg
ft	slug	sec	1	slug/ft^3	lbf	lb/ft^2	slug
ft	lbm	sec	1/32.2	lbm/ft^3	lbf	lb/ft^2	slug
in	lbm	sec	1/386.4	lbm/in^3	lbf	psi	lbm/g
mm	μg	μs	1	Kg/m^3	N	MN/m^2 or MPa	μg
mm	g	sec	1	g/mm^3	μN	N/m^2 or Pa	gram
mm	mg	sec	1/1000	g/cm^3	μN	N/m^2 or Pa	gram

OldBeam This option is provided for backwards compatibility with older beam models. Early Patran models using the **Exodus** preference numbered the attributes incorrectly. The first versions of **Sierra/SD** used that numbering. With the new numbering the code had to change. Providing "oldbeam" in the parameters section selects the old numbering. The new numbering will be used by default. At some point in the future, we plan to eliminate this option.

Table 26: Beam Attribute Ordering

Attribute	1	2	3	4	5	6	7
old numbering	area	orientation			I1	I2	J
new numbering	area	I1	I2	J	orientation		

eig_tol This is the tolerance used by **ARPACK** for eigensolution. If not provided, a small value (near machine precision) is used.

MaxResidual This is a tolerance used to check the rigid body mode vectors calculated by `FETI`. If this residual on the rigid body mode vector is larger than this tolerance, **Sierra/SD** will abort. The default value is 1.0.

LinkStiffness This option makes it easier for some solvers to properly compute the response when there are many rigid links. At present, only RBARS and RRODS (see sections 3.38 and 3.37) are affected. The option causes **Sierra/SD** to compute additional stiffness terms that would be associated with a beam (or truss) in place of the rigid element. Since the constraint limits the deformation to zero, there is no affect on the final solution, but the solution process can be significantly simplified since singularities are removed from the stiffness matrix. Specify `LinkStiffness=yes` or `LinkStiffness=no`. The default value is `yes`, which means the additional stiffness terms are used.

nonlinear_default In nonlinear transient dynamics or nonlinear statics, computing the fully nonlinear response of all of the elements in the mesh may be very expensive, and in some cases it is not necessary to do so. For example, for a simulation that only involves Joint2G elements and solid (3D) elements, the analyst may determine that the nonlinear effects of the solid elements are negligible. In such cases, it is advantageous to be able to control the nonlinear response of elements on a block-by-block basis. In section 2.25.2 of this manual, a block-level parameter is described that turns the nonlinearities on and off for individual blocks. In order to avoid having to enter this parameter for each block, the **nonlinear_default** keyword allows the user to set the default for all blocks. If it is set to no, then all blocks default to linear behavior (unless specified otherwise in the BLOCK section), and if it is set to yes, then all elements default to nonlinear behavior. Note that the block-level flags override the **nonlinear_default** keyword. There are two possible cases for this keyword.

nonlinear_default=no All elements default to linear behavior.

nonlinear_default=yes All elements default to nonlinear behavior.

As noted in section 2.25.2 , there are limitations for using linear materials in nonlinear analysis.

TangentMethod The tangent stiffness matrix may be used in a full Newton update in nonlinear statics or transient dynamics (see sections 2.1.27 and 2.1.28). By default, each of the elements can compute it's own tangent stiffness matrix. There are cases (particularly when elements are under development) when it is better to use a tangent matrix computed from finite difference methods. There are three possible values for this keyword.

TangentMethod=element The standard element method.

TangentMethod=difference Use finite difference.

TangentMethod=compare Use the standard method, but also compute the matrix by the difference method. Unless "none" is specified in the ECHO section (2.8), output of the difference of every element matrix in the model will be sent to the results file.[9]

Info **Sierra/SD** outputs many different details to standard out. Most of the details are for the developers. Many such things output are number of processors, and time taken in certain loops. Also in some cases, the contents of an array or other such storage type are output to the screen.

In many cases, this information is not wanted. The "info" option controls the output to standard out. There are four different levels of control. Each level increasingly allows more output to standard out. However, currently only two levels are supported. The other two levels of control will added in the future.

[9] In parallel solutions the results file is written only for the first processor unless the "subdomains" option is specified in the echo section (2.8).

The FETI block option "prt_debug" overrides "info" when it comes to FETI output. In all other cases, "info" takes precedence. If there is no "prt_debug" command in the FETI block, then FETI output levels are also determined by "info."

The four levels of control are:

0. **Silent** – Will only output warnings and std error to the screen
1. **Normal** – Will only output the kind of data most analysts would use
2. **Detailed** – Not currently implemented. Convergence, solution addressing issues.
3. **Debug** – All of the above, plus output deemed important for debugging.

Example of usage:

```
Parameters
  info=0
End
```

This sets the "info" control level to Silent.

syntax_checking **Sierra/SD** has the ability to check input files for syntax and spelling errors. This option controls this behavior. By default a violation is printed to the screen and execution is terminated. If the user wishes, violations can be printed while execution continue, or the checking for violations can be turned off completely.

The three levels of control are:

0. **None** – No syntax checking is performed.
1. **Warnings** – Syntax checking is performed and violations are printed as warnings.
2. **Errors** – Syntax checking is performed and violations terminate execution.

SkipMpcTouch **Sierra/SD** uses a unique method of determining an active degree of freedom set. Unlike codes like Nastran which use an auto-spc method, **Sierra/SD** loops through all elements and activates only degrees of freedom that are required for elements. Multipoint constraints pose a particular problem because some codes (like Nastran) may include multipoint constraints to unused degrees of freedom. Since these are eliminated with the autospc, this poses no problem to these codes, but may confuse **Sierra/SD** significantly. On the other hand, usually degrees of freedom associated with mpcs should be included in the active set, and leaving them out can produce errors.

As a stopgap measure, we provide the parameter **SkipMpcTouch**. If this parameter is set, no degrees of freedom will be activated through multipoint constraints.

condition_limit Element quality checks are important for evaluating the effectiveness of the mesh. By default elements with moderately bad topology are reported. However, sometimes there are so many of these warnings, that the really bad elements may

get missed. The **condition_limit** parameter permits user control of the reporting. Setting this parameter to a larger number will eliminate message from marginal elements. Element checking can also be turned off (see the *elemqualchecks* parameter in the **output** section 2.9.5). The default value is 1.0.

badqual_limit In some meshes, a really bad element can completely dominate the condition of the matrices to be solved. The correct solution is to correct the mesh. However, sometimes it is very difficult to do this, particularly when no solution has yet been found, and identification of the bad element is difficult.

The problem is especially bad for iterative solvers. This option controls the creation of extra corner nodes *in the FETI solver only.* These corner nodes are placed around the offending element which effectively moves that element stiffness into the coarse grid where it is solved by a direct solver. Sometimes this can permit solution of systems which could not otherwise be solved. A list of those elements tagged as "bad" is also printed to stdout and to the .rslt file.

Note that the `badqual_limit` *and the* `condition_limit` *are ignored unless* `elemqualchecks=yes` *has been specified in the* `output` *section (2.9.5).*

reorder_rbar This option allows **RBAR**s to be reordered so that the number of **RBAR**s connected to a single node is minimized. Having a large number connected to the same node results in a highly populated matrix and a slow computation. Therefore, reducing the number of connections can shorten run time.

If redundant **RBAR**s are present (i.e. connections forming a cycle), they are removed.

Specify `reorder_rbar yes` or `reorder_rbar no`. The default value is `yes`, which means **RBAR**s will be reordered.

thermal_time_step For thermal analysis solution procedures (i.e. statics or transient dynamics with a **thermal_load** body load), or for any solution procedure that uses temperature dependent material properties, the temperature distribution of the structure must be read in from the **Exodus** file. Typically, the input **Exodus** files in this case would be the output files from a thermal analysis, and thus would contain the necessary temperature data. Since such an analysis could contain several time steps of temperature data, the parameter **thermal_time_step** allows the analyst to select which set of temperature data is to be read into **Sierra/SD** . The following gives an example.

```
PARAMETERS
  thermal_time_step 10
END
```

In this case the user would be requesting that the temperature data corresponding to the 10th time step be read into **Sierra/SD** .

energy_time_step This variable is identical to the "thermal_time_step" above, but applies to cases where the energy density is input and must be converted to a temperatures. Either energy density or temperature can be input, but not both.

thermal_exo_var Specify the name of the **Exodus** nodal or element variable to use for temperature. These values will be used for temperature dependent material properties as well as applying thermal loads. The default value is 'TEMP', but it can be changed as in this example:

```
PARAMETERS
  thermal_exo_var "DEGREE"
END
```

energy_exo_var This variable is identical to the "thermal_exo_var" above, but applies to cases where the energy density is input and must be converted to a temperatures. Either energy density or temperature can be input, but not both. The only difference is that the energy density will be divided by the specific heat to arrive at the temperature.

```
PARAMETERS
  energy_exo_var "EDEP"
END
```

FilterRbmLoad Establishes a filter for rigid body components of the input load. The options are described in the table. The default value is no filtering. The parameter may need to be used together with the RbmTolerance and solver parameters. We note that the **FilterRbmLoad** parameter is currently only supported for transient and static solution cases. For other solution cases this parameter will have no effect on the solution.

Option	Description
NoFiltering	skip all RBM filtering for the load
AllStructural	apply filtering to all 6 structural RBM
RotationOnly	apply filtering to rigid body rotation only

RbmTolerance Rigid body filters depend upon accurate rigid body modes. The application checks the matrix product of the stiffness matrix to ensure that these vectors are in the null space of the stiffness matrix. If any of the requested vectors are not in the null space, the application terminates. The RbmTolerance provides user control of the threshold for that error. The tolerance is computed as,

$$tolerance = \frac{\|K\Phi_r\|_2}{\|K_d\|_\infty \|\Phi_r\|_2}$$

where $[K]$ is the stiffness matrix and ϕ_r is a rigid body vector. The default is 1e-10.

MatrixFloor Primarily a debugging option. The nearly zero terms in a matrix can be removed using this parameter. Values below this floor are eliminated from the matrix. This can reduce fill, but if used improperly too much of the matrix can be affected. It can be important when running on different platforms, where round off can affect the matrix fill, and make it difficult to compare solutions. This is a relative value, so `1.0E-6` would remove terms in the matrix that are a million times less than the largest term. Default is zero.

MaxMpcEntries Soft limit on the number of mpc entries in any single multipoint constraint. Normally the default will be sufficient, but large RBE3 type entries may exceed this in rare cases. The limit is there to avoid errors reading the input, and because such large constraints can consume memory.

Mpc_Scale_Factor Multipoint constraints equations are arbitrarily scaled. For example, the constraint that two degrees of freedom have the same value could be written as,

$$\begin{aligned} U_1 - U_2 &= 0 \\ &\text{or,} \\ \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} &= 0 \end{aligned}$$

But, the weighting coefficients could just as easily be $[1000 \ -1000]$. The constraint equations are part of the stiffness matrix system, so it makes numerical sense to scale these weights so they produce less round off.

By default, the MPC equations are scaled by $S_{mpc} = (min(K_d) + max(K_d))/2$ where K_d is the diagonal of the stiffness matrix. A user specified value may be set using the **Mpc_Scale_Factor** keyword. A value of `Mpc_Scale_Factor=1` results in no scaling.

patch negative elements Sometimes (for a variety of reasons), the element stiffness matrices generated in **Sierra/SD** may be negative. This is fairly unusual, but in those rare cases where it occurs, it can be very detrimental to the solution. This is usually manifested to the user by a large negative eigenvalue.

This option allows the analyst to request that **Sierra/SD** check every element stiffness matrix for negative eigenvalues. Any found will be reported, and the matrix will be made positive semi-definite.

eigen_norm Eigenvectors may be arbitrarily normalized. Three common approaches are listed in Table 27. All methods retain orthogonality of the eigenvectors, but the normalization differs. The default, mass normalization, is most commonly used as it ensures that the inner products of eigenvectors with the mass matrix is identity. However, this normalization is not well suited to output visualization. The "visualization" normalization mimics what is automatically done in MSC/Patran, and should provide

a reasonable visualization without rescaling each mode. In "visualization" normalization, the maximum *translational* displacement is normalized to be less than 10 percent of the maximum model extent, while also insuring that the model rotation remains below 1 radian. Unit normalization ensures that the largest value of the eigenvector is one. [1] A global variable, *EigenVectScale,* provides the scale factor by which the mode was scaled.

Method	Algorithm	Comment
Mass	$\phi_i^T M \phi_i = 1$	Default. Simplifies numerics
Visualization	$\max(\phi_i)$=(model size)/10	Simplifies visualization
Unit	$\max(\phi_i)$=1	

Table 27: Eigenvector Normalization Methods

constraint_correction Ensure that each multipoint constraint generated is orthogonal to all rigid body modes. This is useful for lofted surfaces. If the surfaces are tied as if they were coincident, the constraints are incorrect, and result in a loss of rigid body modes. See section 5.4 in the theory manual.

```
PARAMETERS
    Constraint_Correction=yes
END
```

MFile_Format Most of our matrix data can be written as Matlab readable files. By default these are written as sparse matrices, as functions. Other formats are also available. The "full" format does not use the sparse methods (and is thus compatible with *Octave* or other tools. Alternatively, the "3column" format can be used. In this format, the file is loaded using the Matlab "load" command. The data is then converted to a sparse matrix using the Matlab "sparse" command. The "3column" format may be significantly faster in some cases, but it does require more user interaction. Figure 17 compares a simple example for the three formats. In all cases, the matrix symmetry is the same. A fourth format, "CSV", is also available for compatibility with other external tools.[2]

RemoveRedundancy Redundant constraints cause most solvers to fail. Redundant constraints are often introduced when two surface pairs are tied next to each other, but

[1] The "unit" method of normalization computes $\max(\phi)$, which is computed *only* on translational displacement degrees of freedom. Note also that only displacements are renormalized. No effort is made to renormalize element variables such as strains, stresses or energies. Thus, if these are requested in an eigen analysis, they will not be consistent with the renormalized eigenvectors, but will retain mass normalized values.

[2] Note that the CSV format should be readable by Microsoft Excel, but there are often limits on the number of columns that can be read.

Sparse_Function	Full	3column

function s=Kssr()
s=[1 1 0.11
1 2 0.12
2 2 0.22];
s=sparse(s(:,1),s(:,2),s(:,3));
	function
	s=Kssr()
	s=zeros(2,2);
	s(1,1)=0.11;
	s(1,2)=0.12;
	s(2,2)=0.22;
		1 1 0.11
		1 2 0.12
		2 2 0.22

Figure 17: Example MFile Format Results

there are a variety of sources for these redundancies. Exact redundancies are always automatically eliminated, but that is often not sufficient. This parameter removes constraints when a node is applied as more than one slave relation, or if the node is applied both as a slave and as a master. By default it is "true".

RandomNumberGenerator The default random number generator, "rand", is the standard generator available from system libraries. It should be the best random number generator in terms of the quality implementation. In a few cases the analyst may want a more repeatable random number generator, that is independent of the platform. The "test" random number generator can be used in this case. It is *not* recommended for general use, and the statistics of the generator are not well established.

MortarMethods Two mortar methods are available in **Sierra/SD** : standard and dual (see 19). By default the dual method is selected as it is almost always more efficient in memory use.

ComplexStress Most often, analysts do not want output of stress variables in frequency response function analysis. Such output is complex, and huge volumes can be generated. Selecting "ComplexStress=yes", along with "stress" in the echo section permits output of this data. The default is "ComplexStress=no".

num_rigid_mode Is used to signal to the linear solver that the system is singular and that the singularity is associated with structural and/or acoustic rigid body modes. This is used, for example, in the solution of statics problems without any essential boundary conditions or frequency response analysis with the modal acceleration method. Where possible, other methods should be used to eliminate the singularity. For example, in eigen analysis a negative shift is recommended. Currently allowed values for this parameter are 1 (acoustic mode only), 6 (structural modes only), or 7 (structural and acoustic modes). We also note that when using the **FilterRbmLoad** parameter, it is necessary to specify **num_rigid_mode** to correspond to the number of rigid body modes that will be filtered. For example, if **FilterRbmLoad** was set to **AllStructural**, then **num_rigid_mode** should be set to 6.

ignore_gap_inversion During the gap removal process the element quality can be affected. If the element quality is affected enough the test will error out. To ignore the re-

duction of element quality caused by gap removal the parameter **ignore_gap_inversion** can be utilized. The default is set to false.

UseAnalystNodeMap The exodus database which contains all the topological information in a finite element model is based on a 1 to N ordering of nodes, which provide the connectivity to the elements in the model. This is often referred to as a "*serial node map*." Many meshing tools generate a mesh with an arbitrary ordering of nodes. In some cases, the analyst may want to control that ordering of the nodes to help in identifying particular nodes. If possible, a nodeset should be used for that, but there are cases where output locations, or other quantities are identified with specific nodes. That arbitrary, analyst controlled ordering is known as an "*analyst node map*". Visualization tools typically display the mesh using the analyst node map, and error messages from **Sierra/SD** report issues with respect to that map. Because internally the connectivity is expressed in terms of the serial map, there can be situations where it is convenient to turn off the analyst map. This can be done with the command,

```
UseAnalystNodeMap=false
```

This is mostly useful for debugging purposes.

2.4 FETI

This *optional* section provides a way to input parameters specific to the Finite Element Tearing and Interconnecting[17] (FETI) solver, if used. If the FETI solver is not used, this section is ignored. It includes the following parameters, shown in Table 28, and options. For those options which are strings, only enough of the string to identify the value is required. The defaults are shown in the following example.

```
FETI
      rbm  geometric
      scaling no
      preconditioner dirichlet
      max_iter 400
      solver_tol 1.0e-5
      orthog 1000
      rbm_tol_svd 1.0e-10
      rbm_tol_mech 1.0e-8
      projector standard          // ignored in dp
      level 1                     // ignored in dp
      local_solver sparse
      coarse_solver sparse
      grbm_tol 1e-6
      prt_summary yes
      prt_rbm yes
```

Table 28: FETI Section Options

Variable	Values	Description
rbm	Algebraic/Geometric	rigid body mode method
scaling	Yes/No	scaling method
preconditioner	LUMped/DIRichlet	(both may be used)
max_iter	*Integer*	maximum number iterations
solver_tol	*Real*	
stag_tol	*Real*	Used to detect stagnation
orthog	*Integer*	max number of orthog. vectors
rbm_tol_svd	*Real*	SVD tolerance in rigid body modes
rbm_tol_mech	*Real*	mechanical tolerance in rbm
projector	**Standard**/Q	projector
level	1	feti1 (feti2 not implemented)
corner_dimensionality	*Integer*	3 or **6** dofs/corner
corner_algorithm	*Integer*	1, 3, 5-8
corner_augmentation	*String*	**"none"**, "subdomain", "edge"
local_solver	AUto, SKyline, SParse, single_sparse	solver for local LU decomp
precondition_solver	Same as local_solver	solver for preconditioner Only used if using Dirichlet preconditioner
coarse_solver	AUto, SKyline, SParse PSparse, single_skyline, single_sparse, iterative	solver for coarse G^TG problem (psparse is parallel sparse)
grbm_tol	*Real*	tolerance for rigid body detection in G^TG
prt_summary	Yes/**No**	print summary timer information
prt_rbm	Yes/**No**	print # rbm in each subdomain
prt_debug	integer	debug output. values **0=none**, 1-3
bailout		if set, the solver will continue even if the solution is not converged at each intermediate solve
mpc_method	*Integer*	**0**=Lagrange multipliers everywhere 1=Local elimination where possible

```
        prt_debug 3
        corner_dimensionality 6      // for dp only
        corner_algorithm 3           // for dp only
        corner_augmentation none     // for dp only
END
```

2.4.1 Corner Algorithms

Corner selection is an important issue (and an ongoing research area) for **FETI-DP**. Several algorithms are available. They all vary by the total number of corners picked in the model for the coarse problem. The various algorithms are intended to give a little more power to the advanced user. The more corners that are picked, the quicker the solution will converge. The disadvantage being that there might not be enough memory available for these corners, hence, **Sierra/SD** might abort because of this memory depletion. Memory statistics can be observed and with experience, the advanced user can pick the optimal corner algorithm. The possible choices for the various parameters are given in Table 29. All the options for each corner parameter are listed such that the first option for each parameter picks the least amount of corners.

Typically, corner algorithm 15 selects the minimal number of corner points. This is a useful option to try if memory becomes an issue when running on large numbers of processors. As noted above, smaller coarse grids increase the number of iterations to convergence.

Corner algorithm 14 selects three corners between along the interface between two neighboring subdomains (Γ_{ij} designates the interface between subdomain i and subdomain j). The first node is selected as the node along Γ_{ij} that touches the most subdomains. The second node is the node that maximizes the distance between any two nodes along Γ_{ij}. The third node is selected to maximize the triangular area created by three non-collinear nodes along Γ_{ij}. Corner algorithm 14 will typically select less corner nodes than Corner algorithm 3.

> *Note that additional corner nodes can be placed in a special file,* `extraNodes.dat`*. Nodes in this file will be added to the current corner selection algorithm. While this method is seldom useful, it can help in cases where an isolated element is causing catastrophic problems. The format of* `extraNodes.dat` *is to simply put the* **global** *node numbers, one per line, in the file.*

2.4.2 Solves within Solves

The FETI algorithm employs three linear solves as part of the iterative solution strategies. Each of these solves consumes memory and resources.

Table 29: Corner Options

Parameter	Option	Description
Algorithm	0	Picks 1 corner per interface
Algorithm	1	Most robust algorithm
Algorithm	2	Picks 2 corners per interface
Algorithm	3	Picks 3 corners per interface
Algorithm	9	Picks all interface nodes *debug only*
Algorithm	14	Improved version of Corner Algorithm 3
Algorithm	15	Improved version of Corner Algorithm 0
Algorithm	16	No automatic corners. (uses `extraNodes.dat`).
Algorithm	17	like 3, but add corners for conms
Algorithm	99	like 14, but add will not pick corners on mpcs
Dimensionality	3	Fixes 3 translational d.o.f. per corner
Dimensionality	6	Fixes *all* d.o.f. per corner
Augmentation	none	no additional corners are selected
Augmentation	edge	Additional corners on interface edges are selected. (Stiffness weighted).
Augmentation	subdomain	Additional corners per subdomain are selected.

local. The local solve is a complete factorization of each subdomain independently. This factorization requires that the subdomain be properly connected so the subdomain stiffness matrix is nonsingular. Failure at this level results in a reported zero energy mode (or ZEM), which causes the solver to fail. The size of this problem depends only on the subdomain size, so increasing the number of subdomains decreases the local solve memory.

preconditioner. This is the least important of the solves, and seldom affects either the robustness or memory of the solve.

coarse. The algorithm constructs a coarse solution space from the interface degrees of freedom. This solution has the properties of the original problem including rigid body modes if they exist in the global problem. This solve will fail if singular, which can occur if there is no negative shift, or if there are mechanisms in the original global problem. The coarse problem size increases with the number of subdomains, and depends upon the corner algorithm selected.

Various options are available for the solution to these sub-problems. These are listed in Table 30.

Skyline. The most robust and oldest of the solvers, this method is also usually the slowest, and it often uses the most memory.

Sparse. This is the workhorse solver, and should be the baseline for any study.

Single_skyline. A reduced accuracy solution (using single precision arithmetic), it may be used in cases of limited memory.

Single_sparse. A sparse single precision solution. It may have the least memory footprint.

PSparse. Generally the coarse problem is solved redundantly on each processor. This is both faster and more robust than parallel solutions. The *psparse* option allows for a parallel solution to the coarse problem. It may be faster and *may* use less memory than the *sparse* method when the coarse problem size increases. It is typically not recommended for solutions using less than 1000 processors as it is not as robust as the sparse method, and may actually use more resources.[a]

[a] The interface for psparse is still under development. Currently you can set the number of processors to use in the parallel direct solution by creating a file in the current directory named "psparse_params". The file must contain a line like the following.

```
nproc 4
```

Where in this case we set the number to 4. Note that this interface is expected to change in the future.

Table 30: Linear Solver Options

2.4.3 Levels of Diagnostic Output

The **prt_debug** flag takes various values from 0-4. Table 31 shows the various values and their result. Note, for **prt_debug** value of 3, a file named *corner.data* is written. The format is as follows:

```
Ncorners
GlobalId LocalId SubdomainId Xpos Ypos Zpos
.
.
.
GlobalId LocalId SubdomainId Xpos Ypos Zpos
```

Ncorners is the total number of corners, *GlobalId* is the global id of the corner, followed by

the local id (*LocalId*), the subdomain on which the corner exists (*SubdomainId*), and the coordinates of the corner (*Xpos Ypos Zpos*).

Table 31: Prt_Debug Options

prt_debug value	Result
0	No Output
1	Some Output
2	Lot of Output
3	Output + Corner.data file
4	Output + Corner.data file + Matlab files

Other parameters that affect diagnostic output include the following.

prt_summary If *Yes,* summary performance information is reported to `stdout`.

prt_rbm if *Yes*, the number of zero energy modes determined on each subdomain will be reported to `stdout`. If *No,* then only subdomains with a nonzero number of ZEMS are reported.

2.5 CLOP

The CLOP solver may be specified in the solver section (see section 2.2.2). Parameters for the **CLOP** solver can be specified in an *optional* "clop" section.[1] Parameters are listed in table 32. An example follows.

```
CLOP
  max_iter=1000
  solver_tol=1e-5
  orthog=200
  prt_summary=1
  prt_debug=0
  overlap=1
END
```

Comments:

The "orthog" option can be very memory intensive, and caution is advised when setting this to a value above 200. Krylov_method 7 uses left preconditioned GMRES and is an

[1] *Note that the "CLOP" section only specifies the linear solver parameters. The "solver=clop" specification is required in the solution section.*

Table 32: CLOP Section Options

Variable	**Values**	**Dflt**	**Description**
max_iter	*integer*	400	maximum number of iterations
solver_tol	*real*	1e-6	relative residual convergence tolerance
krylov_method	*integer*	0	0 - PCG, 1 - right preconditioned GMRES, 7 - left preconditioned GMRES *(an option for structural acoustics)*
overlap	*integer*	0	number of layers of overlapping elements for preconditioner
orthog	*integer*	200	number of stored search directions *(caution setting this above 200)*
scale_option		0	0 - no scaling in factorizations 1 - use scaling in factorizations
prt_summary	*integer*	1	output flag: 0 - no summary 1 - basic summary 2 - basic summary + condition # estimates
prt_debug	*integer*	0	0 - no debug output 1 - basic debug output
bailout	*keyword*		If keyword is found, we try to complete the solve even if errors are found.
coarse_solver	direct 3level		solver for coarse G^TG problem
stag_tol	*real*		Used to detect stagnation
local_solver	*integer*		solver for local LU decomp

option for structural acoustics. Left preconditioning attempts to scale the acoustic and structural unknowns appropriately, but this scaling can be sensitive to the conditioning of the system matrix.

2.6 GDSW

The GDSW solver is presently the default solver that is used whenever parallel jobs are executed, i.e. on two or more processors. The older version of the solver can still be accessed using the *version* keyword (see Table 34), but users are advised to use the current version whenever possible since the older one will be eventually phased out. Please report any problems using the GDSW solver with the default solver parameters to the Sierra help system at `sierra-help@sandia.gov`.

Parameters for the GDSW solver can be specified in an *optional* "GDSW" section.[2] Alternatively, solver options may also be specified using the "solver_options" keyword with a corresponding "solver_options" section (see 2.7). These parameters are listed in Tables 33-36, and detailed descriptions of some of them are provided below. Table 33 describes the basic solver parameters, while those for advanced usage are given in Table 34. Parameters for supplemental output useful for diagnostic purposes are described in Table 35. The GDSW Helmholtz solver is a relatively new capability, and the relevant solver parameters are given in Table 36. Please report any problems using the new Helmholtz solver to `sierra-help@sandia.gov`. For convenience, parameters and defaults specific to the older version of the GDSW solver are provided in Appendix 5.

Table 33: GDSW Section Options (Basic)

Variable	**Values**	**Dflt**	**Description**
max_iter	*integer*	1000	maximum number of iterations
solver_tol	*real*	1e-6	relative residual convergence tolerance
overlap	*integer*	2	number of layers of overlapping nodes for preconditioner
orthog	*integer*	1000	number of stored search directions used to accelerate solver convergence
prt_summary	*integer*	3	output flag: 0 - no summary 1 - basic summary 3 - more detailed summary

solver_tol It is very important to control the accuracy of the solution. For all our linear solvers, *solver_tol* is the requested accuracy of the computed solution as measured by

[2] *Note that the "GDSW" section only specifies the linear solver parameters. The "solver=GDSW" specification is required in the solution section.*

Table 34: GDSW Section Options (Advanced)

Variable	**Values**	**Dflt**	**Description**
version	*integer*	2	GDSW version (1 for older version)
krylov_method	*integer*	1	0-pcg: preconditioned conjugate gradients 1-GMRES: right preconditioned GMRES (generalized conjugate residual version) 2-lGMRES: left preconditioned GMRES 3-flexGMRES: flexible right precond GMRES 4-flexGMRES2: variant of FLEXGMRES 5-GMRESClassic: right preconditioned GMRES (classic version)
default_solver	*integer*	1	1-direct: Esmond Ng's sparse direct solver 3 - Pardiso for Pardiso sparse direct solver (only available with Intel MKL) 6-NoPivot: Clark's templated sparse direct solver
num_rigid_mode			note: see **parameters**, section 2.3.
max_numterm_C1	*integer*	100	maximum # of terms for Type 1 constraints
coarse_option	*integer*	1	0-additive: additive coarse correction, 1-multiplicative: multiplicative " "
SC_option	*integer*	1	0-no/1-yes: eliminate subdomain interior unknowns using static condensation
weight_option	*integer*	2	1 - to not use weighted residuals for overlapping subdomain problems
coarse_size	*string*	auto	coarse space reduction option auto: automatic selection small: use reduced coarse space large: use larger coarse space
reorder_method	*string*	metis_edge	metis, metis_edge, rcm, minimum_degree, none
num_GS_steps	*integer*	1	number of Gram-Schmidt orthogonalization steps for stored search directions
con_tolerance	*real*	1e-10	singularity tolerance for processing constraints
con_row_tolerance	*real*	1e-1	pivoting tolerance for processing constrains
scale_option		0	0 - no scaling in factorizations 1 - use scaling in factorizations
diag_scaling	*string*	none	none - no scaling of operator matrix diagonal - symmetric diagonal scaling
PTAP_solver	*integer*	1	solver for conjugate gradient matrix 0-diag: diagonal (holds in exact arithmetic) 1-full: full $\Phi^T A \Phi$ matrix
bailout	*keyword*		If keyword is found, ignore errors
atLeastOneIteration	*integer*	0	0-no/1-yes Iterate once at least
coarsening_ratio	*integer*	1000	coarsening ratio for multilevel solver
minCoarseLevels	*integer*	1	min number of coarse levels (for testing only)
maxCoarseLevels	*integer*	1	max number of coarse levels
maxCoarseSize	*integer*	3000	max size for coarsest problem
graphPartitioner	*integer*	0	graph partitioner for multilevel solver 0-Parmetis, 1 PHG in Zoltan

Table 35: GDSW Section Options (Supplemental Output)

Variable	Values	Dflt	Description
prt_coarse	*integer*	0	0-no/1-yes: print coarse matrix
prt_constraint	*integer*	0	0-no/1-yes: print constraint matrix
prt_memory	*integer*	0	0-no/1-yes: print memory information
prt_timing	*integer*	0	0-no/1-yes: print timing information
prt_interior	*integer*	0	0-no/1-yes: print interior matrices
prt_overlap	*integer*	0	0-no/1-yes: print overlap matrices
write_orthog_data	*integer*	0	0-no/1-yes: write orthogonalization data to file

the relative residual error. In other words, the 2-norm of the residual vector for the computed solution divided by the 2-norm of the right-hand-side force vector should be no greater than *solver_tol*.

orthog One useful feature of both the FETI-DP and GDSW solvers is the ability to accelerate convergence of their iterative methods by using stored search directions from previous solves. This feature requires additional memory, but may significantly reduce iterations. However, in some cases, the application of these vectors can lead to numerical instabilities caused by loss of orthogonality. We recommend setting orthog=0 as an early step in diagnosing any solver convergence problems.

krylov_method A variety of Krylov iterative methods are available as shown in Table 34, but the default should work fine in most instances. If convergence problems arise, we recommend switching to classic right preconditioned GMRES (krylov_method = GMRESClassic) without the use of any stored search directions (orthog = 0).

num_rigid_mode note: see **parameters**, section 2.3. This keyword should not appear in the GDSW solver block but rather the Parameters block.

max_numterm_C1 Constraints for the GDSW solver are classified by two types:

Type 1: simple constraints like those applied by an RBAR, tied contact, or rigid surfaces.

Type 2: more complex, averaging constraints like those in an RBE3.

Type 1 constraints typically have a smaller number of terms, whereas Type 2 constraints may involve many terms in a single constraint equation. Solution of problems with Type 2 constraints using Type 1 methods is possible and desirable if they are small enough, but the memory requirements could be prohibitive if the number of terms N in any constraint equation is too large. Specifically, storage of a dense matrix with at least N^2 terms would likely be required. The parameter *max_numterm_C1* specifies the maximum number of terms that can appear in a Type 1 constraint following a constraint pre-processing step. Constraints with more than *max_numterm_C1* terms are then considered to be Type 2. The algorithm used to enforce Type 2 constraints in the preconditioner is generally not as efficient as the one for Type 1 constraints.

coarse_size Is used to specify a reduction strategy for the coarse problem size. There is no need to consider this parameter for problems run on less than a few hundred processors. However, as the number of processors (subdomains) becomes large, solving the coarse problem can become a bottleneck. The default (*auto*) automatically selects to use the *small* coarse space only if the number of processors exceeds 1000. Specifying a *small* rather than a *large* coarse space can often reduce the amount of memory needed by the solver.

reorder_method Allows one to specify a reordering method for a sparse direct solver. Currently it is only available for default_solver = direct (see Table 34).

num_GS_steps Is used in conjunction with the use of stored search directions. Its default value is 1 (one orthogonalization step), but it can also be set to 2 to reduce the loss of orthogonality of stored directions.

con_tolerance The GDSW solver uses a sparse LU decomposition algorithm to process the constraint equations. This involves choosing pivot rows for numerical stability (much like Gaussian elimination with partial pivoting). A constraint equation is deemed linearly dependent if the magnitude of its pivot is less than con_tolerance.[3] The number of numerically redundant constraints in a model will typically be reduced as the `con_tolerance` is increased.

Messages of the form,

```
min/max pivot for constraint factorization = some number
You may want to consider increasing the con_tolerance
parameter in the GDSW solver block.
```

are issued if the ratio of magnitudes of the smallest to largest pivots is less than 0.01. This provides a recommendation to carefully examine the constraints in the model for any potential problems.

scale_option There are presently two options for matrix scaling in the GDSW solver. Including `scale_option yes` or, equivalently, `scale_option 1` in the GDSW solver block will apply symmetric diagonal scaling to all matrices prior to them being passed to Esmond Ng's sparse direct solver. Notice for parallel runs that both the subdomain matrices and the coarse problem matrix will be scaled. In exact arithmetic, this option should have no effect on the number of iterations for each solve of a parallel run.

diag_scaling Including `diag_scaling diagonal` in the solver block will apply symmetric diagonal scaling to the original operator matrix and is not tied to a specific sparse direct solver. In contrast to the scale_option parameter, this parameter can have an effect on the number of iterations for each solve of a parallel run since GDSW is now solving the scaled problem $DADy = Db$ to a specified relative residual tolerance rather than the original problem $Ax = b$ (note substitution of $x = Dy$, where D is a diagonal scaling matrix) for that same tolerance.

[3] The constraints are normalized so that con_tolerance can be viewed as a dimensionless parameter.

coarsening_ratio Is a target ratio between the number of subdomains prior to and after coarsening by the multilevel solver. For example, if there are originally 8000 subdomains (processors) and `coarsening_ratio` is chosen as 100, then the number of subdomains after coarsening will be 80.

maxCoarseLevels Is the maximum number of coarse levels allowed by the multilevel solver. For a standard 2-level method this parameter has a value of 1.

maxCoarseSize Is the largest size for the coarsest problem allowed before another level of coarsening is made. The solver parameter `maxCoarseLevels` takes precedence over `maxCoarseSize`.

graphPartitioner Specifies which graph partitioning software to use when coarsening the subdomains.

Additional details and troubleshooting strategies for the GDSW solver can be found in documentation available on the `compsim.sandia.gov` website. Relevant documentation includes GDSW 101 and the GDSW Solver Tutorial. We note that solver strategies for dealing with poor mesh decompositions caused by the presence of constraints equations or multiple physics (i.e. structural-acoustics problems) are describe in the GDSW Solver Tutorial. These include rebalancing algorithms internal to the solver that can be accessed using GDSW solver parameters. We hope this will provide a useful interim solution for challenging problems prior to the deployment of alternative decomposition tools that effectively address these issues prior to the solution phase.

Table 36: GDSW Section Options (Helmholtz)

Variable	**Values**	**Dflt**	**Description**
Hprecond	*integer*	5	Helmholtz preconditioner: 0-stiffness: Stiffness based 1-LG: Laird-Giles 2-custom: Custom 3-SL: shifted Lapalacian 5-operator: Operator with damping
orthogH	*integer*	20	maximum number of stored search directions for Helmholtz problems
max_previous_solutions	*integer*	0	maximum number of previous solutions used to accelerate convergence
precondUpdateFreq	*integer*	10	frequency to update preconditioner as as operator changes
viscous_damping	*real*	0	viscous damping coefficient (see text)
structural_damping	*real*	0.12	structural damping coefficient (see text)
alphaK	*real*	0	custom precond stiffness coefficient (see text)
betaK	*real*	0	custom precond stiffness coefficient (see text)
alphaM	*real*	0	custom precond mass coefficient (see text)
betaM	*real*	0	custom precond mass coefficient (see text)
krylov_methodH	*integer*	5	same as krylov_method but for Helmholtz problems
SC_optionH	*integer*	0	same as SC_option but for Helmholtz problems

The *custom* option for *Hprecond* in Table 36 preconditions the matrix

$$-\omega^2(\alpha_M + i\beta_M)M + i\omega C + (\alpha_K + i\beta_K)K,$$

where ω is the circular frequency of excitation, i is the imaginary unit, and M, C and K are the mass, damping, and stiffness matrices, respectively. With $\gamma = \mathit{structural_damping}$ and $\beta = \mathit{viscous_damping}$ the non-zero parameters for the other preconditioning options are $\alpha_K = 1$ for *stiffness*, $\alpha_K = 1$, $\alpha_M = -1$ for *Laird-Giles*, $\alpha_K = 1$, $\alpha_M = 1$, $\beta_M = -\gamma$ for *shifted Laplacian*, and $\alpha_K = 1$, $\beta_K = \gamma + \beta\omega$, $\alpha_M = 1$ for *operator*. Notice one should not use the *stiffness* preconditioning option for ω near zero for structures with rigid body modes since K is singular or near singular in this case.

2.7 Solver_Options

Through a multicase solution it may be useful to specify a different set of linear solver parameters in various steps of the solution. For example, an eigen analysis may use a shift parameter to eliminate rigid body modes in a solution, while a statics analysis cannot do this. The default "GDSW" block only allows for a single set of parameters in an analysis.

The **solver_options** sections allow multiple definitions of these parameters. Each "solver_options" section may be called out from a separate multicase section of the solution block. See Figure 18 for an example. At this time, these sections apply *only* to the GDSW solver.

```
Solution
    case preload
        statics
        solver_options for_preload
        load=10
    case eig
        eigen
        nmodes=100
        solver_options for_eig
End
Solver_options for_preload
    solver_tol = 1e-4
    constrain_rbms="x y z"
End
Solver_options for_eig
    solver_tol = 1e-8
End
```

Figure 18: Multiple Solver Options Example

2.8 ECHO

Results, in ASCII format, from the various intermediate calculations may be output to a results file, e.g. *example.rslt*, where the filename is generated by taking the basename of the text input file (without the extension) and adding *.rslt* as an extension. Output to the results file is selected in the **Sierra/SD** input file using the **ECHO** section. An example is given below, and the interpretation of these keywords is shown in Table 37.

```
echo
    materials
    elements
```

```
    jacobian
    all_jacobians
    timing
    mesh
    echo
    input
    nodes
    mpc
end
```

Note that if **none** is used, the order of selection is important. Thus, if you add **none** at the end of the list, no output will be provided in the echo file. However, if you put **none nodes** then only nodal summary information will be included. Entering **nodes none mesh** only outputs the mesh information (**nodes** information is canceled by the **none**).

We remark that for virtual blocks, element variables such as element force are also written to the results file. Since only Joint2G elements are currently supported as virtual blocks, the only element variable that can be written at this time is the element force, **eforce**.

2.8.1 Mass Properties

The mass properties may only be reported in the **echo** section (i.e. at this time there is no mass property report in the **outputs** section). The mass properties reports the total mass, the center of gravity and the moments of inertia of the system. All are reported in the basic coordinate system. Note that moments are about the origins, not about the center of gravity. Masses are reported in a unit system consistent with the input, whether or not the **WtMass** parameter has been used (see section 2.3).

An additional option of **block** may be used in the echo section to output the block wise mass properties to the results file. Please note that the block wise mass properties, though summed for all processors (if running on a parallel machine), are only output to the result file from the first processor (processor 0). The block wise mass properties option, called **block**, reports the number of blocks, the mass of each block, and the center of gravity of each block along the x, y, and z axis. Please note that **block** may only be used in the **ECHO** section just following the **mass** option as shown below.

```
echo
    materials
    elements
    mass=block
    nodes
end
```

Table 37: ECHO Section Options

Option	Description
acceleration	nodal accelerations (better in output section)
debug	debug output
all_jacobians	jacobians for every element
block	block wise mass properties (used only following mass)
displacement	nodal displacements (better in output section)
echo	dumb echo of input *(for parse errors)*
eforce	element force for beams
elements	element block info, i.e. what material, element type, etc
ElemEigChecks	element eigenvalues
energy	element strain energy and strain energy density
eorient	element orientation
feti_input	
force	applied forces (better in output section)
genergies	global kinetic and strain energy sums
input	summaries of many sections
jacobian	block summary of jacobians
kdiag	diagonal of stiffness matrix
adiag	diagonal of dynamics matrix
mass	mass properties in the basic coordinate system
materials	material property info, e.g. E, G
mesh	summary of data from the input **Exodus** file
mesh_error	mesh discretization error metrics
NLresiduals	turns on residual output per iteration of the Newton loop for non-linear solution methods
nodes	nodal summary
rhs	Right Hand Side vector (better in output section)
subdomains "0:3:6,10"	Controls which processor will output results file
modalvars	modal force and amplitude for modal solutions (echo section)
timing	timing and memory information
memusage	prints per processor per task memory use to a text file
velocity	nodal velocities (better in output section)
residuals	residual vectors
tindex	control time axis index
strain	element strains at centroids
stress	element stresses at centroids
vonmises	von mises stress only
vmrs	RMS quantities (random vibration only)
mpc	mpc equations
all	everything
none	nothing

If the keyword **mass** does not directly precede **block** in the **ECHO** section, then **Sierra/SD** will abort with the following error.

```
Unrecognized "echo" option 'block".
Aborting.
```

Finally, we note that if the user requests both **mass** and **mass=block**, then only the global mass properties will be written to the result file. If only block-level mass properties are desired, then it is only necessary to have the **mass=block** specified, as follows

```
echo
    mass=block
end
```

2.8.2 Mpc

The keyword **mpc** instructs the code to write out the mpc equations to the result file. This is a good tool for debugging purposes, as well as a check on the input deck. An example of the output format is as follows

```
MPC
  coordinate 0
    25 P 1
   106 P -1
  // G = 0.000000
  // the source is global
END
```

In this case, the mpc equation is constraining the acoustic pressure in nodes 25 and 106 to be equal in the global (default) coordinate system.

2.8.3 ModalVars

The **ModalVars** keyword generates text output containing modal forces and modal amplitude for modal based superposition solutions including "modaltrans", "qmodaltrans", "modalfrf", and "qmodalfrf". Two text files are written: "Qdisp.txt" and "Qforce.txt". Each line of the file contains data for a solution increment (a time or frequency step). For transient solutions, each column corresponds to a mode in the solution. Because FRF solutions are complex, two adjacent columns describe the complex modal amplitude (or force) for a mode. The modal force is,

$$f_{q_i}(t_n) = \phi_i^T F(t_n)$$

where $F(t_n)$ is the physical force at time t_n, and ϕ_i is the modal vector for mode i. The corresponding modal amplitude is defined by,

$$u(t_n) = \sum_{i}^{Nmodes} \phi_i q_i(t_n)$$

where $u(t_n)$ is the physical displacement and $q_i(t_n)$ is the modal amplitude. The expressions in the frequency domain are exactly the same, with frequency replacing time in the equations.

The text files may be loaded into matlab or MS/excel for analysis.

2.8.4 Subdomains

In parallel calculations, one results file is written per subdomain. Only data associated with that subdomain are written to the file. Use the "subdomains" option to specify which subdomains for which data will be written. The **subdomains** specification is made using a Matlab like string. The string should be enclosed in quotation marks to group the terms together. A range of values is represented by an initial value, an optional step, and a final value. For example,

```
subdomains '0:2:8'
```

selects subdomains 0, 2, 4, 6 and 8. Groups of such ranges may be combined using a comma. The following selects subdomains 0, 2, 3, 4, 6, 8, 9 and 15.

```
subdomains '0:2:8,3:3:9,15'
```

In addition, the keyword "all" selects all subdomains.

2.8.5 Memusage

The **memusage** keyword will output memory usage information to results file as well as a per processor, per task break down to a separate text file. The memory in the text file is shown in Mbytes. The output for the text file will be formatted as follows:

```
 1   2   3   4
40  41  41  42  "initialization"
23  12  15  15  "assembling matrices"
45  56  65  54  "initializing solver"
10  25  12  52  "writing output"
```

This keyword will only work if the **timing** keyword is also added.

2.9 OUTPUTS

The **outputs** section determines which data will be written to selected output files. All geometry based finite element results are written to an output **Exodus** file. The name of this file is generated by taking the base name of the input **Exodus** geometry file, and inserting *-out* before the file extension. For example, if the input **Exodus** file specification is *example.exo*, output will be written to *example-out.exo*. When using a multicase solution *(section 2.1.1)*, the case identifier is used in place of "out". More details are available in the **FILE** section (2.12).

Various non-geometry based finite element data, such as system matrices and tables may be available in Matlab compatible format, or in Harwell-Boeing format. These ASCII files have the *.m* or *.hb* file extensions respectively. The base file names are derived from the type of data being output. These files are generated in the current working directory.

In the following example, the mass and stiffness matrices will be output in Matlab format, but the displacement variables, stresses and strains will not be output. All the various options of the **OUTPUT** section are shown in Table 47. The next sections describe each of the options and their results assuming an input file named *example.inp* and a geometry file named *exampleg.exo*.

```
OUTPUTS
        maa
        kaa
        faa
//      displacement
//      stress
//      strain
//      energy
END
```

2.9.1 Maa

Option **maa** in the **OUTPUTS** section will output the analysis-set mass matrix (if it exists) to a file named *example_Maa.m*. If the harwellboeing option is selected, output will also go to a file named *example_Maa.hb*. These are the file names for the serial version of **Sierra/SD** . In the parallel version, an underscore and the processor number will precede the ".m", and a separate file will be written for each processor.

2.9.2 Kaa

Option **kaa** in the **OUTPUTS** section will output the analysis-set stiffness matrix to a file named *example_Kaa.m*. If the harwellboeing option is selected, output will also go to a file named *example_Kaa.hb*. These are the file names for the serial version of **Sierra/SD** . In the parallel version, an underscore and the processor number will precede the ".m", and a separate file will be written for each processor.

2.9.3 Faa

Option **faa** in the **OUTPUTS** section will output the analysis-set force vector (if it exists) to a file named *example_Faa.m*. If the harwellboeing option is selected, output will also go to a file named *example_Faa.hb*. These are the file names for the serial version of **Sierra/SD** . In the parallel version, an underscore and the processor number will precede the ".m", and a separate file will be written for each processor.

2.9.4 ElemEigChecks

Option **ElemEigChecks** will turn on the element output of the lowest eigenvalue, the 7th eigenvalue (commonly the first flexible eigenvalue), and the largest eigenvalue for the element. The output will be stored in the Exodus output file. The element variable names for the 1st eigenvalue, the 7th eigenvalue, and the maximum eigenvalue are *ElemEig_1st*, *ElemEig_7th*, and *ElemEig_max*, respectively. Note: All 3-d and 2-d elements have this capability. The **Beam2**, **OBeam**, **Spring**, **Truss**, **Spring3**, and **RSpring** elements are also supported. All remaining elements will output values of zero. Finally, if *ElemEig_1st* < *-1e-12 ElemEigi_max*, a negative eigenvalue warning will be printed.

2.9.5 Elemqualchecks

Option **Elemqualchecks** takes either one of three choices, **on**, **off**, or **sum**. The default is **sum**. If this option is **on** or **sum**, then all of the elements in the input file are checked for quality using methods developed by Knupp (Ref. 20). Knupp uses a condition number to evaluate the health of an element. The following table shows the elements currently checked and their acceptable ranges. The element quality reporting may also be modified by the *condition_limit* parameter specified in the *Parameters* section (2.3).

Element Type	Full Range	Acceptable Range
Hex8	$1 - \infty$	$1 - 8$
Tet4	$1 - \infty$	$1 - 3$
Tria3	$1 - \infty$	$1 - 1.3$
TriaShell	$1 - \infty$	$1 - 1.3$
Quad4	$1 - \infty$	$1 - 4$
Wedge6	$1 - \infty$	$1 - 5$

If the option **on** is selected and the element's condition numbers falls outside the acceptable range, a warning message is printed. The value output with the warning is normalized by the maximum number of the acceptable range for that element. If the option **sum** is selected, only a summary is printed, reporting the maximum condition number of all elements in the mesh.

In addition to these checks, solid elements are checked for negative volumes. This can occur if the node ordering for the element establishes a "height" vector using the right hand rule that is in the opposite direction of the actual element height. In other words, the nodes should normally be ordered in a counter clockwise direction on the bottom surface of the element. Some codes such as Nastran, are insensitive to this ordering. If element checks are run, then **Sierra/SD** will correct (and report) any solid elements found to have negative volumes. Without these corrections, the code will continue, but results that depend on these elements are suspect.

It is strongly recommended that any **Exodus** file with negative volumes be corrected.

2.9.6 Displacement

Option **disp** in the **OUTPUTS** section will output the displacements calculated at the nodes to the output **Exodus** file. The output file has the following nodal variables.

Variable	Description
DispX	X component of displacement
DispY	Y component of displacement
DispZ	Z component of displacement
RotX	Rotation about X
RotY	Rotation about Y
RotZ	Rotation about Z

In addition, if the analysis involves complex variables (currently `ceigen` section 2.1.25.3, frequency response analysis such as `modalfrf` or `sa_eigen`), then the imaginary vectors are also included. The imaginary component of the vector has *"Imag"* prefixed to the name. For example, the the imaginary component in the X direction is "`ImagDispX`".

2.9.7 Velocity

Option **velocity** in the **OUTPUTS** section will output the velocities at the nodes to the output **Exodus** file.

2.9.8 Acceleration

Option **acceleration** in the **OUTPUTS** section will output the accelerations at the nodes to the output **Exodus** file.

2.9.9 Strain

Option **strain** in the **OUTPUTS** section will output the strains for all the elements to the output **Exodus** file.

The following strains will be output for shell elements:

SStrainX1, SStrainY1, SStrainXY1	- *strains in the top layer of the shell*
SStrainX2, SStrainY2, SStrainXY2	- *strains in the mid-plane of the shell*
SStrainX3, SStrainY3, SStrainXY3	- *strains in the bottom layer of the shell*

Note: the top layer of the shell is determined by the ordering of the nodes of the shell. Also, the strains are in the local element coordinate system defined by the ordering of the nodes.

The following strains will be output for volume elements:

```
VStrainX, VStrainY, VStrainZ, VStrainYZ, VStrainXZ, VStrainXY
```

Note: These strains are in the global coordinate system, not the local coordinate system.

For more information on stress/strain recovery, see section 4.

2.9.10 Stress

Option **stress** in the **OUTPUTS** section will output the stresses for all supported elements to the output **Exodus** file. Only shell and volume elements are supported, i.e. there is no stress output for beams.

2.9.10.1 Shell Stresses

The following stresses will be output for shell elements.

SStressX1, SStressY1, SStressXY1, SvonMises1 - *top layer of the shell*
SStressX2, SStressY2, SStressXY2, SvonMises2 - *mid-plane of the shell*
SStressX3, SStressY3, SStressXY3, SvonMises3 - *bottom layer of the shell*

Note: the top layer of the shell is determined by the ordering of the nodes of the shell, and can be output by using the **EOrient** *output options (see section 2.9.25). Also, the stresses are in the local element coordinate system defined by the ordering of the nodes.*

2.9.10.2 Volume Stresses

For volume elements, the stress is always output in the global coordinate system, not the local coordinate system. The following stresses will be output for volume elements:

Variable	Value
VStressX	σ_{xx}
VStressY	σ_{yy}
VStressZ	σ_{zz}
VStressYZ	σ_{yz}
VStressXZ	σ_{xz}
VStressXY	σ_{xy}
VonMises	von mises stress

For more information on stress/strain recovery, see section 4.

2.9.11 VonMises

Option **VonMises** in the **OUTPUTS** section will output the von Mises stress for all the elements to the output **Exodus** file. For volume elements, the output will be the von Mises stress of the element. Surface elements define stresses on the top, center and bottom layers. The output will be the maximum of these 3 values.

Note that the von Mises stress is computed and output as a portion of the output if full stress recovery is requested. This option provides a mechanism for reducing output. Thus, if full stress output is requested, then the **VonMises** will provide no additional output. In other words, specifying both **VonMises** and **stress** in the outputs section is redundant, but does not result in an error.

2.9.12 Stress = GP

An output specification of **Stress = GP** reports stress at the Gauss points of volumetric elements. It is currently only available for Hex20 elements. Note that for a Hex20 there are 27 Gauss points with 6 stresses, for a total of 162 outputs per element.

The Gauss point ordering follows the description in the paper by Thompson.[21] For the convenience of the reader, that order is reproduced here.

2.9.13 VRMS

Option **vrms** will output computed root mean squared (RMS) quantities from a random vibration analysis. These quantities are written to a separate output file. Quantities output include the RMS displacement, acceleration and von Mises stress. In addition for the SVD option, the D matrix terms which contribute to the von Mises stress are output[4] (see section 2.1.21).

2.9.14 Energy

Option **energy** in the **OUTPUTS** section will place strain energies and strain energy density in the output **Exodus** file. Note that the current implementation of strain energies requires recomputation of the element stiffness matrix, which can be expensive.

2.9.15 GEnergies

Option **GEnergies** in the **ECHO** or **OUTPUTS** section will trigger computation of global energy sums for the results and output **Exodus** file, respectively. For the **ECHO** case, the computation includes the following.

strain energy The strain energy is computed from $u^T K u/2$ where u is the displacement and K is the current estimate of the tangent stiffness matrix. Note that this may not be complete for nonlinear solutions. Linear visco elastic materials have contributions that will not be included in this sum.

kinetic energy Computed as $v^T M v/2$. Here v is the velocity and M is the mass matrix.

work The **work** is defined as,

$$W(t) = \int_{x(0)}^{x(t)} F(x)dx$$

[4]For a definition of D, see Reese, Field and Segalman.

number	**label suffix**	**X**	**Y**	**Z**
1	111	0	0	0
2	112	0	0	A
3	110	0	0	-A
4	121	0	A	0
5	122	0	A	A
6	120	0	A	-A
7	101	0	-A	0
8	102	0	-A	A
9	100	0	-A	-A
10	211	A	0	0
11	212	A	0	A
12	210	A	0	-A
13	221	A	A	0
14	222	A	A	A
15	220	A	A	-A
16	201	A	-A	0
17	202	A	-A	A
18	200	A	-A	-A
19	011	-A	0	0
20	012	-A	0	A
21	010	-A	0	-A
22	021	-A	A	0
23	022	-A	A	A
24	020	-A	A	-A
25	001	-A	-A	0
26	002	-A	-A	A
27	000	-A	-A	-A

Table 38: Hex20 Gauss Point Locations. The constant A=0.77459666924148. The unit element is 2x2x2, with a volume of 8 cubic units.

where F is the force and dx is the distance traveled. This can be restated as an integral over time.

$$W(t) = \int_0^t F(\tau)v(\tau)d\tau$$

where $v = dx/dt$ is the velocity. We approximate this at discretized time t_n as,

$$W_n \approx \sum_i^n F_i v_i \Delta t$$

Note that this is a sum over time using the simplest method possible. Because of integration error, it may not be completely consistent with the other energies above. For the **OUTPUTS** case, the total energy is written out at each time step.

2.9.16 Mesh_Error

The **mesh_error** keyword causes mesh discretization error metrics to be computed. These are computed as output quantities, but the overhead associated with the metrics is not negligible. Mesh discretization quantities depend upon the solution type, and are not available for all solutions. Output is typically available as element quantities (usually in the *mesherr* field). For some mesh discretization errors, a global quantity is also output.

Output	**Description**
ErrExplicitLambda	Relative error in λ.
ErrExplicitFreq	Frequency error estimate (Hz)

We note that for eigenvalue analysis, *relative* errors are reported for the eigenvalue when using the **mesh_error** keyword. Thus, for a given eigenvalue λ, the reported error is

$$\text{ErrExplicitLambda} = \frac{\lambda_h - \lambda}{\lambda} \tag{32}$$

This is more convenient since the analyst does not have to divide by the eigenvalues to see the percent error. The global variable "ErrExplicitFreq" provides an absolute estimate (useful in plots for example).

2.9.17 Harwellboeing

Option **harwellboeing** in the **OUTPUTS** section will output the mass and stiffness matrices in Harwell-Boeing format to files with *.hb* extension.

2.9.18 Mfile

Option **mfile** will cause **Sierra/SD** to output various Mfiles like *Ksrr.m*, *Mssr.m*, etc. These files are mainly used by the **Sierra/SD** developers for code maintenance and verification. Since many of these files can be quite large, caution should be exercised when using this option on large models. An index of some of the files written using this option is provided in Table 39.

2.9.19 Force

Option **force** in the **OUTPUTS** section will output the applied force vector to the output **Exodus** file.

2.9.20 rhs

Option **rhs** in the **OUTPUTS** section will output the Right Hand Side vector from the calculations. For statics and dynamics, we repeatedly solve equations of the form, $Ax = rhs$. The **rhs** vector contains the applied forces and pressures as well as the inertial forces. Pseudo forces introduced in preload (say by TSR) are also part of this vector. This output is useful primarily for verification and debugging purposes.

2.9.21 EForce

Option **eforce** in the **OUTPUTS** section will output the element forces for line elements (such as beams and springs) to the output **Exodus** file. Each two node, 1-dimensional element will have 3 force entries for each node, for a total of 6 element forces per element.

The element force is not a stress or a strain, and should not be used as such. If you want beam stresses, you may want to mesh that portion of the structure either as a shell or a solid. Only limited stress output is available for beams. EForce is used primarily to help understand the behavior of nonlinear line elements such as the Joint2G element (see section 3.31). The output is actually the direct output of our internal force routine (which is a nonlinear routine). It can be quite confusing to output these nonlinear forces in a linear analysis.[5]

[5] Confusion arises because of the transformation to the element coordinate frame. For finite length elements, we perform a transformation of the element coordinate frame based on the displacements. After the coordinate frame is transformed, we express the element force in the new coordinate frame. This is done for both linear and nonlinear analyses. The resulting element force is no longer linear in displacement. Zero length elements do not have a rotated coordinate frame. Forces for zero length elements are linear in the displacement.

Table 39: Data Files Written Using the Mfile Option

Filename	**Description**
Stiff.m	Unreduced stiffness matrix including all active dofs
Kssr.m	Reduced stiffness matrix
Mass.m	Unreduced mass matrix
Mssr.m	Reduced mass matrix
LumpedMass.m	unreduced lumped mass matrix
xxx_gid.m	global IDs of the nodes
Fetimap_a.m	Map to convert from G-set to A-set The right hand side is the equation number. The lhs index is 8*(node index)+coordinate
Dampr.m	unreduced damping matrix (real components)
Dampi.m	unreduced damping matrix (imaginary components)
xxx_accelNN.m	G-set acceleration output of step NN
xxx_accel_aNN.m	A-set acceleration output of step NN
xxx_afNN.m	G-set applied force output of step NN
xxx_af_aNN.m	A-set applied force output of step NN
xxx_dispNN.m	G-set displacement output of step NN
xxx_disp_aNN.m	A-set displacement output of step NN
xxx_presNN.m	G-set nodal applied pressure of step NN
xxx_pres_aNN.m	A-set nodal applied pressure of step NN
xxx_velocNN.m	G-set velocity output of step NN
xxx_veloc_aNN.m	A-set velocity output of step NN
modal_amp.m	modaltransient output of mode amplitude vs time
ModalFv.m	modaltransient output of generalized forces

- The `xxx` above refers to the input file name root.
- The G-set output is 8*(number of nodes).
- The file names above are for the serial version of **Sierra/SD** . In the parallel version, an underscore and the processor number will precede the ".m". For example, the reduced stiffness matrix becomes `Kssr_0.m`. There is no output of a globally assembled, parallel matrix - it does not exist.
- Some solution methods will not write all files. For example, there are no mass matrices output in the solution of statics. Generally, matrices are output in sparse symmetric row format.

NOTE: The force returned is in the element (not global) coordinate frame. No provision is made for output of moments.

2.9.22 Residuals

For most solution types, a linear solver is used to compute systems of the form $Ax = b$. For direct serial solvers, these systems are typically solved to numerical precision. However, with iterative solvers the solution is only approximate. Sometimes it is advantageous to evaluate the performance of the solver. For example, regions with large residuals may be candidate areas for mesh refinement, or may point to other mesh problems.

Eigen. For eigen analysis, the residual is $(K - \lambda_i M)\phi$. The vector is *not* normalized by the norm of ϕ, or any other quantity. A nodal residual work is also output. This is the product $\phi^T(K - \lambda_i M)\phi$ summed to the nodes, i.e. on a given node we sum the contributing degrees of freedom. Again, the value is *not* normalized. Clearly with mass normalized eigenvectors (which do not have units of length), the units of the residual work are not energy, and the term may well be negative. The residual is output for each mode.

Transient Dynamics. For transient analysis the residual reported is $Au - b$, where A is the dynamics stiffness matrix (see section 1.1 of the theory manual). With a displacement based Newmark-Beta integrator the dynamic stiffness is $K + \frac{2}{\Delta t}C + \frac{4}{\Delta t}M$. The residual is output at each time step.

In addition to the residual vector, the norm of the residual is output as a global variable.

2.9.23 Resid_only

No longer supported. Use TIndex (section 2.9.24).

2.9.24 TIndex

It is occasionally useful to examine the residual after each iteration or solve. In the cases of nonlinear transient, nonlinear statics or eigen analysis, there may be many solves per output. Because of limitations in the output database format, it is very difficult (or impossible) to intersperse the residuals from each solve with the usual solution output. However, it is possible to select between the standard time step and an "iteration time step". Note that the **Exodus** database writes output for each "time step". It uses the step number as an index to the data, and only one such index is supported. When we substitute the iteration number for the time step we can write the data properly, but once iteration has completed,

we may not write data using the other index (time step, or mode number). Should that occur, we would have residuals from one iteration sharing the same time axis index with transient data. The parameters for the option are listed in Table 40.

Keyword	**Application**
standard	use time step or mode number as index
iteration	use the iteration count as index

Table 40: TIndex parameters

TIndex Example:

```
OUTPUT
     disp
     residuals
     tindex=iteration     // output on each iteration
END
```

TIndex makes sense only in solutions that require multiple iterations per solve. These include eigen analysis and nonlinear solutions. In other solutions, it is ignored, and output is provided at the standard time step.

NOTE: TIndex is really a debugging function. As such, we do minimal checks. In some solutions (notably eigen), it is possible to output data using both steps. However, the eigenvectors will normally be properly written to the first m steps (where m is the number of modes), and the residuals will be written once per solve. There is no clear way to relate the residuals with the eigenvectors.
NOTE: For eigen analysis, it is possible to output the the applied **forces** *at each solve. This is the only time that forces make sense in an eigen analysis. These are really the load vectors provided by the iterative eigenvalue scheme (***ARPACK***).*

2.9.25 EOrient

Option **eorient** in the **OUTPUTS** section will output the element orientation vectors for all elements. The element orientation is a design quantity that normally does not change significantly through the course of an analysis. This output is provided to help in model construction and debugging.

The orientation vectors are output as nine variables that collectively make up the three vectors required for element orientation. The output variables and the associated meanings for various elements are shown in tables 41 and 42 respectively.

Table 41: Element Orientation Outputs

Name	**Description**
EOrient1-X	first orientation vector
EOrient1-Y	
EOrient1-Z	
EOrient2-X	second orientation vector
EOrient2-Y	
EOrient2-Z	
EOrient3-X	third orientation vector
EOrient3-Y	
EOrient3-Z	

Table 42: Element Orientation Interpretation

Element	**EOrient1**	**EOrient2**	**EOrient3**
Beam2	axial	first bending (I1)	2nd bending (I2)
Shells	Element X	Element Y	Normal
Solids	Element X	Element Y	Element Z
Hexshell	Element X	Element Y	thickness
ConM	NULL	NULL	NULL

2.9.26 Pressure

Option **pressure** in the **OUTPUTS** section will output the applied pressure to the output **Exodus** file as a sideset variable as well as a new nodeset variable.[1] The addition of nodeset pressure output enables restarts using the output pressure as an input load. For most applications this also provides a useful tool for checking input loads.

2.9.27 NPressure

Option **NPressure** in the **OUTPUTS** section will output the *nodal* pressure to the output **Exodus** file as a nodal variable. This output is only available for solutions that introduce nodal pressure (currently only the random pressure loading).

[1] Prior to release 4.4 pressures were output as element variables. However, element variables cannot capture pressure applied to more than one face of an element, instead representing only one of those pressures with a single variable. Thus, element pressure output has been replaced entirely by sideset and nodeset pressure output.

2.9.28 APressure

Option **apressure** in the **OUTPUTS** section will output the acoustic pressure to the output **Exodus** file as a nodal variable. For purely acoustic elements, this will result in one degree of freedom per node, but for acoustic elements on the wet interface, this will result in four degrees of freedom per node in the output **Exodus** file.

2.9.29 APartVel

Option **apartvel** in the **OUTPUTS** section will output the acoustic particle velocity to the output **Exodus** file as an element variable. This is simply the velocity of the fluid particles. It is computed in **Sierra/SD** as the gradient of the velocity potential. For purely acoustic elements, this will result in three degrees of freedom per element.

2.9.30 Slave_Constraint_Info

Many linear solvers are very sensitive to redundant or conflicting constraint requirements. The **Slave_Constraint_Info** keyword requests output of constraint information on the slave nodes. This information can be used to help prepare models with less sensitivity to redundant constraints. Redundancy can be generated in any of the constraint types; here we report only the information from tied surfaces and tied joints. The following information is output when Slave_Constraint_Info is selected.

Num_Slave_Constraints indicates the number of sideset pairs in which a slave node appears. Typically a number greater than 1 is an issue.

Slave_Node_Redundancy provides an indicator of slave nodes which may have redundancy. A number above 0 indicates redundancy. The Slave_Node_Redundancy is typically one less than Num_Slave_Constraints unless there are more than 3 independent constraints for a specific sideset pair.[2]

Slave_Node_Gap indicates the distance a slave node must be moved to be placed on the master surface. Many problems with constraints stem from surfaces that do not properly match up geometrically.

Slave_and_Master indicates if a particular node is part of a master surface in one constraint relation and also part of a slave surface in a different constraint relation. A 1 means it is part of such relations and a 0 means it is not.

[2] As may occur if a sideset pair is used in both a tied constraint and a slip contact, or tied joint.

2.9.31 Statistics

For transient dynamics solutions only, summary statistical information may gathered and output for the time history of variables listed in Table 43. Currently we gather information about the mean and the standard deviation. Data is gathered at each time step, independent of the frequency of output (i.e. **nskip** is ignored).

Because this is summary data, it is not convenient to append this data to the file used for output of the time history. Another file is generated with the *"-stat"* tagging to store that data.

Statistical data requires *two* keywords for output. Both "statistics" and the keyword associated with that output quantity must be selected. To output statistics of the force, the following output section is required.

```
OUTPUTS
    statistics
    force
END
```

Keyword	Section	Comment
Displacement	2.9.6	
Velocity	2.9.7	
Acceleration	2.9.8	
Force	2.9.19	applied force
RHS	2.9.20	Right Hand Side vector at each load.

Table 43: Supported Statistical Data types for Transient Dynamics. Selection of these quantities along with "statistics" results in an addition **Exodus** file containing mean and standard deviation data.

2.9.32 KDiag

Option **kdiag** in the **OUTPUTS** section will output the maximum and minimum values of the diagonal of the stiffness matrix as nodal variables KDiagMax and KDiagMin. These are the max and min of the 7 variables associated with the 3 translational, 3 rotational and 1 acoustic degree of freedom on each node. These values are primarily useful for diagnostics purposes, where they may help identify regions of a model that have extremely high stiffnesses. All 7 terms may be seen by outputting kdiag in the **ECHO** section.

Figure 19 illustrates the use of this option. Note how the center sections of the model are highlighted by their stiffness terms. This tool is especially important for analyzing some collections of beams. Since beam stiffnesses are proportional to $1/L^3$, it is common to accidentally generate beams of extremely high stiffness, which can ruin the numerical solution. See section 2.9.33 for a related diagnostic on the dynamics matrix.[3]

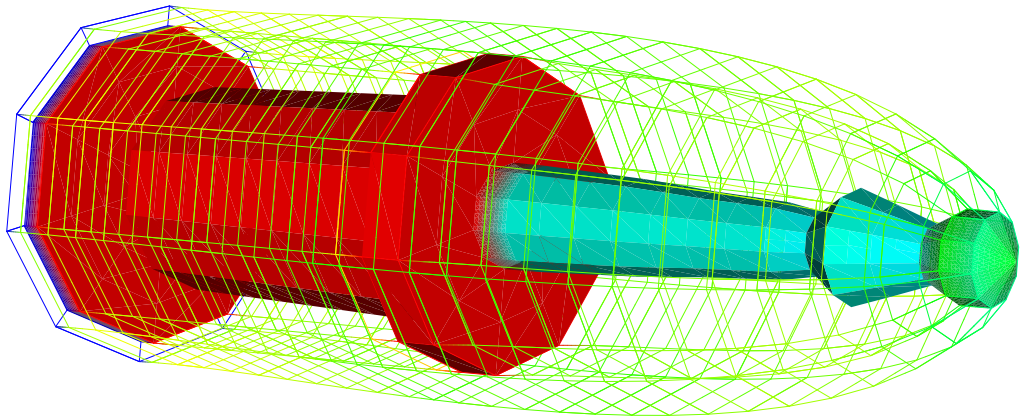


Figure 19: Example *KDIAG* output.

[3] The stiffness diagonal and dynamic matrix diagonal depend to some extent on the linear solver used. Domain decomposition solvers generally use Lagrange multipliers to eliminate constraints, while some sparse solvers remove constraints through reductions of rows and columns of the matrices. Because the matrices to be solved are different, the diagonals and conditioning of the matrices are also different.

2.9.33 ADiag

Option **adiag** in the **OUTPUTS** section will output the maximum and minimum values of the diagonal of the dynamics matrix as nodal variables ADiagMax and ADiagMin. Refer to the KDiag section, (2.9.32), for format information.

The "dynamic matrix" is the matrix which is solved by the linear solver. The "ADiag" diagnostic can help identify regions of the model that may contribute to poor matrix conditioning. Summary of a few of the dynamics matrix terms are listed in Table 44. Refer to the theory manual for details of the matrix to be solved. Dynamics matrix output is available for most solvers (including GDSW), and for some solution methods.

Solution	**Matrix**	**Comment**
eigen	$K - \sigma M$	real eigen problem
transient	$K + \frac{4}{\Delta T^2} M + \frac{2}{\Delta T} C$	standard Newmark-Beta
statics	N/A	dynamics matrix is stiffness matrix
qevp	N/A	unimplemented

Table 44: Selected Dynamic Matrix Definitions

2.9.34 Warninglevel

We have *partially* implemented some control over the output of warning messages. This is not implemented in general, but may be useful for some cases. The keyword **warninglevel** may be followed by either an integer, or a string.

Level	**Descriptor**	**Comment**
0	none	minimal warning output
1	severe	only severe warnings output
2	bad	severe and bad warnings output
4	information	all warnings (default)

Table 45: Warning Diagnostic Options

2.9.35 Precision

The binary results in **Exodus** files may be stored in either a lower or higher precision. For most applications, lower precision is sufficient. This typically represents 8 digits of accuracy which is more than any physical model of the structures warrants. However, in some cases, a

higher precision is appropriate. This can be very important for restarts in eigen analysis for example, where it is necessary to maintain orthogonality of eigenvectors. See section 2.2.1. The options are shown in Table 46.

Keyword	Description
low	Typically 4 bytes or about 8 digits accuracy. Default
high	Typically 8 bytes or about 16 digits accuracy

Table 46: Output Exodus Precision Options

2.9.36 ddamout

The **ddamout** keyword is a single keyword that can be used to trigger **Exodus** output (if used in the OUTPUT section), or history file output (if used in the HISTORY section). We note , by default, that much of this data is written to .txt files in ddam analysis - however, in many cases it is convenient to have it in **Exodus** files also.

Table 48 lists the nodal and element variables that are output when the **ddamout** keyword is selected. There are a total of 8 nodal variables and 2 element variables.

Finally, we note a couple of additional details for output of ddam data.

- In parallel runs, the .txt file output will not include nodal variables, since that data would not be usable in that form. Instead, that data could be written to the **Exodus** file with the **ddamout** keyword.
- History output of ddam data will only write the nodal variables, not the element variables. Element variable history output for ddam analysis is currently not in place.

2.10 HISTORY

All the results from the "OUTPUT" section can be output to a limited portion of the model using a history file. Only those outputs described in Table 47 are supported. Note that if the output is also specified in the OUTPUT section, there is little need to write the data in the history file. The following output section options are ignored in the history section because all history file output will be in the **Exodus** format.

- mfile
- harwellboeing
- kaa, maa, faa

Table 47: OUTPUT Section Options

Option	Description
maa	mass matrix in the a-set
kaa	stiffness matrix in the a-set
faa	force vector in the a-set
elemqualchecks	on ‖ off ‖ summary, default is summary
ElemEigChecks	outputs first eigenvalue, seventh eigenvalue, and the largest eigenvalue
disp	displacements at nodes
velocity	velocity at nodes
acceleration	acceleration at nodes
strain	strain of element
stress	stress of element
vonmises	vonmises stress on element
stress = gp	element stresses at Gauss points (huge)
vrms	RMS quantities (random vibration only)
energy	element strain energy and strain energy density
genergies	global sum of energies
mesh_error	mesh discretization errors
harwellboeing	mass and stiffness matrices in Harwell-Boeing format
mfile	Outputs various Mfiles (mainly for developers)
locations	Outputs nodal coordinates and DOF to node map
force	Outputs the applied force
rhs	Outputs RHS of system of equations to be solved
pressure	Outputs pressure load vector
eforce	Outputs element forces for beams
eorient	Outputs element orientation vectors
statistics	Mean and Std deviation of some variables
warninglevel	Control of warning messages

Table 48: Variables that are output from ddam analysis

Option	data type	Description
ddam_mdisp	nodal variable	ddam modal displacements
ddam_mvel	nodal variable	ddam modal velocities
ddam_macc	nodal variable	ddam modal accelerations
ddam_mfor	nodal variable	ddam modal forces
ddam_nrl_sdisp	nodal variable	ddam nrl-summed displacements
ddam_nrl_svel	nodal variable	ddam nrl-summed velocities
ddam_nrl_sacc	nodal variable	ddam nrl-summed accelerations
ddam_nrl_sfor	nodal variable	ddam nrl-summed forces
ddam_mvmstr	element variable	ddam modal vonmises stresses
ddam_nrl_svmstr	element variable	ddam nrl-summed vonmises stresses

- vrms

In addition to the **output** selection options, the history file section contains information about the regions of output. The default is NO output selection. Selection may be for node sets, side sets, a node list file (see section 2.14.3), or element blocks. Virtual blocks can be included in this section. For example, one could output the element force in a virtual Joint2G element. If side sets are selected, the side set selection is for the nodes associated with that side set, not for the elements themselves. All nodal variables selected in the history file will be output for all selected nodes. Selecting an element block automatically selects the associated nodes in that block. The format for the selection uses a Matlab concatenated range that of the subdomains selection in the ECHO (section 2.8.4). For example,

```
HISTORY
   nodeset '1:10,17'
   sideset  '3:88'
   nodeset '8,15' coordinate 4
   block 5,6,3
   stress
   disp
   nskip 10
   flush 4
END
```

Any number of **nodeset** selections can be specified in the history section. Nodeset specifications may be followed by an optional coordinate entry. If a **coordinate** is specified, all nodal results for the nodes in the nodeset are transformed to the specified coordinate system before output to the file. If a particular node is identified in more than one specification, the last specification is used for the output. The coordinate ID of nodes in the history file may be printed out in the echo file by specifying **nodes** in the **echo** section of the input. The coordinate ID will also be written to the history file (as a nodal variable *CID*) provided

any nonzero coordinate frames have been specified. Note that the **coordinate** keyword for history section output will only work with nodesets, it is not supported for node list files, sidesets, or blocks.

Only one **block** and one **sideset** specification is permitted in the history section. Note that for **block** and **sideset** output specifications, the corresponding block and/or sideset numbers should be specified as shown in the above example. Multiple block and/or sideset numbers should be separated by commas or colons, but not spaces. For example, the following specification for the history file would only output the element force in block 1, but not in block 2.

```
HISTORY
   block 1, 2
   eforce
END
```

In order to get element force output for both blocks, one could use the following

```
HISTORY
   block 1,2
   eforce
END
```

Alternatively, quotes could be used without the comma

```
HISTORY
   block '1 2'
   eforce
END
```

Output coordinate frames may only be specified on nodesets.

In transient dynamics solutions, user control of output step interval "nskip" and output buffer "flush" operations are provided to increase efficiency of output. See section 2.1.34 for examples. The history file respects the "nskip" and "flush" parameters set in the solution block, but additional user control is provided for history files by inserting the **nskip** and **flush** keywords in the history block. In that case, history files for all multicase solutions will have output and buffer flushing at the intervals specified in the history section, and the entries in the solution section will be ignored for history files.

Unlike subdomains, node set and side set IDs need not be contiguous in the **Exodus** file. The selection criteria may identify nonexistent sets. These will be silently ignored. In the above example, if the input **Exodus** file contains no node set with ID=10, it will not be

treated as an error. Node set and side set IDs in the history file will be consistent with the corresponding **Exodus** input file.

Only one history file will be written per analysis. The name of the history file is derived from the name of the **Exodus** output file, except that the extension is ".h". Table 65 shows the corresponding values for cylindrical and spherical coordinates.

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited when output in many coordinate frames is desired. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node. See the **coordinate** section (2.28) for information on obtaining the transformation matrices from each coordinate frame directly.

2.11 FREQUENCY

The frequency section provides information for data output from the modalFRF, direct-FRF, shock, modalshock, and random vibrations solution methods. One frequency file is written per analysis. The name of the frequency file is derived from the name of the **Exodus** output file, except that the extension is ".frq". The section format follows that of the history section. As in the case of the history section, data can be written to a sideset, nodeset, node_list_file, or a block. In the case of output to a block, the block can be a virtual block. Thus, one could output element force on a Joint2G element. Solution methods that do not write frequency domain output silently ignore the Frequency section.

The frequency section also includes the definitions of the frequency values for calculation. A frequency section (with some output selection region) must be selected for any solution method requiring frequency output. To fail to do so is an error, since the solution would be computed and no output provided.

The frequency values may be specified using the methods specified in Table 49. The methods are mutually exclusive, i.e. do not mix keywords from the "linear" method with those of the "table" method. An example follows.

```
FREQUENCY
   nodeset '1:10,17'
   sideset  '3:88'
   block 5,6,3
   disp
   acceleration
   freq_min=10    // starting frequency in HZ
   freq_step=10   // frequency increment
   freq_max=2000  // stop freq. This example has 201 frequency points.
END
```

Table 49: Frequency Value Specification Methods

Method	Keyword	Description
method=linear	freq_min	minimum frequency (typically in Hz)
	freq_max	stop frequency
	NF	number of frequency intervals.
	freq_step	frequency increment (or use NF)
method=log	freq_min	minimum frequency (typically in Hz)
	freq_max	stop frequency
	NF	number of frequency intervals.
method=table	table	name of a 1D table (see section 2.31)

For the "linear" and "log" methods, the frequencies are obtained by the following equation.

$$F_k = \begin{cases} F_{min} + k \cdot F_{step} & \text{for method=linear} \\ F_{min} \exp(kD) & \text{for method=log} \end{cases}$$

where $D = 1/N_F \log\left(\frac{F_{max}}{F_{min}}\right)$. If both `freq_step` and `NF` are specified, `NF` is used.

Output Region:

The controls in the **frequency** section also affect data written to the results (or echo) file. In particular, the echo file contains data only for those nodes in the selection region of the **frequency** section. Selection of a specific output (such as displacement or acceleration) is independent. For example, you may echo only displacements, but write displacements and accelerations to the **Exodus** frequency output file. The history section (2.10) has more information on specification of the output region.

The *seacas* translator *exo2mat* may be used to translate the output into Matlab format for further manipulation and plotting.

2.12 FILE

Disk files names are specified in the **FILE** section. The parameters for the **FILE** section are,

Option	Description
geometry_file	Indicates which **Exodus** file to use
sierra_input_file	optional file name for transfer of data from a sierra application

2.12.1 geometry_file

The geometry file is used for input of the mesh geometry including the nodes, elements, connectivity and attributes. It is typically a binary **Exodus** file.

2.12.1.1 Multiple Processor: In a multiprocessor environment, the file name is determined by appending the "dot qualified" processor number and processor id to the geometry file specification.[4] For example, if the user specifies,

```
geometry_file='temp1/example.par'
```

and there are 4 processors, then the following files will be opened.

```
temp1/example.par.4.0
temp1/example.par.4.1
temp1/example.par.4.2
temp1/example.par.4.3
```

2.12.1.2 Single Processor: On a single processor, the file is not "spread", and the full file path is provided. For example, on a single processor, a **FILE** section may look like this.

```
FILE
    geometry_file 'exampleg.exo'
END
```

Note:

- If the file name is not included in quotes, it will be converted to lower case.
- A single processor run, even using mpi protocol, will not append the number of processors and processor ID to the file name.
- Appendix 3 shows the steps involved in the parallel execution of **Sierra/SD** .

[4] In other words, the user specifies the path name of the first parallel file, but omits the processor count information. This method permits specification of the file name independent of the number of processors used.

2.12.2 sierra_input_file

The **sierra_input_file** may be used as a restart following a sierra calculation (using **Presto** for example). This is an alternative to directly transferring the same data using the sierra transfer services. The **sierra_input_file** has the same format and usage as **geometry_file**, and can be used to transfer data in parallel or serial. See also section 2.1.29.

2.12.3 Additional Comments About Output

A text log or *results* file can be written for the run. Details of the contents of the results file are controlled in the **ECHO** section (see section 2.8). The results file name is determined by the name of the input file, and will be in the same directory as the input text file, regardless of whether **Sierra/SD** is being executed in serial or parallel. However, if executing in parallel, using the "subdomains" option in the **ECHO** section allows control of the number of *results* files. For example, if running on 100 processors, up to 100 result files may be output. Using **subdomains** "`0:2`" will only output three files, from subdomains 0, 1, and 2. The default is to output a *results* file only for processor zero. The results file name uses the base name of the input, with an extension of ".rslt". In a parallel computation, the results file names use the base name of the input file, followed by an underscore and the processor number, then followed by the "`.rslt`" extension.

For calculations in which geometry based output requests are included (see section 2.9), an output **Exodus** file will be created. The **Exodus** file is a binary file containing the original geometry plus any any requested output variables. The output **Exodus** file name is determined from the geometry file name. The base name of the output is taken from the geometry file by inserting the text "`-out`" just before the file name extension. The output **Exodus** file will be written to the same directory where the geometry file is stored. If executing **Sierra/SD** on a parallel machine, the **Exodus** output files should be written to the raid disks for reasonable performance.

2.13 Linesample

The line sample (**linesample**) section of the input file provides a means of evaluating and outputting fields or internal variables at sampling points within a structure. These sampling points are defined on a series of lines.

Section 2.1.36 discusses the primary application of line sample, verification of stress field input to **Sierra/SD** from TSR. Line sample is used for energy deposition (see *Two Element Exponential Decay Variation Hex20* in the Verification manual[14]). Energy deposition is interchangeable with supplying an applied temperature. And line sample is used for acoustics problems (for far-field processing (see 2.14.7.1 or How To[22]) for example with infinite

elements.[23]

Keywords for the line sample input are listed in the table below. An example follows.

Keyword	Arguments
samples per line	*integer*
endpoint	*6 real numbers*
format	*string*

samples per line The number of sample points on each line. All lines will have the same number of samples.

endpoint The endpoints of the line. There should be 3 real numbers for the XYZ location of the beginning of the line, followed by 3 real numbers at the end. There can be any number of endpoint entries.

format The format of the output file. Two output formats are current supported: `exodus` and Matlab `mfile`. The default is `mfile`.

Output from the line sample is written to `linedata.m` for mfile output, or to `linedata.exo` for **Exodus** format. There is no need to join this data for parallel runs. In those output files, a nodal variable called `Displacement` will be created. The entries in this array correspond to 3 displacement variables, 3 rotation variables, acoustic pressure, and generalized degrees of freedom. For transient data, the time values are also output for each of these arrays.

2.13.0.1 LineSample Example:

```
LINESAMPLE
   samples per line 5
   endpoint 0. 0. 0.   1. 1. 1.
   endpoint 0.0 0.5 0.5   1. 0.5 0.5
   format exodus
END
```

2.14 BOUNDARY

Boundary conditions are specified within the **Boundary** section. Node sets, side sets or node lists may be used to specify boundary conditions. Currently the **coordinate** keyword is not supported for boundary conditions. [1] The example in Figure 20 illustrates the method.

The descriptors for the displacement boundary conditions are, **X**, **Y**, **Z**, **RotX**, **RotY**, **RotZ**, **P**, and **fixed**. Their application and meaning are listed in Table 50. An optional

[1] Robust BCs may be applied *only* in the 3 coordinate axes of the basic coordinate frame.

```
BOUNDARY
  nodeset 1
       x = 0.1    // constrain x=0.1 for all nodes in set
       y = 0      // constrain y=0 throughout nodeset 1
       RotZ = 0   // constrain the rotational dof about Z
  nodeset 2
    fixed         // constrain all structural dofs in nodeset 2
  nodeset 3
    accelx = 0.3  // constrain the x component of acceleration,
    function=1    // in nodeset 3, with the time-dependence
    disp0 = 0.0   // given by function 1, and initial conditions
    vel0 = 0.1    // given by disp0, vel0
  sideset 2       // acoustic sideset
    p = 0         // fixed acoustic pressure
                  // (also known as pressure release condition)
  sideset 3       // acoustic sideset
    pdot = 1.0   // constrain the time derivative of acoustic
                 // pressure for enforced accelerations
    function = 2 // in sideset 3, with the time-dependence
    p0=1.0       // given by function 2, and initial condition p0
  sideset 5
    absorbing    // apply absorbing boundary condition on sideset 5
  sideset 6
    impedance_pressure = 0.5 // pressure impedance bc on sideset 6
    impedance_shear = 0.5 // apply shear impedance bc on sideset 6
  sideset 7
    slosh = 0.6   // apply slosh boundary condition on sideset 7
  sideset 8
    infinite_element // apply infinite elements on sideset 8
    radial_poly = legendre
    order = 5
    origin = 0 0 0
  node_list_file='clamped.nodes'
    fixed
END
```

Figure 20: Example Boundary Section

Keyword	Description
prescribed displacement keywords	
X	X Component of displacement
Y	Y Component of displacement
Z	Z Component of displacement
RotX	Component of Rotation about X axis
RotY	Component of Rotation about Y axis
RotZ	Component of Rotation about Z axis
fixed	Constrain all components of rotation and translation
P	Acoustic pressure
prescribed acceleration keywords	
AccelX	scaling factor on X component of motion
AccelY	scaling factor on Y component of motion
AccelZ	scaling factor on Z component of motion
RotAccelX	scaling of rotational motion about X axis
RotAccelY	scaling of rotational motion about Y axis
RotAccelZ	scaling of rotational motion about Z axis
disp0	unscaled initial displacement
vel0	unscaled initial velocity
Pdot	derivative of acoustic pressure

Table 50: Boundary Enforcement Keywords

equals sign separates each descriptor from the prescribed value. The value **fixed** implies a prescribed value of zero for all degrees of freedom.

2.14.1 Prescribed Displacements and Pressures

In linear statics, one may prescribe a nonzero displacement by entering a value following the coordinate direction. In the example above, the displacement for nodeset 1 is set to 0.1 in the X direction.

For acoustics, pressures may be fixed by specifying $p = 0$, as in the above example on sideset 2. This corresponds to a pressure release condition.

For linear statics, there must be no **function** entry following the entry. Prescribed displacements have the same limitations as prescribed accelerations described in the next section. The load in this case is introduced by the prescribed displacement. However, the **loads** section must exist (for error checking purposes) even if it is empty.

2.14.2 Prescribed Accelerations

In transient dynamics, the acceleration on a portion of the model may be prescribed as a function of time. The descriptors for prescribed accelerations are, **accelX**, **accelY**, **accelZ**, **RotaccelX**, **RotaccelY**, **RotaccelZ**, **disp0**, **vel0**, and **Pdot**, as shown in Table 50. A function must be used to apply the time-dependent boundary accelerations. Optional initial displacement and velocity can also be specified; if not, they default to 0. In the example above, the x acceleration of nodeset 3 will be prescribed as $0.3 \times f(t)$, where $f(t)$ is defined in function 1. The initial displacement is given as 0, and the initial velocity is 0.1. Currently, only accelerations can be prescribed. However, this does not preclude problems with prescribed velocities and displacements, since these cases can be converted to a prescribed acceleration by differentiation.

$$u(t) = a * \left[\int_0^t \left(\int_0^t f(t)dt \right) dt + t * v_0 + u_0 \right] \tag{33}$$

In transient dynamics, the equivalent displacement can be calculated as shown in Equation 33. Here a is really a scale factor on motion, rather than just acceleration. Note that if no function is listed, an error message will be generated.

In the case of an acoustic sideset or nodeset, the prescribed value is the first time derivative of acoustic pressure, denoted above as **Pdot**. This is because, internally, **Sierra/SD** solves for the velocity potential, and the first time derivative of the velocity potential is the acoustic pressure. Thus, by specifying the first time derivative of pressure, one is actually prescribing the acceleration of the velocity potential.

An additional point to consider when applying prescribed accelerations is that the initial velocity and displacement (denoted as **disp0** and **vel0**), are also necessary to completely define the boundary condition. These values account for the constants of integration obtained when integrating the prescribed acceleration to obtain the corresponding velocity and displacement on the sideset or nodeset. In the case of acoustics, only one initial condition is needed (**p0** which specifies the initial acoustic pressure), since only the first time derivative of acoustic pressure is specified. Note that **disp0**, **vel0**, and **p0** all default to 0 if not specified.

There are some limitations with the prescribed acceleration capability, which are listed in the following, and in Table 51. First, prescribed accelerations are not currently set up to work with multicase solutions. Also, they only work in the standard (Cartesian) coordinate system. Prescribed accelerations can be used in meshes that have nonlinear or visco elastic elements, as long as the prescribed accelerations are not applied directly to the nonlinear or visco elastic elements. Note that the nodes involved in prescribed accelerations cannot coincide with nodes that are involved with mpcs.

Finally, note that when prescribed accelerations are used, they induce a load on the structure. Thus, in many cases the **loads** section serves no purpose, unless an additional external load is applied. In these cases, however, an empty **loads** block is still needed in the input file. An error message will be generated if the input file has no **loads** section.

1. No support for multicase.
2. Only in basic coordinate directions.
3. Cannot be used on nodes attached to visco elastic elements.
4. Cannot be used on nodes attached to nonlinear elements.
5. Cannot be used on nodes connected to rigid elements or MPCs.
6. Load section is required, even if empty.

Table 51: Limitations for Prescribed Boundary Conditions

2.14.3 Node_List_File

To make it a little easier to apply boundary conditions, a *node_list_file* option is provided. In this option, an additional text file is provided which contains a list of global node ids separated by white space. No comments, or other characters are allowed in the file, as shown in Figure 20. The remainder of the boundary condition specifications are unchanged.

There are several limitations place on collections of nodes specified in this manner.

1. This is a rather inefficient method of supplying the nodes. It is recommended that nodesets or sidesets be employed when practical.
2. No node distribution factors may be provided.
3. The output **Exodus** file will have no record of this list.
4. The global node numbers are the unmapped **Exodus** indices. This means that the numbers go from 1 to N, where N is the maximum number of nodes in the model. This definition is the only one which allows the same node numbering to be used in both a serial and parallel file.
5. There is NO requirement that the nodes be sorted in the list, but repeating a node in the list can have undefined results, i.e. don't do it.

2.14.4 Nonreflecting Boundaries

Nonreflecting boundary conditions for acoustics and for elasticity may be specified using the “absorbing" keyword.

This section allows the user to specify an exterior boundary for acoustic, elastic, or coupled structural acoustic simulations. Once specified, first-order non-reflecting boundary conditions are applied on this surface. The boundary is specified with a sideset. The sideset

can be placed either on acoustic or elastic elements - the code automatically determines whether the sideset is placed on acoustic or elastic elements, and then applies the appropriate boundary conditions. For acoustic elements, only pressure waves need to be absorbed, whereas for elastic waves both pressure and shear waves need to be absorbed.

For acoustic elements, the absorbing boundary could represent an infinite fluid surrounding a structure. For elastic elements, it could represent an infinite elastic medium, such as in a seismic problem.

An example of this syntax is given below

```
BOUNDARY
  sideset 5
     absorbing
     radius = 1.0
END
```

The parameter "radius" specifies the radius of the sphere that defines the absorbing boundary. For a planar absorbing surface, one can either specify no radius, or a very large radius (the radius is actually equal to infinity for a planar surface). In those cases, the absorbing boundary condition reduces to a plane-wave absorbing condition. We also note that the radius parameter refers to the distance from points on the spherical surface to the center of curvature, not to the origin of the coordinate system. Thus, it is independent of the coordinate system that is specified. For example, one could shift the coordinates of the nodes of the acoustic mesh by any constant, but the radius parameter would remain the same.

2.14.5 Impedance Boundary Conditions

Impedance boundary conditions are partially reflecting and partially absorbing. Thus, they are somewhere in-between a rigid wall and an absorbing boundary condition. They reduce down to these special cases for certain choices of the impedance parameters.

An example syntax for an absorbing boundary condition is given below

```
BOUNDARY
  sideset 6 // sideset on acoustic material
     impedance = 0.5
  sideset 7 // sideset on elastic material
     impedance_pressure = 0.5
     impedance_shear = 0.5
END
```

In this case, sideset 6 is attached to acoustic elements, and sideset 7 is attached to elasticity elements. For acoustic elements, only one impedance parameter is needed, and it corresponds to an impedance condition for pressure waves only (acoustic elements support no shear waves). For elasticity elements, the **impedance_pressure** and **impedance_shear** correspond to impedances for pressure and shear waves, respectively. This example specifies that sideset 6 is to have an impedance of $Z = 0.5\rho c$, where ρ is the density and c is the speed of sound. Thus, the "impedance" parameter that is parsed in is simply the multiplier on the characteristic impedance ρc. Similarly, for the elasticity element the pressure and shear impedances would be $Z_P = 0.5\rho c_P$ and $Z_S = 0.5\rho c_S$, where c_P and c_S are the speeds of sound for the pressure and shear waves, respectively.

Currently, impedance boundaries are only set up to work with the standard characteristic impedance ρc. Thus, specifying the "radius" parameter with an impedance boundary condition will have no effect.

We note that if the impedance parameters are all set to 1.0, the problem reduces to the absorbing boundary described in the previous section. If set to 0, the impedance condition becomes a pressure-release boundary for acoustics and a free boundary for an elasticity element. If set to a very large number, the impedance boundary condition reduces to a rigid-wall condition for acoustics, and a fixed condition for elasticity elements.

2.14.6 Slosh Boundary Conditions

Slosh boundary conditions are applied at free surfaces that are effected by gravity. This type of free surface is typically only important on the surface of a liquid such as water. It adds an additional contribution to the mass matrix that results in "surface" wave modes.

An example syntax for an absorbing boundary condition is given below

```
BOUNDARY
  sideset 7
     slosh = 0.102 // 1.0/9.8 (m/s^2)
END
```

This specifies that sideset 7 is to have a slosh boundary condition. In this case, the slosh coefficient needs to be set to $\frac{1}{g}$, where g is the gravity constant. Thus, for SI units, the slosh coefficient is 0.102. Currently, slosh boundary conditions are only valid for acoustic elements. Applying them to elastic elements will generate an error.

2.14.7 Infinite Elements

In this section, we describe how to use infinite elements for acoustics. These elements serve as both high-order absorbing boundary conditions, as well as far-field calculators that

Parameter	Description	Options	default
radial_poly	the type of polynomial for radial expansion	legendre	legendre
order	the order of the radial basis	0-19	0
source_origin	the origin of the ellipsoid	3 real numbers	0 0 0
ellipsoid_dimensions	radial dimensions of ellipsoid axes	3 real numbers	0 0 0
neglect_mass	indicates whether to neglect infinite element mass	yes or no	yes
correct_mass	whether to correct negative mass terms.	yes or no	yes

Table 52: Available parameters for the infinite element section

allow the analyst to compute the solution at far-field points outside of the acoustic mesh. This latter step is a post processing step.

The infinite element specification begins with a sideset on the **Exodus** file of interest. Currently, that sideset has to be a spherical surface or part of a spherical surface. Thus, a full spherical surface, hemispherical surface, or a quarter of a sphere would all be acceptable. Note that the infinite element accuracy will degrade if the element surfaces on the spherical boundary do not adequately represent the spherical surface. The finite element surfaces will be faceted, but enough elements on the boundary are needed to represent the spherical curvature.

Once a sideset is identified for the infinite element surface, the **BOUNDARY** section in the input deck would be modified as follows

```
BOUNDARY
  sideset 1
  infinite_element
    radial_poly = legendre
    order = 5
    source_origin = 0 0 0
    ellipsoid_dimensions 15 15 30
    neglect_mass = yes
END
```

The parameters are summarized in Table 52. Currently, only Legendre polynomials are available for the radial basis. In the future we expect to offer Jacobi and Lagrangian polynomials also. The order of the polynomial can vary from 0 to 19. Order 0 corresponds to a simple absorbing boundary condition. Higher orders will be more accurate, but also more computationally expensive. The source point is the location of the center of the spherical surface that the infinite elements emanate from. This would coincide with the origin of a spherical coordinate system that is anchored to the spherical surface of the infinite elements.

The ellipsoid_dimensions parameters indicate the axial dimensions of the ellipsoid in the

global coordinate system. They are specified as ellipsoid radii rather than ellipsoid diameters. In the case of a sphere, all 3 parameters are equal and set to the radius of the sphere. These parameters are currently required, and an error will be generated if they are not specified.

The **neglect_mass** keyword indicates whether to neglect the mass matrix contributions from the infinite elements. Note that for a spherical surface, the mass matrix contributions from an infinite element are identically zero. However, when numerically generated, small entries will be present in the mass matrix, and thus an option is provided to include these terms in the analysis. It is recommended to neglect the mass in most cases, and thus this keyword would typically be set to **yes**. By default, **neglect_mass** is set to yes.

Note that infinite elements only require a specification of a sideset on the surface of interest. No elements need be set up explicitly on this interface. Internally, **Sierra/SD** constructs virtual elements and virtual nodes that define the actual infinite elements, but the analyst need not build a layer of elements on the boundary of the sideset.

In the time domain, infinite elements can be used in serial or parallel. In the frequency domain (i.e. for **directfrf**) solutions, only the serial capability is currently functioning. The parallel capability will require enhancements to the parallel complex solver.

The infinite element formulation in **Sierra/SD** uses a Petrov-Galerkin formulation, rather than a standard Galerkin formulation. As a result, nonsymmetric system matrices are encountered with infinite elements. This restricts the solver options to the GDSW solver for time domain and either the GDSW or the SUPERLU solver for frequency domain (i.e. **directfrf**).

We note that infinite elements can be used either with purely acoustic problems, or with coupled structural acoustics. The formulation is the same, and the GDSW solver is required for the solutions since nonsymmetric matrices are encountered.

2.14.7.1 Far-Field Postprocessing The infinite element formulation allows the analyst to compute the response outside of the acoustic mesh as a postprocessing step. The response can be computed at any point outside the mesh, and for any period of time. Currently, the **linesample** capability is used to write out the far-field data (see section 2.13). This data may be written in a readable Matlab format, which can easily be read in to create plots of the data.

The output will be written to a **Matlab** m-file with the name "linedata.m" or "linedata.exo", depending on which option is selected for output. One file is written per analysis (results are joined analogous to history file output). For example, reading this file in will create vectors **FieldTime** and **Displacement**. The acoustic pressure is found in the **Displacement1** variable.

We note that the infinite element output in the far-field is always given with respect to some time shift. Details of this are given in the theory notes on infinite elements. The shifted times are included in the `linesample` output for the analyst to use. These allow for

plotting the time histories against the appropriate time vectors.

The shifted time output is available in the linesample output in a nodal array called FieldTime. The dimension of the FieldTime array is the same dimension as the acoustic pressure output, since each node in the linesample output has its own FieldTime array. One FieldTime array is available for each sample point in the `linesample` output.

The following command in Matlab will plot the pressure for the first sample point.

```
FieldTime = nvar09;
pressure= nvar01;
plot(FieldTime(1,:),pressure(1,:))
```

We note that the linesample points defined in the **LINESAMPLE** file can contain points that are both inside and outside of the acoustic mesh. For points that are inside of the mesh, the FieldTime array for each node will be identically equal to the time array. For points outside of the acoustic mesh (i.e. inside of the infinite element mesh), the FieldTime values will be larger than the corresponding time values in the Time array, since the acoustic waves will take additional time to reach these far-field points.

2.15 LOADS

Loading conditions are specified within the **loads** section. The following example illustrates the method.

```
LOADS
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      coordinate 11
      force = 0. -1 0
   nodeset 7
      point_volume_vel = 1
      scale = 1.0
      function = 1 // time history of dV/dt,
                   // where V is the volume of the source
   body
      gravity
      0.0 1.0 0
      scale -32.2
   body
```

```
      thermal_load
      function = 1
   sideset 7
      pressure 15.0
   sideset 12
      traction = 100.0 20.0 0.0
      coordinate 0
   sideset 13
      acoustic_vel 1.0
      function = 1
   sideset 14
      pressure = 1
      follower=yes
   node_list_file='force.nodes'
      force=1.0 0 0.
      scale = 100.
      function=2
END
```

Loads may be applied to node sets, side sets, node lists (see section 2.14.3) or the entire body (in the case of inertial loads). Pressure loads may be applied using side sets. The pressure is always normal to the surface. All loads applications are additive. Forces should not be applied to sidesets.

The components of each load specification are listed in Table 53. The syntax followed is to first define the region over which the load is to be applied (either **nodeset**, **sideset**, **node_list_file** or **body**). Each such region defines a *load set*. For each such definition, one (and only one) load type may be specified. However, any region definition (except **node_list_file**) may be repeated so that forces and moments may be applied using the same node set.

Following the definition of the load type, a vector (or scalar in the case of pressure loads) must be specified, except in the case of a thermal load, where no vector or scalar multiplier is needed. The vector is the load applied in the basic coordinate frame unless a coordinate frame is also specified (see section 2.28).

2.15.1 Scale Factors for the Load

The total load *on each degree of freedom* is the product of the load vector, the scale factor, and the nodeset distribution factor found in the **Exodus** file. For pressures and tractions, the load is also multiplied by the area of the face. Note that in some cases the nodeset

distribution factor may be zero.[2] In that case, the total applied force will also be zero.

2.15.2 Sideset Loading

The **pressure**, **acoustic_vel**, and **acoustic_accel** loadings may only be applied to side sets. The total pressure is the product of the scale factor, pressure (scalar) and sideset distribution factors. By default, pressure loads are not follower loads, i.e. pressures are applied in the direction of the undeformed element normal for the entire simulation. The **follower** keyword may be applied to user defined functions if a follower load is required. See section 2.15.5 for follower stiffness specification.

If the pressure loading is NOT normal to the sideset, the **traction** capability should be used. NOTE: Pressure will act on a surface in a compressive sense, while a traction can be specified as any vector which will act on the **sideset** specified in the direction given by the triple values specified after **traction**. Also, traction loads are applied on the faces of the shell elements in a piecewise manner, i.e., the traction load acting on a face of the element is assumed constant. If the distribution factors on the nodes of the element vary, the average of the load (element per element) is assumed.

Traction loads may be specified in either the *local* element coordinate frame, or in a coordinate frame projected onto the surface.

The *local* element coordinate system is used if no coordinate frame is specified in the input. This implies that there can be a mesh dependence on which direction the forces will be applied. The third component of the vector will always correspond to the surface normal (and hence will be applied as a pressure), but the first two components correspond to the two surface tangent vectors, which depend on the local element node numbering. Thus, some trial and error may be needed to determine which directions need to be specified in the traction loading section in order to get the forces in the right direction. We highly recommend using a coordinate projection.

If the analyst provides a coordinate frame with the traction definition, then that frame is projected onto the surface of each element. Figure 21 illustrates that projection. Like the local element frame, the third coordinate of the traction is always normal to the surface.[3] [4]

[2] Because the nodeset distribution factors are part of the **Exodus** file, and may be difficult to check, errors in the distribution factors are very common. Analysts are urged to be very careful to examine the distribution factors.

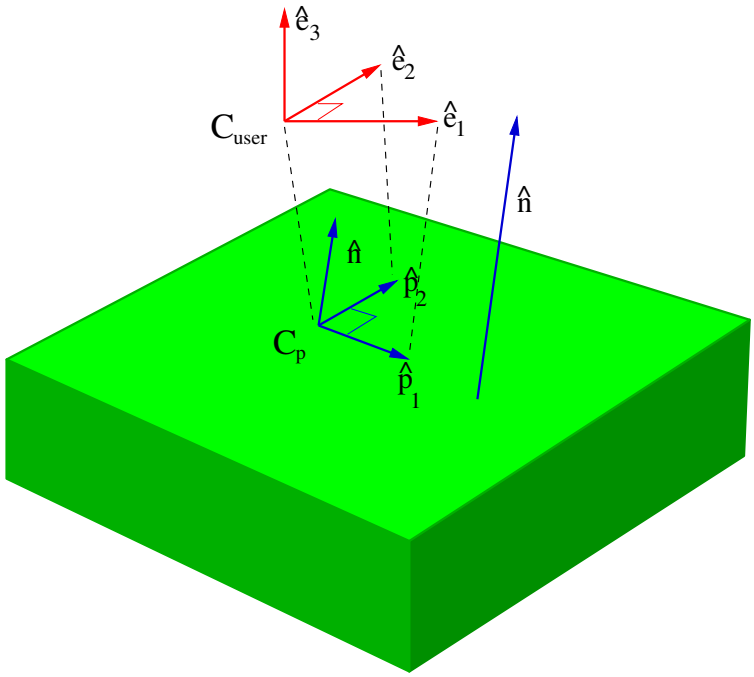
[3] This transformation is singular when $\hat{\boldsymbol{e}}_2 \times \hat{\boldsymbol{n}}$ is zero. Near that location, the transformation is modified.

$$\begin{aligned} \vec{\boldsymbol{p}}_2 &= \hat{\boldsymbol{n}} \times \hat{\boldsymbol{e}}_1 \\ \vec{\boldsymbol{p}}_1 &= \vec{\boldsymbol{p}}_2 \times \hat{\boldsymbol{n}} \end{aligned}$$

[4] This transformation is dependent on the direction of the element normal, $\hat{\boldsymbol{n}}$. If $\hat{\boldsymbol{e}}_3$ is in the opposite direction of $\hat{\boldsymbol{n}}$, the $\hat{\boldsymbol{e}}_2$ direction will be preserved, but the $\hat{\boldsymbol{p}}_1$ direction will be the opposite of $\hat{\boldsymbol{e}}_1$. This preserves the right hand rule.

Section	**Keyword**	**Parameters**
Region *(defines application area)*	body nodeset node_list_file sideset	- *id* *filename* *id*
Load Type *(defines application method)*	force moment gravity pressure point_volume_vel point_volume_accel acoustic_vel acoustic_accel traction thermal_load energy_load	*val1 val2 val3* *val1 val2 val3* *val1 val2 val3* *value* *value* *value* *value* *value* *val1 val2 val3* - -
optional specifications		
Coordinate Frame *(for vector loads only)*	coordinate	*id*
Scale Factor Multiplier	scale	*val1*
Function *(Required for transient analysis)*	function	*id*
Follower *available with user functions, 2.29.12*	follower	*yes/no*

Table 53: Load Specification Keywords



Consider a user defined coordinate frame, C_{user} defined by the basis vectors,

$$(\hat{\boldsymbol{e}}_1, \hat{\boldsymbol{e}}_2, \hat{\boldsymbol{e}}_3)$$

A surface normal, $\hat{\boldsymbol{n}}$, is defined by the element normal. The user defined coordinate frame is projected onto the surface as follows.

$$\begin{aligned} \vec{\boldsymbol{p}}_1 &= \hat{\boldsymbol{e}}_2 \times \hat{\boldsymbol{n}} \\ \vec{\boldsymbol{p}}_2 &= \hat{\boldsymbol{n}} \times \vec{\boldsymbol{p}}_1 \end{aligned}$$

When normalized, $(\hat{\boldsymbol{p}}_1, \hat{\boldsymbol{p}}_2, \hat{\boldsymbol{n}})$ form the basis for a coordinate frame, C_p, on the surface of the element in which to apply the tractions.

Figure 21: Coordinate Frame Projection for Tractions

2.15.3 Spatial Variation

Variation of the load over space is accomplished using node set or side set distribution factors.[5] If these are provided in the **Exodus** file, the load set is spatially multiplied by these factors. The total loading is the sum of the loads for each load set summed over all the load set regions.

2.15.4 Required Section

When prescribed accelerations are applied in the **boundary** section (2.14), they induce a load on the structure. In these cases the **loads** section may serve no purpose, unless an additional external load is applied. In these cases, an empty **loads** block is still needed in the input file. An error is generated if the input file has no **loads** section.

2.15.5 Follower Stiffness

This section allows the user to specify a follower stiffness corresponding to a an applied pressure load. When using follower loads, pressure loads applied to structures will "follow" the structure during deformation, always remaining normal to the surface where they are applied. As such, the applied force due to a pressure load depends on the deformed state,

[5] User defined functions may also be used. See section 2.29.12.

and this induces a follower stiffness matrix that contributes to the overall stiffness matrix of the structure.

The boundary where the pressure is applied is specified with a sideset. Also, the magnitude of the applied pressure field must be specified, as shown in the example below. The follower stiffness matrix scales linearly with the magnitude of the applied pressure.

```
LOADS
  sideset=1
  pressure = 10.0
  follower=yes
END
```

In the above example, sideset 1 is used to denote the surface where the pressure is applied. The parameter "pressure" specifies the magnitude of the applied pressure field.

2.15.6 Acoustic Loads

The **acoustic_vel**, **acoustic_accel**, **point_volume_vel**, **point_volume_accel** loading conditions are specifically designed for acoustic elements, and thus may only be applied to acoustic elements. In all cases, a time function is required that defines either the time or frequency dependence of the loads.

The **acoustic_vel** and **acoustic_accel** keywords specify the fluid velocity and fluid acceleration in the normal direction of the element faces in the sideset, respectively. Note that these are the counterparts to the **pressure** load for structures in the sense that they are Neumann boundary conditions.

We note that the **acoustic_vel** and **acoustic_accel** approaches should yield the same acoustic response, provided that the **acoustic_vel** time function is precisely the time integral of the **acoustic_accel** function. This time integration must include the constant of integration. If the two time functions for **acoustic_vel** and **acoustic_accel** are complementary in this way, the acoustic pressure output from these approaches will be the same up to first order. They are not exactly the same since the time derivative of velocity potential is needed to generate the acoustic pressure for output, and that time derivative is only first-order accurate.

An example of the **acoustic_vel** keyword is given below.

```
LOADS
    sideset 1
        acoustic_vel = 1.0
```

```
            function = 1
    END
```

In this case, sideset 1 is given a prescribed normal velocity of amplitude 1, with a time dependence given by function 1.

Currently, a given load case cannot contain both an **acoustic_vel** and an **acoustic_accel** input. Only one or the other can be specified in a given load case, though for a multicase solution the **acoustic_vel** and **acoustic_accel** inputs could be present in separate load cases. We also note that for coupled structural acoustics, only the **acoustic_vel** keyword is applicable. For analysis involving only acoustic elements, either keyword can be used.

The **point_volume_vel** and **point_volume_accel** keywords prescribe an acoustic point source on a nodeset. This force is the product of the fluid density with the first and second derivatives, respectively, of volume of the source. The **function** for the point source contains the time history of the first (for **point_volume_vel**) and second (for **point_volume_accel**) time derivative of volume.

Since the code scales by density in the internal calculations, there is no need to multiply the time history of volume by density to get the acoustic force. Thus, for point sources the **scale** parameter is typically set to 1.0, unless a direct scaling is desired. The units of the input time functions for **point_volume_vel** and **point_volume_accel** are volume per unit time and volume per unit time squared, respectively. The density need not be multiplied by these functions, since the code is already dividing by density internally (see the theory notes on structural acoustics for a more detailed discussion.)

Currently, the point acoustic source is only implemented for the time domain (transient) calculations, but we expect it to be extended to frequency response methods in the near future.

2.15.6.1 Example

```
LOADS
   nodeset 1
   point_volume_accel = 1.0
   function = 1
END

FUNCTION 1
   type LINEAR
   name "volume_acceleration"
   include inc/volume_acceleration.inp
END
```

In this case, nodeset 1 would consist of a single node, and the file "volume_acceleration.inp" would contain the second time derivative of volume velocity of the source, with units of volume per time squared. Note that the amplitude of the point source is taken to be 1.0, and that it does not include the density multiplier.

The sign conventions of the **acoustic_vel**, **acoustic_accel**, **point_volume_vel**, and **point_volume_accel** keywords are important. For the **acoustic_vel** and **acoustic_accel** cases, the equations of motion are given by,

$$\frac{1}{c^2}\ddot{p} - \Delta p = -\int_\Gamma \rho q(a \cdot n) d\Gamma \tag{34}$$

or, in discrete form,

$$M\ddot{p} + Kp = f \tag{35}$$

where ρ is the density, q is the surface shape function, a is the acceleration vector on the surface, n is the normal to the surface, and Γ is the portion of the surface where the loading is defined. M, K, and f are the mass, stiffness, and discrete force vectors. We denote $a \cdot n = a_n$ as the normal component of acceleration. We also note that this force has a negative sign in front of the integral, which comes from the variational formulation. This implies an inverse relationship between surface acceleration and acoustic pressure. Thus if the acceleration is oriented in the same direction as the normal, then a_n will be positive, and thus the total force vector will be negative. Intuitively, this makes sense, since if the acceleration is in the same direction as the surface normal, mass will be ejected from the acoustic space, causing a decrease in pressure. Conversely, if the acceleration is oriented in the opposite direction as the surface normal, then a_n will be negative. This will cause the total force vector to be positive, resulting in a positive pressure. This makes sense, since in this case mass will be added to the acoustic space, causing an increase in pressure.

For the **point_volume_vel** and **point_volume_accel** loadings, the equations of motion are given by

$$\frac{1}{c^2}\ddot{p} - \Delta p = \rho \frac{\partial^2 V}{\partial t^2} \delta(x - x_0) \tag{36}$$

or, in discrete form,

$$M\ddot{p} + Kp = f \tag{37}$$

where $\frac{\partial^2 V}{\partial t^2}$ is the second derivative of the volume change with respect to time, and $\delta(x - x_0)$ is the Dirac delta function that makes the term zero everywhere except where $x = x_0$. We note that V is the volume of fluid added to the surrounding acoustic space, *not* the volume of the point source per se. Thus, the sign of the acoustic pressure will be related to the sign of $\frac{\partial^2 V}{\partial t^2}$. A positive $\frac{\partial^2 V}{\partial t^2}$ would result in a positive acoustic pressure, implying that fluid mass is added to the surrounding acoustic space. Conversely, if $\frac{\partial^2 V}{\partial t^2}$ is negative, mass will be subtracted from the acoustic space, and thus a negative acoustic pressure will result.

Although the previous examples involved time functions that did not very spatially, the acoustic loadings can be used with spatially-varying time functions. This is accomplished

using the **ReadNodal** and **ReadSurface** functions which are described in more detail in sections 2.29.9 and 2.29.11.

2.15.7 Thermal Loads

The **thermal_load** option is used in conjunction with a spatial temperature specification for the structure. The temperature distribution can either be specified via the input **Exodus** file, or on a block-by-block basis, as described below. Based on the temperature distribution, a thermal load is computed and then applied to the structure.

If the solution method is selected to be statics, the **thermal_load** option will provide the thermal load necessary to solve the thermal expansion problem. If the solution method is transient dynamics, the same thermal load will be applied as in the statics case, but modulated by the function that is specified below the **thermal_load** keyword. This corresponds to a thermal shock analysis. Thus, for a transient dynamics problem that includes damping, and with a function that is equal to 1.0 for all time, the transient analysis would eventually converge to the same solution as obtained in the statics analysis, which would be the solution from a classical thermal expansion analysis. On the other hand, for a transient dynamics problem with a **thermal_load** in which the associated time function is not equal to 1.0, the thermal load will be scaled according to that time function. For example, in the case of a mesh that has block-by-block values of temperature *T_current* specified in the input deck, and a thermal load function that ramps up from zero to one, the actual thermal load applied to the structure will be multiplied by that time function. In this case, the full thermal load will only be seen after the ramp in the time function is completed.

If it is desired to apply a thermal preload to a structure, we generally recommend using a **static** analysis rather than a **transient** analysis, since in the latter case the preload that will be computed will be a dynamic preload that will oscillate around the static preload solution. If damping is used, this dynamic preload will converge to what would be obtained from using a **static** analysis. However, in some cases such as when rigid body modes are present, a **transient** analysis may be the only option for applying the preload.

The temperature field can either be read from an **Exodus** file, which would typically be the result of a thermal analysis, or it can be specified on a block-by-block basis in the input deck. For temperature fields that change from element to element, the temperatures must be read in from the **Exodus** input file. For more uniform temperature distributions, it is more efficient to specify them block-by-block in the input deck. Note that when using thermal loads, the temperature data is expected to either be in the mesh (exodus) files, or specified using the input deck (i.e. block-by-block). We note that when temperatures are specified both in the **Exodus** file as well as on a block-by-block basis in the input deck, the input deck values take precedence.

Sometimes it is of interest to output the stress after a thermal load analysis. In this case, the stresses that are output to the **Exodus** file will be correct in the case of block-by-block temperature input, but they will *not* be correct if the temperatures are read in from the

Exodus file. This is due to a known bug in the way that thermal stresses are computed. Thus, if thermal stresses are needed, the only method that can currently generate them is by specifying the temperatures on each block in the input deck.

If temperatures are specified using the input deck, then each block must be given its own temperature. In the example below, there are 2 blocks, and each is given a different temperature.

```
BLOCK 1
  material 1
  T_current 100
END
BLOCK 2
  material 2
  T_current 200
END
```

Note that if $T_{current}$ is specified for some blocks and not for others, the code will error out.

When temperatures are read in from the **Exodus** file, the material properties can be specified as temperature-dependent. This implies that each element will have different material properties. More details are given in the section on temperature-dependent material properties.

For thermal statics or thermal transient analysis, each material block must be given two additional parameters, the reference temperature, $T_{\text{ref}} = \texttt{Tref}$, and the coefficient of thermal expansion, $\alpha_t = \texttt{alphat}$. These parameters are defined via the thermal strain, which is given by

$$\epsilon_{thermal} = \texttt{alphat}\,(T_{\text{current}} - T_{\text{ref}}) \tag{38}$$

An example is the following.

```
MATERIAL 1
        E 10e6
        nu 0.3
        tref 300.0
        alphat .001
        density 0.1
END
```

The defaults for `tref` and `alphat` are both 0.0. This implies that if they are not specified, then the material will not contribute to the thermal analysis (see equation 38).

Note: currently, isotropic thermal strain is supported for isotropic, isotropic viscoelastic, and anisotropic materials.

Shell and beam type elements are not supported in thermal analysis. If used in conjunction with a thermal load, their contributions to the thermal expansion analysis will be ignored. This shortcoming is expected to be corrected in a future release.

The default **Exodus** file labels for the temperatures are shown in the table below. This is the default variable format that **Sierra/SD** looks for. However, it is also possible to read in element variables and variables of different names. Using the keyword **thermal_exo_var** in the **PARAMETERS** section (2.3) allows you to specify the name of the temperature variable in the **Exodus** file. **Sierra/SD** will first look for a nodal variable of this name, but if there isn't one, it will look for an element variable. If no element variable is found by the given name, an error will be generated.

Name	Definition
`TEMP`	the nodal temperature

The **thermal_load** load case can be used in a multicase solution method. In that case, the stresses and internal forces from the thermal analysis are used as initial conditions for the next case. For example, for a fixed-fixed cantilever beam that is subjected to a uniform temperature increase, the beam will undergo a stretch due to the thermal static analysis, and will have residual stresses. If this beam were then subjected to an eigen analysis in a subsequent case, the modes would be modified due to the geometric stress stiffening. Conversely, for a fixed-free beam, there would be no residual stresses and thus no effect on subsequent cases. Note that the displacements from thermal analysis are not carried over to subsequent cases. Thus, to get the total displacement from a thermal analysis followed by transient, one would need to add the displacement results from the two cases separately.

The **thermal_time_step** keyword must be specified in the **PARAMETERS** block, to specify which time step of the previous thermal analysis should be used to extract temperature data. The following gives an example.

```
PARAMETERS
  thermal_time_step 10
  thermal_exo_var "TEMP"
END
```

The **Exodus** files can contain multiple time steps of temperature data. The user can select which time step is to be used for defining temperature data in **Sierra/SD**, using the keyword **thermal_time_step**. In this example the tenth time step will be read in from the **Exodus** file. The default value for the **thermal_time_step** is 1.

The following is an example of some of the input for a thermal statics analysis.

```
SOLUTION
  statics
```

```
END

PARAMETERS
  thermal_time_step 10
END

LOADS
  body
    thermal_load
END
```

2.15.8 Energy Deposition Input and Loads

Input from energy deposition are very similar to thermal loads (section 2.15.7). These loads are specified when energy is deposited directly in the structure as with an X-ray deposition. For consistency with other applications, the energy is defined as *specific energy*, i.e. the energy per unit mass. Such direct energy deposition is converted to a change in temperature after which thermal strains and loads are computed exactly as for the **thermal_load** approach.

Energy is converted to a change in temperature using the specific heat of the material (see section 2.27.9).

$$\tilde{E} = C_v \Delta T$$

where $\tilde{E}$ is the specific energy of the body, C_v is the specific heat capacity for constant volume, and ΔT is the change in temperature.

The energy load is specified using the keyword **energy_load**. All other parameters are identical to *thermal_load*. Note that by the nature of these loads there is often an exponential decay in energy as a function of depth. For this reason, it is very advantageous to specify the loads at Gauss points, particularly when using higher order elements.

Energy may also be used as an input for thermally dependent material properties. To ensure that the energies are converted to temperature before determining the material properties, identify the variable name from the exodus file with the **energy_exo_var** and **energy_time_step** keywords, rather than the **thermal_exo_var** and **thermal_time_step** keyword.

2.15.9 Consistent Loads

The loads for all of the 3-D and 2-D elements are calculated in a consistent fashion when a pressure load is applied. For more details on the implementation, see the programmer's notes. It is very important that consistent loading be used. This is especially true for shell elements where the consistent loading is required to properly apply rotations.

2.15.10 Pressure_Z

Depth dependent pressure loads may of course be applied using a user defined function. To simplify this loading condition, depth dependent pressure may also be applied using the **pressure_z** keyword. An example is shown in Figure 22. This loading is applied only in the basic coordinate frame, and the analyst must specify that the pressure is either "below" or "above" an offset to the coordinate axis. The pressure is always proportional to the depth. In the example of Figure 22, the pressure is zero at $x = 5$, 10 at $x = 4$, 20 at $x = 3$ and so on. At depths above the "waterline", the pressure is zero. Any of the basic coordinate directions (x, y, or z) may be used as a reference.

```
// depth dependent pressure for a waterline at x=5.
LOADS
     sideset 2
          pressure_z 10.0 below x = 5
     sideset 20 // air, but silly
          pressure_z 1e-4 above x = 5
END
```

Figure 22: Depth Dependent Pressure Load Example. This load section applies a pressure to sideset 2 which is proportional to the distance below $x = 5$.

2.15.11 Static Loads

Static loads only require the definition of the load region and load keyword (e.g. force) with it's accompanying parameters. However, a function (including a user defined function, see 2.29.12) may be used as well. In this case, the function will be evaluated at time $t = 0$.

This changed following release 2.3. Previous versions did not allow any function definitions for static loads, and any loads with temporal functions would not be applied to static analysis.

2.15.12 Time Varying Loads

Additional options provide the capability of varying the load over time. The **loads** options include,

- **scale** with one parameter provides a scale factor to be applied to the entire load set. Only one scale may be provided per load set.
- **function**. A time varying function may be applied by specifying a function ID. Only one function may be applied per load set. The function is defined in the **function** section (see section 2.29 on page 205). The loads applied at time t for a particular load set will be the sum of the force or moment vectors summed over the nodes of the region and multiplied by the scale value and the value of the time function at time t.

NOTE:
If no function is applied for a particular load, then the function is defined as 1.0 for all time. All loads will be applied to the transient solution, regardless of whether an explicit time function is defined.

This is in marked contrast to the loads application defined up to release 2.3 of **Sierra/SD** *, where transient loads required an explicit temporal function definition, and static loads prohibited it.*

2.15.12.1 Reading Loads from Exodus Data Loads may be read in from previous analyses when stored in the input exodus file. These are read using an appropriate function. See sections 2.29.9, 2.29.10, and 2.29.11 for functions which read data on nodal values, on a node set and on a surface respectively.

2.15.13 Random Pressure Loads

Input for random loads can be complicated, though the loads are not uncommon and are important for many applications.[1] This type of random pressure loading is developed for use of direct transient loading typical of a turbulence load on a hypersonic vehicle. Throughout the development, we maintain a concept of flow direction, and correlation distances that may be different in flow and transverse directions. By computing the random pressure fields as part of the time evolution, we avoid the need to compute these complex quantities before the run. The approach requires a linear solve at each solve to compute the loads.

[1] A hypersonic vehicle is a prime example of a random loading. Turbulence provides a time varying loading which has a limited spatial and temporal correlation on the surface of the hypersonic vehicle.

The most general type of input is the correlation matrix, which is the inverse Fourier transform of the spectral density matrix. The **RandomPressure** load option provides a simplified means of specification of the loading. The material in this section is consistent with and builds on section 2.29.6.

Section 4.4 in the theory manual[24] details the approximations involved in the implementation. These approximations are summarized in Figure 23.

The simplified correlation matrix is not general, but may be useful for a large class of problems. It has the following limitations.

1. The system must be time stationary.
2. The correlation function must be separable (a product of temporal and spatial correlations).
3. The same PSD shape must apply throughout the entire hypersonic vehicle body. The PSD may be scaled as a function of z, but there may be no change in the shape.
4. The PSD must have some sort of cutoff. The time integration must occur above this cutoff frequency.
5. By default, the temporal function is represented by a *sinc* function. This may be replaced by a user defined temporal function.

Figure 23: RandomPressure Loading Approximations

The random loading is a component of the loads section. An example is shown here, and described in Table 54.

```
LOADS
   sideset 22
      randompressure
         correlation_length_z = 2.0  // required
         correlation_length_r = 0.67 // required
         cutoff_freq  = 16.8         // required
         correlation_function = 20   // defaults to sin(x)/x
         psd_scale_function = 10     // defaults to Sigma=1
         ntimes = 5                  // defaults to 5
         coordinate 1                // defaults to basic frame
         MinimumNodalSpacing = 1.0e-5 // defaults to 1.0e-8
         numberOfInitializationSteps = 100 // defaults to 5
END
```

Details for the parameters to the correlation matrix input are described below.

Parameter	Type	Default	Comment
correlation_length_z	real	required	spatial decay in flow direction
correlation_length_r	real	required	spatial decay orthogonal to the flow
cutoff_freq	real	required	cutoff frequency
correlation_function	int	$\frac{\sin(t\omega_c)}{t\omega_c}$	
psd_scale_function	int	$\Sigma(z) = 1$	
ntimes	int	5	
coordinate	int	0	defaults to basic frame
MinimumNodalSpacing	real	$1.0e-8$	smallest allowable inter-node spacing
Random_Seed	int	ignore	random number seed
numberOfInitializationSteps	int	5	iterations to improve initial spatial distribution

Table 54: Random Pressure Inputs

correlation_length_z Spatial decay in the flow direction, L_z. The flow direction is the Z axis of the coordinate frame. The correlation function $C(\Delta Z)$ is proportional to $\exp(-\Delta Z/L_z)$, where ΔZ is the distance between two points in the flow direction.

Correlation_Length_R Correlation_length_r is the spatial correlation distance in the radial or transverse direction. The correlation function is proportional to $\exp(-\sqrt{(\Delta x^2 + \Delta y^2)}/L_r)$.

Cutoff_freq The cutoff frequency, F_c is very important to the operation of the random-pressure algorithm. No energy may be found in the PSD above this frequency. The time integrator may not sample the system lower than this frequency, i.e. $dt < 1/F_c$.

NTIMES The matrix is proportional to the number of time values assembled, and affects the interpolation as described in equation 4.49. Typically only a small number of terms are required. Note that there are $2 * NTIMES + 1$ terms in the sum, and the dimension of the correlation matrix grows commensurately. The number may depend on the interpolation time step and on the shape of the PSD. Default=5 (which produces 11 terms in the sum).

CORRELATION_FUNCTION The temporal time function, whose argument is $(t_1 - t_2)$. By default this function is $\sin(x)/x$, with $x = \pi F_c(t_1 - t_2)$. It must be an even function of the argument.

PSD_SCALE_FUNCTION provides a means of scaling the power spectral density as a function of flow direction. This type of input requires that the PSD have the same shape at all locations, but the value may be scaled. Scaling the PSD effectively scales the standard deviation of the pressure. Default is no scaling. The function must be positive for all values of the coordinates.

COORDINATE is an optional coordinate frame that is used to define the flow direction. The z component of that frame is the direction of flow. By default, the basic frame is used.

MINIMUMNODALSPACING Some models can contain co-located nodes on the surface where the random pressures are to be applied. This can cause the correlation matrix to be singular, since the repeated nodes would result in two identical rows in the correlation matrix. The **MinimumNodalSpacing** keyword allows the analyst to specify the smallest inter-node spacing (absolute) that is allowed on the surface where the random pressure is being applied. Any nodes that are closer than that tolerance will be treated as identical in the correlation matrix manipulations. The **Exodus** file and corresponding nodal output will not be changed. This will avoid a singular correlation matrix, but does not alter the mesh database.

NUMBEROFINITIALIZATIONSTEPS The initial spatial pressure distribution may appear unrealistically correlated. This problem becomes more likely with mesh resolution. The issue is mitigated by taking a few steps of the stochastic iterative process. If issues are evident with the initial distribution, this parameter could be increased. The cost of increasing this parameter is comparable to the cost of an implicit time step. Default=5 (values lower than 5 are not recommended).

OMEGA_C Deprecated. Use Cutoff_freq.

ALPHA_Z Deprecated. $\alpha_z = 1/L_z$.

BETA_T Deprecated. $\beta_t = 1/L_R$.

The computation of the random pressure loads depends on matrix factorizations (see 4.4 of theory manual[24]). However, the Cholesky matrix factorizations are defined only if the correlation matrix is (numerically) nonsingular. At this time, the code stops with an error if this occurs. A common cause of this error is using too many time steps ntimes with too small a time step. For this reason, the condition number of the temporal correlation matrix is always evaluated, and, if it is singular, the cutoff frequency is decreased. In this case the warning message

```
Singular temporal correlation matrix
Increasing Delta_T to ...
```

will be printed in the result file for processor 0. Another source of ill conditioning is the use of very large correlation lengths correlation_length_z or correlation_length_r, or a very fine mesh.

For this reason inverse condition number estimates are printed in the result files. An inverse condition number is the relative distance to a singular matrix, and is denoted Rcond, for reverse condition number. In double precision, an Rcond below 10^{-12} indicates that the factorization may fail. The precise statements in the results files are

```
TemporalCorrelationMatrixRcond = ...
Estimated SpatialCorrelationMatrixRcond = ...
Estimated CorrelationMatrixRcond = ...
```

2.15.14 Frequency Dependent Loads

Frequency dependent loads may be applied for frequency response analysis. The real part of these loads is applied exactly as above with the understanding that the functions referenced now apply to frequency not time. Frequency dependent loads may include an imaginary component. This is done by prefixing the load types listed above by the letter *"i"*. Thus the imaginary part of the load uses these load types.

For Complex Analysis

Option	Parameters
iforce	*val1 val2 val3*
imoment	*val1 val2 val3*
igravity	*val1 val2 val3*
ipressure	*val1*
itraction	*val1 val2 val3*

A function should be associated with each such load. An example follows.

```
LOADS // example for FRF analysis
  nodeset 1
     force=1 0 0      // the real part of the load
     function=11
  nodeset 2
     iforce=1 0 0     // the imaginary part of the load
     scale .707
     function=12
END
```

2.15.15 Rotational Frames

Often when analyzing rotating structures, it is convenient to perform the analysis in the rotating frame where the structure is not undergoing large displacement. Analysis in that frame introduces "fictional" or "pseudo" forces with centrifugal,[2] Coriolis and Euler contributions. These are termed "forces", but the contributions are introduced from operating in a noninertial coordinate frame. For the theory, see section 4.2 in the theory manual. The associated keywords are found in Table 55.

The Galerkin framework used for finite elements, introduces matrices associated with these pseudo forces. In addition to the standard mass and stiffness matrices that arise in linear

[2]There is often confusion about the description of the "centrifugal" or "centripetal" term. The *centripetal* force is a real force applied in the inertial coordinate frame which causes an object to travel in a circular path. The *centrifugal* force is the pseudo-force that appears from inertial terms in a rotating coordinate frame.

Option	Parameters
angular_velocity	*vel1 vel2 vel3*
angular_acceleration	*accel1 accel2 accel3*
coordinate	coordinate id

Table 55: Rotating Frame Parameters

```
LOADS
   body
     angular_velocity = 0.0 2.0 0.0
     coordinate = 1
   body
     angular_velocity = 0.0 0.0 3.0
     coordinate = 3
END
```

Figure 24: Application of centrifugal and Euler forces. The loads above apply an angular acceleration of 3 radians/s^2 in the Z direction of coordinate frame 3, and an angular velocity of 2 radians/s in the Y direction of coordinate frame 1. Angular acceleration is applicable only in statics.

structural dynamics, force-based matrices are also common. These include follower stiffness matrices from applied pressures, and Coriolis/centrifugal matrices in rotating structures.

Figure 24 provides the corresponding **Sierra/SD** input for a rotational load applied to a body. The centrifugal stiffness and Coriolis coupling matrices are both derived from the rotational velocity of the structure, which uses the keyword **angular_velocity**. The vector angular velocity components are specified after the **angular_velocity** keyword.

An angular acceleration, $\dot{\Omega}$, may also occur, as when an aircraft carrying a weapon makes a rapid course correction. This angular acceleration results in a pseudo-force, called the Euler force, that is tangent to the angular acceleration vector. Application of angular acceleration is restricted to linear and nonlinear statics analysis in Sierra/SD.

In many instances, the angular acceleration and angular velocity are applied independently for static loads analysis. The may seem a bit of a contradiction. But it is useful and very similar to the static loads analysis of a rocket that provides envelope survivability information during launch.

Left Hand Side contribution

Angular velocity introduces both left hand side matrices and right hand side force vectors. The algebraic expression for dynamics can be written as follows.

$$(K_m + K_g + K_{cen})\, u + (C + C_{cor})\, \dot{u} + M\ddot{u} = f_{extern} + f_{cen} \tag{39}$$

where,

K_m	is a material matrix,
K_g	is the geometric stiffness matrix correction,
K_{cen}	is the centrifugal softening term,
C	is the damping/coupling matrix,
C_{cor}	is the Coriolis coupling matrix,
M	is the mass matrix,
f_{extern}	is the external force vector,
f_{cen}	is the centrifugal force, and
u	is the displacement.

With the exception of K_g, all of the matrix terms are constant, depending only on the geometry and the elements. The Coriolis and centrifugal terms are also independent of displacement, u, though they depend on Ω.

For linear analysis (both linear statics and linear transient dynamics), the geometric stiffness terms is zero. However, since this term depends on stress, which is proportional to displacement, the geometric stiffening is typically proportional to the square of the angular velocity. As the geometric stiffening is typically of the same magnitude as centrifugal softening (also proportional to Ω), confusion can arise.

In multicase analyses, the matrices are typically generated only once; exceptions occur for nonlinear solutions and for the **tangent** method. It is recommended that linear solution cases include an update to the tangent stiffness matrix as part of a multicase solution. An example is shown in Figure 25.

Limitations

There are a number of limitations for the rotational frames implementation.

1. Static analysis appropriately applies the centrifugal and Euler forces. The left hand side matrix for geometric stiffness is only properly updated if the **tangent** step is applied.
2. In a single case solution, quadratic eigen solutions will include Coriolis and centrifugal terms if angular velocity is specified in the **LOADS** section. This is the case even though there is no true load for QEVP.

```
Solution
   case s1
      statics
        load=1
   case up
      tangent
   case s2
      statics
        load=1
End
```

Figure 25: Example using Tangent Update

3. Currently, QEVP solutions can be computed for rotating structures only if there are no rigid body modes in the structure. An example is shown in Figure 26. In this case, a static preload with a rotational load is computed, followed by a tangent update, and then followed by a QEVP analysis. This type of analysis would be useful for examining the effect of rotational loading on the modes of a structure. However, this will only work correctly if there are no rigid body modes in the structure. Future releases will allow for rigid body modes in these types of computations.

4. The user is limited to one rotational frame per analysis. In other words, all of the body must be rotating together. One could not model a helicopter in a fixed frame and the associated rotor in another.

5. Rotational loads applied in the rotating frame are linear loads, and do not require a follower keyword.

6. For transient dynamics, the time varying function must be 1.0.

7. Angular acceleration is only applicable to statics analyses.

8. Superelements do not retain full accuracy. It is recommended that interface dofs for superelements retain either 3 or 6 degrees of freedom.

9. The BOUNDARY section applies to all cases in a multicase solution.

NOTE:
A time varying function with magnitude 1.0 for the full time span should be used for time varying solution cases. Additional work would be required to apply general loading patterns.

Finally, for optimal accuracy, the user can update the tangent matrix at the expense of greater computational expense.

```
Solution
   case 'statics'
      statics
        load=1
   case 'up'
      tangent
   case 'qevp'
        qevp
        method=projection_eigen
        nmodes=100
        load=1
End
```

Figure 26: Example of using qevp for Tangent Update

2.15.16 Rigid Body Filter for Input

For some analyses, it is advantageous to remove all or some of the rigid body components of a solution. The input forces may be filtered so that only self-equilibrated forces are applied. The filter is applied using input in the **parameters** section (2.3). While the filter can ensure equilibrated loads, additional parameters may be required to help the linear solver address the singularity generated by floating structures. Typical input is provided here, with details in the appropriate sections. Note that when **FilterRbmLoad** is used, the **constrain_rbms** solver option must be specified as part of the **solver_options** block to signal to the solver now many rigid body modes are present. Currently, only **GDSW** supports selectively constraining rigid body modes. The **num_rigid_mode** parameter is now deprecated, but is still available for backward compatibility. The **FilterRbmLoad** parameter is currently only supported for transient[3] and static solution cases. For other solution cases this parameter will have no effect on the solution.[4] The similar capability for modal solutions is presented in section 2.1.38.

```
Solution
    solver=gdsw
    solver_options=gdswOptions
End

Parameters
```

[3] The FilterRbm option is only compatible with Newmark-Beta time integration. The generalized-alpha time integrator may be used, but the coefficient alpha_m=(2 rho-1)/(1+rho) will be set to zero, where rho is the integration parameter. The resulting integration scheme is still 2nd-order accurate. The rigid body motion equations will be integrated using the same scheme as the flexible body equations (see section 1.18 in the theory manual for details).

[4] Modal solutions, such as `modaltransient`, do not use FilterRbmLoad. However, see the HowTo manual for means of accomplishing the same process by direct use of the geometry rigid body modes.

```
    FilterRbmLoad=allStructural
    rbmtolerance=1e-10
End

solver_options gdswOptions
    constrain_rbms "x y z rotx roty rotz p"
End
```

The names of the rigid body modes to constrain are the same as setting boundary conditions 2.14. The 'p' keyword refers to the constant pressure in a structural acoustics problems, and the other six are the typical rigid body modes of a structure in Cartesian coordinates.

The **num_rigid_mode** keyword is *deprecated* in favor of the **constrain_rbms** keyword. Previous usage was:

```
Parameters
    FilterRbmLoad=allStructural
    rbmtolerance=1e-10
    num_rigid_mode 6
End
```

The names of the rigid body modes to constrain are the same as The range of the `num_rigid_mode` parameter is $\{0, 1, 6, 7\}$. The value 1 refers to a structural acoustics problem in which the acoustic region is floating. The value 6 refers to a structural problem in which the structural region is floating. The value 7 refers to a structural acoustics problem in which both the acoustic and structural regions are floating. If the parameter `num_rigid_mode` is parsed, Salinas calculates the corresponding rigid body modes, checks the residuals, and report a fatal error if the residual norms

$$\frac{\|K\Phi_r\|_2}{\|K_d\|_\infty \|\Phi_r\|_2}$$

are larger than `RbmTolerance`. The default `RbmTolerance` is 10^{-10}. Each module that needs the rigid body modes will recalculate them.

2.16 Load

Loading conditions *for all multicase solutions* are specified within the **load** section. See paragraph 2.1.1 for information on specifications for multicase solutions. The **load** section is *identical* to the **loads** section described in the previous paragraph (2.15), except the section begins with the **load**, and a load step identifier is required. The following example illustrates the required input.

```
LOAD=57
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      force = 0. -1 0
END
```

Unlike the **loads** section, there may be multiple **load** sections in the file, with each entry corresponding to an applicable step in the solution.

2.17 INITIAL-CONDITIONS

Initial conditions are specified via the **INITIAL-CONDITIONS** section. Currently, only velocity and displacement can be specified as initial conditions. The initial conditions can then be used in either an implicit or explicit transient analysis. Both linear and nonlinear transient are supported.

Three options are available.

1. Initial conditions are read in from the **Exodus** file.
2. Initial conditions are specified globally in the **INITIAL-CONDITIONS** section.
3. Initial conditions are specified on a block-by-block basis in the input deck.

An example of the first option (input from **Exodus** file) is given below.

```
INITIAL-CONDITIONS
   velocity=from_file
   displacement=from_file
END
```

In this case, **Sierra/SD** will read both velocity and displacement initial conditions from the **Exodus** file. The variable names on the **Exodus** file must be VEL for velocity and DISP for displacement. The full names of the nodal variables for displacement are ”DispX”, ”DispY”, ”DispZ”, ”DispRX”, ”DispRY”, ”DispRZ” and for velocity are “VelX”, “VelY”, “VelZ”, “VelRX”, “VelRY” ,“VelRZ”. Case is not significant, but all 6 components must be present for the desired conditions. If only velocity is to be specified as an initial condition, the syntax would be,

```
INITIAL-CONDITIONS
   velocity=from_file
END
```

An example of the second option is given below.

```
INITIAL-CONDITIONS
   velocity=1 0 0
END
```

In this case, the entire model is given an initial velocity of 1 in the x direction, and 0 for the y and z directions.

An example of the third option (block-by-block specification) follows.

```
INITIAL-CONDITIONS
   velocity=by_block
END

BLOCK 1
  velocity = 1 0 0
END
```

In this case, the velocity is read in from the input deck on a block-by-block basis. This simple example only has one block, which is given an x velocity. If more than one block is specified in the mesh, each block could have its own initial conditions. However, if two blocks share nodes and are given different initial conditions, then the results may be unpredictable, since the common nodes on the blocks would have conflicting initial conditions. Thus, we recommend the user verify that blocks are disjoint before specifying different initial conditions on a block-by-block basis.

Initial conditions are currently only implemented for transient analysis. They can also be used in multicase solutions, but they will only have an effect on the transient analysis that are in the multicase solution. For multiple transient analysis in a multicase, only the first transient analysis will use the initial conditions, since the subsequent transient cases would simply get their initial conditions from the previous case.

2.18 RanLoads

The **RanLoads** section is used to provide input information for spectral input to a random vibration analysis. In a random analysis, the output response relates to the input, as follows.

$$a_i(\omega) = \sum_{j,k} H^T_{ji}(\omega) S_{jk}(\omega) H_{km}(\omega) \tag{40}$$

where,

a_i is the output quantity at degree of freedom, i. For example, a_i may be the acceleration power spectrum, measured in $(in/s^2)^2/Hz$.

H_{ij} is the transfer function from input i to dof j.

S_{jk} is the input power spectrum. Typically this is in units of $(\text{force})^2/Hz$. It is dimensioned to the number of independent inputs.

The **RanLoads** section provides a specification for $S_{jk}(\omega)$. Note that this input will contain both a spatial and spectral component. In **Sierra/SD** , we require that each matrix element in the input power spectrum be expressible as a product of spectral and spatial components.

$$S_{ij}(\omega, x) = Y_i(x)Y_j(x)F_{ij}(\omega) \tag{41}$$

where Y_i is a spatial loading term associated with the ith row and column of S, and F is a spectral only matrix function.

The **RanLoads** section contains the following required keywords.

Parameter	Argument	Description
matrix	*Integer*	matrix-function identifier
load	*Integer*	row/column identifier

The **matrix** keyword identifies the appropriate **matrix-function** (see section 2.30). The matrix-function determines the dimensionality of the input (using the **dimension** keyword). It also determines the spectral characteristics of the load.

The spatial characteristics (which correspond to Y_i in equation 41) are determined in **load** sections within the **RanLoads** definition. There must be exactly as many **load** sections as the dimensionality of input. For example, if the S_{FF} matrix is a 3x3, then there should be 3 separate load sections. Each load section within the **RanLoads** block must be followed by an integer indicating to which row/column it corresponds. The details of each **load** section are identical to the over all **loads** section (see 2.15) except that no time/frequency function is allowed. Note that only one load is required per row of the S_{FF} matrix, but each *entry* of the matrix may have a spectral definition (identified by a real and/or imaginary **function**).

The following example illustrates the definition of a single input specification. The loading is scaled so that a 1000 lb mass located on the input point (in nodeset 12 here) is scaled to produce a unit g^2/Hz loading.

```
RANLOADS
   matrix=1
   load=1
     nodeset 12
        force=0 1 0
```

```
        scale 1.00e3  // needed to convert to g
        // loads input in lbs. The PSD is in g^2/Hz.
        // F = accel * mass
        //   = accel * (scale_factor)
        //   = accel * ((1000*.00259)*384.6)
END
```

Scaling the input force for a random vibration analysis can be confusing.[5] This is especially true since enforced acceleration cannot be used to apply the force. The example above applies to english units where a **wtmass** parameter has been applied. For SI units or other systems where wtmass=1, the force would need to be multiplied by g to apply the input as acceleration in g's.

The input acceleration may be examined by evaluating the output PSD at the input degree of freedom. This is done by putting the applied load set into the **frequency** section (2.11), and adding the **acceleration** keyword. The output is in the native units of analysis. For the example above, the output will be in $(in \cdot lbm/s^2)^2/Hz$, and must be divided by $(386.4)^2$ to convert to g^2/Hz.

2.19 Contact Data

Not truly functional at this time.

2.20 Tied Surfaces

Tied surfaces provide a mechanism to connect surfaces in a mesh that will always be in contact. Because the surfaces are always tied, the constraints may be represented by a set of linear multipoint constraints (see Appendix 3.36). Tied surfaces are also known in the literature as "glued surfaces" or as "tied contact". They are used almost exclusively to combine two surfaces of a mesh that have not been meshed consistently.

There are a number of ways of combining surfaces that have not been consistently meshed. The simplest method constrains the nodes of the slave surface to lie on the master surface. In this method, the constraint is called *inconsistent* because the mesh does not ensure that linear stress will be maintained across the boundary. The stress and strain in the region of the constraint will be wrong. However, loads are properly transferred across the boundaries, so a few element diameters away from the boundary, the stresses and strains should be approximately correct.

[5] Note that we are scaling the spatial forces, Y_i, which are combined as a product in equation 41. Thus the scale factor is linear in the load. The resulting input power spectrum, S_{ij}, will contain the square of the scale factor.

Tied surfaces can currently be specified for structural-structural interfaces, acoustic-acoustic interfaces, and structural-acoustic interfaces (i.e. wet interfaces). The syntax in the **TIED DATA** block is the same in all three cases. In the first case, the nodal displacements on the slave surface are constrained to lie on the master surface. In the last case, the nodal acoustic pressures on the slave surface are constrained to match those on the adjacent master surfaces. In the case of tied structural-acoustic interfaces, it is necessary to ensure a weak continuity of both stress and displacement (velocity) across the wet interface.[25,24] Also for tied structural-acoustic interfaces, we recommend that the acoustic surface be defined as the master (and hence should have its sideset number listed first in the input deck). Defining the structural surface as the master sometimes causes an error related to singular subdomain matrices.

We do allow mixing of tied surface cases in a given simulation. For example, one may have tied acoustic-acoustic and tied structural-acoustic data blocks in the same input file. However, it is necessary that each sideset be exclusively attached to either structural elements or acoustic elements. A single sideset cannot simultaneously contain both acoustic and structural elements. This does not restrict the types of analysis that can be done, but it may lead to more **TIED DATA** blocks. However, this extra input will reduce confusion and likely also reduce potential modeling errors.

There is also an additional option associated with the enforcement of transverse displacements. The keyword **TRANSVERSE** can be specified as tied, slip, or friction. The tied option is the standard inconsistent tied surface approach. The slip option only constrains normal degrees of freedom between the master and slave surfaces. In this option, the tangential degrees of freedom are free to slide. This would be the case if there was no friction between the surfaces. Finally, the friction option allows one to specify a simple friction model. This option is currently not supported. The default value of the **TRANSVERSE** keyword is "tied".

A note about gap removal. Currently, **Sierra/SD** will remove the gap on **TIED DATA** block if the tied contact is inconsistent, i.e., the **method** is not set to mortar or other options. To turn gap removal off, set the **gap removal** keyword in the tied data block to *off*. The gap removal algorithm will move the nodes from the slave surface (specified by the 2nd surface id in the tied data block) to faces on the master surface. The gap is provided by using ACME's gap and push back vector quantities. The output **Exodus** file (if created) will have the updated coordinates and not the original coordinates. The system matrices will also use the updated coordinates. The gap removal option is set to *on* (default) for inconsistent contact and does not work for any other method.

2.20.1 Mortar Methods

Mortar methods may also be used to tie the surfaces. This is currently under development, but some capability is available. Large tied surfaces using the mortar methods may have a very large number of fully coupled constraints which can overwhelm most parallel solvers.

The cost in computing the mortar contribution is higher than the *inconsistent* method, but the solution will typically be much better in the region of the constraint.

Two "flavors" of mortar methods are used. In both approaches, the slave surface is constrained to meet the master surface in an integral sense. Standard mortar methods are somewhat simpler, but can result in a constrained system which fully couples all the nodes of the slave surface with the nodes of the master surface. Dual mortar methods are much more friendly to the linear solver, as the constraint system decouples the slave nodes.

Mortar methods are specified in the TIED DATA block using the keyword **mortar**. To select the type of method, standard or dual, the keywords **MortarMethod=standard** OR **MortarMethod=dual** must be specified in the PARAMETERS block. If no method is specified, **Sierra/SD** defaults to the dual mortar method.[6]

2.20.2 Node to Face

Tied surfaces are specified by a listing of master and slave side sets. Any number of tied surfaces may be specified in the input, i.e. more than one tied surface section may occur in the input. Each tied surface section represents a *single* logical pairing of constraint side sets.

```
TIED DATA
    Surface 12, 18
    transverse slip
    search tolerance = 1e-7
    edge tolerance = 1e-8
    gap removal = on
END
```

In the example above, sideset 12 is defined as a master surface. Side set 18 is the slave surface. Each node in the slave surface is tied to the set of nodes in the corresponding element face of the master surface. The transverse degrees of freedom are allowed to slip in this example. If the **TRANSVERSE** keyword were omitted, standard tied surfaces would be used.

Tied surfaces use a node-to-face search algorithm. In this algorithm, the "search tolerance" represents the normal distance from a node on one surface to a corresponding face on the other. Thus, the search tolerance will typically be quite small and represents the amount the two surfaces may not be coincident. This is in contrast to a node-to-node search, where the "search tolerance" represents a search radius. See Figure 27.

[6] There is no means of applying standard mortar methods to some interactions, and dual mortar methods elsewhere.

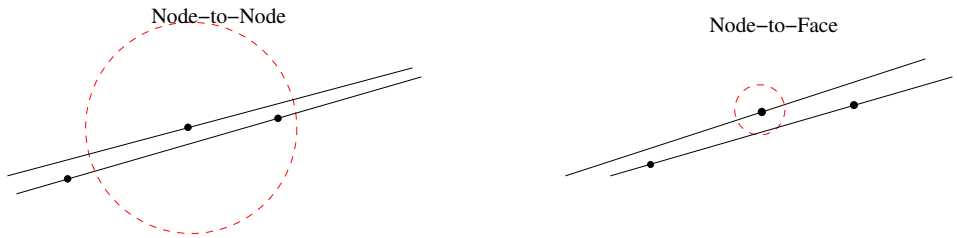


Figure 27: For node-to-node searches the *search tolerance*, must be large enough to capture nearby nodes. For node-to-face searches (as used in tied surfaces), it should only capture the nearby surface.

Special care should be used when using the "edge tolerance". If this tolerance is too large, non-intuitive interactions can be created.

Note: The current implementation ties a master and slave surface that are **face** *connected only. We have not implemented or tested a capability to tie the* **edges** *of shells.*

The relevant parameters for tied surfaces are shown in Table 56.

Smoothing parameters may be needed to control smoothing of the normal. Figure 28 illustrates the normal definitions on a faceted surface. The discontinuity in normals can be an important consideration on curved surfaces where faceting affects tangential sliding. Smoothing parameters are illustrated in Figure 29 and include the following.

smooth angle If an angle between two faces exceeds this value (in degrees), then the angle is considered to be "sharp", and no smoothing is done. Default is 30°.

smoothing distance If the *relative* distance from a node exceeds this value then no smoothing is done. This is a number between 0 and 1. At small values, normals are smoothed only very near the node. As the distance goes to 1, all normals are smoothed. Default=0.5.

smoothing resolution The resolution method can be either node based, or edge based. This may be needed to control smoothing on edges that include both a sharp and a non-sharp edge. Default=node.

2.21 Contact Normals

For all the contact type interactions, including tied surfaces and tied joints, we use the ACME package to search for the interactions.[26] The algorithms used restrict the search

Table 56: Tied Surface Parameters

Parameter	type	description
Name	*String*	name of tied data block, useful for diagnosing error messages defaults to tied
Surface	*integer pair*	master and slave sideset separated by comma or space
Search Tolerance	*Real*	face normal of search tolerance defaults to 1e-8
Edge Tolerance	*Real*	search tolerance beyond an edge facet defaults to 1/10 search tolerance.
Newton Tolerance	*Real*	convergence tolerance used in ACME defaults to 1e-12
Interaction	*String*	node-to-node *(not supported)* node-to-face *(default)*
Method	*String*	inconsistent *(default most solvers)* mortar *(default for CF solver)*
Transverse	*String*	tied *(default)* slip *(transverse displacements can slip)* friction *(currently not supported)*
Gap Removal	*String*	Yes *(default: for collocation/inconsistent tied contact only)* No
smooth angle	*Real*	maximum angle for smoothing (def=30)
smoothing distance	*Real*	relative distance for smoothing
smoothing resolution	*String*	"node" or "edge" based

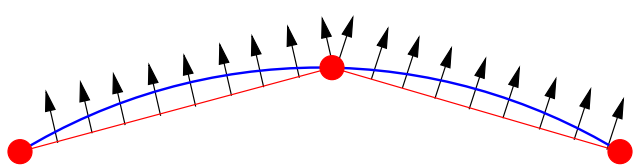


Figure 28: Normal Definitions on Faceted Geometry. When low order elements are used to describe a curved boundary, the normal is poorly defined at the edge of the facets.

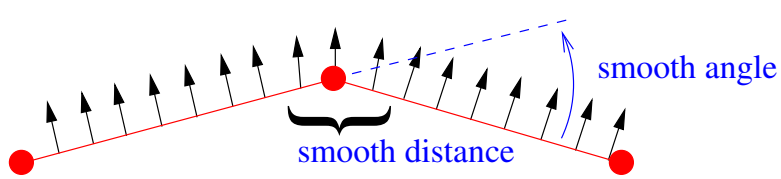


Figure 29: Smoothing Parameters for Surface Normals. No smoothing occurs for faces that are misaligned by more than the specified "smooth angle". Within the "smooth distance", normals vary linearly with relative distance from the node.

to matching faces that have opposing normals. For solids, this is seldom an issue. The normals for a solid are always outward from the solid, so two interacting solids (unless they occupy the same volume), will naturally have opposing normals. However, the situation for shell-shell or shell-solid interactions can be more complicated.

Sidesets may be created from the top or bottom surfaces of the shells. Thus, the shell surface has a natural normal direction determined by its connectivity, and the sidesets generated from the shells have a direction as well. The sideset direction may align *or oppose* the direction normal of the shell itself. If the shell normal does not oppose the normal of the mating surface, no interactions will be found, and the surfaces cannot be tied. See Figure 30.

2.22 RigidSet

Rigid Sets are intended as a usability tool to permit the analyst to treat a set of nodes as completely rigid. The input is straightforward.

```
RIGIDSET set1
     sideset 1
     sideset 2
     nodeset 88
END
```

The above definition would establish a single set that is tied together. For purposes of error reporting only, the name "set1" is associated with this example set. If multiple *independent* sets are required, then multiple rigidset definitions may be made.

The relevant parameters for **rigidsets** are shown in Table 57. Any number of `RigidSet` sections may be introduced, each will act independently. Exodus sideset or nodeset information may be included in the definition.

Table 57: RigidSet Parameters

Parameter	**type**	**description**
sideset	*integer*	sideset id
nodeset (*not recommended*)	*integer*	nodeset id
center node tied to node	*integer*	*see below*

2.22.0.1 Tied Node: The rigid set is often used as part of a tied joint (section 2.24). In this case, a "reference" node may be generated and tied to another block or element. This is accomplished with the keywords below.

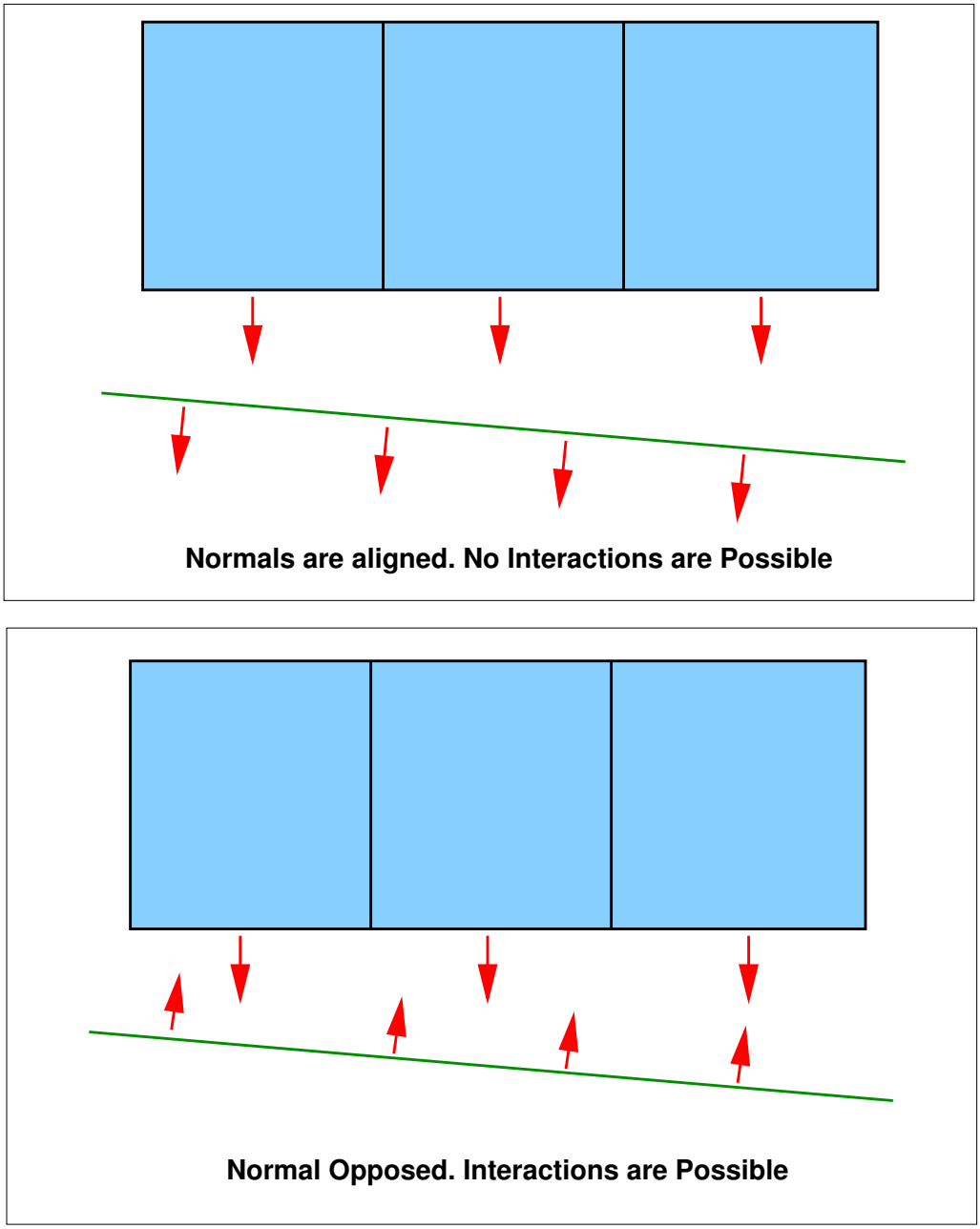


Figure 30: Shell Normal in Contact or Tied Interactions

```
centernode tied to node XX block YY
```

Here `XX` is the node number of the element in the block. This only works for blocks with a single element and has only been exercised for two node elements. Thus `XX` is either 1 or 2. There are examples in the "HowTo" documentation. Figure 31 illustrates the concept.

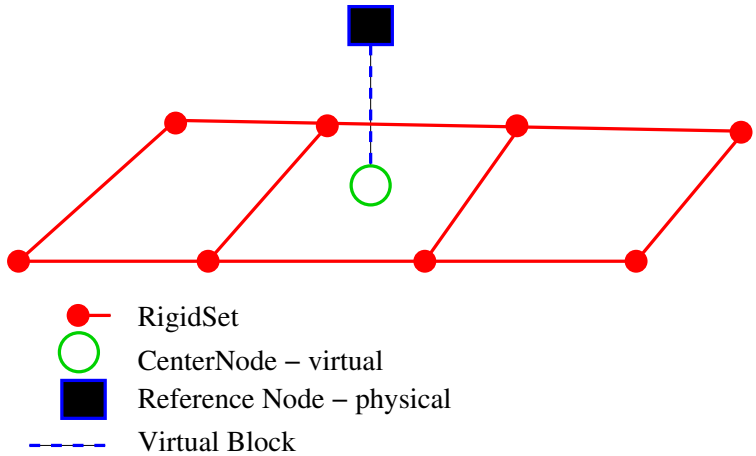


Figure 31: RigidSet/TiedJoint Centernode Connection. The model illustrates the connection of a physical rigid set to a physical reference node via a virtual center node and virtual connection block.

2.22.0.2 Limitations. Rigidsets meet an important need to tie many nodes together. Generally they are much more robust than generating collections of **RBAR**s or other rigid elements. However, it is very easy to generate redundant constraints through this input. Redundant constraints cause most linear solvers to fail, and we aren't good at providing diagnostics. Generally,

1. rigidsets must be completely disjoint, i.e. they may share no common nodes. If they share a node, they should be put in the same rigid set.
2. None of the nodes in the rigidset should be constrained (as through a boundary condition).
3. While nodesets can be used to define rigidsets, this is not recommended because parallel decompositions may put only one or two nodes on a processor. So few nodes may introduce local singularities in rotation that impact the linear solver. If possible, use a sideset to define the rigidset.
4. Other constraints (such as RBARS) should not further constrain the set.

This limitation does not prohibit the addition of an **RBAR** or other constraint which ties the rigidset to an *otherwise unconstrained* node.

2.23 RrodSet

Like the **RigidSet** of section 2.22, the **RrodSet** provides a convenient means of tying together a surface. All the limitations of the rigid set apply here. Unlike the rigid set, the rrodset constrains only the distance between nodes on the faces, and no rotational degrees of freedom are constrained. The **RrodSet** acts much like a Kevlar skin; it resists stretching, but does not impede bending.

For a quadrilateral face, the Rrodset is equivalent to applying a rigid rod to each of the edges of the face. A constraint is also placed across one of the diagonals of the face as shown in Figure 32. An example is shown below. Note that nodesets are not allowed in Rrodset specifications.

```
RRODSET
     sideset 5
END
```

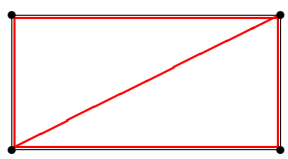


Figure 32: RrodSet Constraints. The black lines indicate the edge of the element. Red lines are corresponding linear constraints.

Like the **RigidSet**, the **RrodSet** may be used to connect a "reference" node to a block.

2.24 Tied-Joint

The "Tied Joint" structure is a meta structure that provides an efficient and robust means of modeling a joint structure. At the heart of the *Tied-Joint* is a whole joint model. For example, an *Iwan* element may represent the joint. However, the *Tied-Joint* permits flexible mixing of whole joint models and other models. For example, the *Iwan* element may be used to represent the shear response of the joint, while the normal components of response are represented by a tied surface. Thus, the energy loss of the joint would be properly approximated by the *Iwan* element, while the tied surface ensures that the normal surfaces remain properly aligned. It also avoids adding some of the artificial stiffness that a rigid set or collection of rbars introduces.

2.24.1 Input Specification

Refer to Figure 34 for reference to the model definition. An example input is shown in Figure 35. There are several sections to the model definitions.

Name: Optional name of this joint. Useful primarily in diagnosing error messages.

Normal Definition: In the *Tied-Joint* the joint behavior in the normal direction is governed by a "tied data" type definition. For many joints, the energy loss is primarily due to microslip in the shear directions. The tied conditions are described exactly as in the "Tied Data" (users section 2.20), but the data is entered within the tied joint specification. These specifications apply only in the normal direction. A pair of matching surfaces is a required part of this definition.

Shear Definition: A whole joint model may be referenced as part of the shear definition. Details of the shear specification depend on the normal definition. For a normal definition = "none", all 6 dofs must be specified in the referenced block. With a "slip" definition, the normal components have been specified. Only the 3 dofs associated with in plane motion are used. A fully tied normal requires no connector element.

Coordinate Frames: If the shear behavior is not isotropic, a reference to a coordinate frame may be required. The frame may be curvilinear (e.g. cylindrical), in which case whole joint quantities are evaluated at the centroid of the surfaces (see coordinates, 2.28). To reference the basic (or default) frame, use coordinate frame "0", which is the default. The coordinate frame is specified in the connected element frame.

For curvilinear coordinate frames, it may be difficult to exactly specify the orientation of the centroid of the surface. Any user defined coordinate frame will be projected to the plane of the surface at the centroid, and a new coordinate frame is generated for specification of the orthogonal, in-plane coordinates. The "X" and "Y" axes are projected to the plane, with the "Z" axis in the normal direction.

Shear Axis:

In the case when the whole joint model is a **Joint2G** element, the **shear_axis** can be used to specify the coordinate direction used for the first in plane constitutive component. We refer to Figure 33 for a description of the local coordinate system used to specify the constitutive behavior of the **Joint2G** element. The surface normal, n, is obtained from ACME as the normal on the node that is closest to the centroid of the sidesets that define the tied joint. This normal direction defines the "z" axis of the local coordinate system. The **shear_axis** definition specifies which of the 3 axis of the user-specified coordinate system (in this example coordinate 5) is intended to be the first shear direction for the constitutive response. Thus, if **shear_axis** is set to 1, then x' is defined as the part of the x axis (that is, the x axis as defined in the user-defined coordinate system) that is orthogonal to n. If the normal (or "z" direction) of the user specified coordinate system lines up exactly with the normal that is obtained from ACME, then the shear direction will be in exactly the same direction

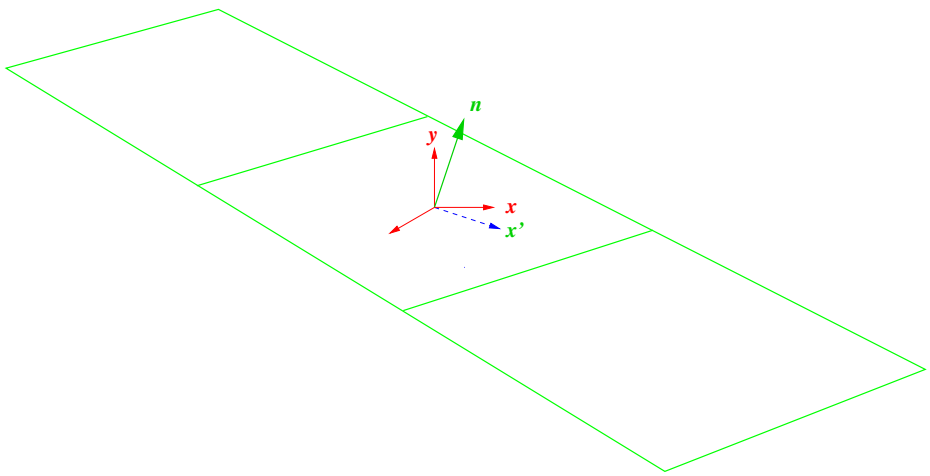


Figure 33: The surface normal, n, is defined by the normal of the surface at the centroid point. The shear axis direction (shown as x) is projected onto the surface at x'. The x' vector and the normal provide the basis for the generated coordinate frame.

as the x axis in the user-defined coordinate system. Generally, they will not line up perfectly, and this is the main reason why the **shear_axis** is needed. Of course, once n and x' are defined, the third component of the coordinate system $\hat{y}$ can be obtained by a cross product.

Parameters of the input are summarized in Table 58. Details are described below.

Keyword	**Description of option**
Name	*optional* name of this joint
Normal Definition	"slip" or "none"
- surface	matching surfaces
- *other Tied Data options as needed*	
Shear Definition	
- connect to block	reference block for whole joint
- shear distribution	"uniform"
	"distance_function"
	"rigid"
- side	"average"
	"rigid"
	"rrod"

Table 58: Tied Joint Parameters

Normal Definition: This keyword performs two functions with respect to the normal loading. First, it defines the type of connection. This may be "slip" or "none".

SLIP implies the faces will remain in contact, and shear effects are managed by the "shear definition".

NONE implies that no specific normal interaction is provided. Surfaces may separate or interfere. It may be used together with a "rigid" shear distribution.

The normal definition also includes a definition of the surface pairs in the joint.

Surface: Required specification for the joint surfaces. Other "tied data" parameters may follow.

connect to block: A reference to a block containing parameters for the whole joint.

shear distribution: The constraint weighting for the shear part of the surfaces. This keyword is currently not used, but will be implemented in a future update.

uniform provides equal weights to all nodes.

distance_function provides a weighting based on the return value of a function. For example,

```
distribution distance_function 5
```

refers to function definition "5" (section 2.29) to determine the weighting function. The function is passed the distance from the centroid of the surfaces to determine the weight at any given node.

rigid provides a means of constraining all the nodes on the surface in a rigid set (see users 2.22). This option can only be selected if the "normal definition" is "none". Presumably the analyst would provide a normal connection using a Joint2G or other element.

The "rigid" option removes all flexibility from the joint surfaces. The other options automatically generate a "virtual" node on each surface, and constrain the shear motion to that point much like a weighted "RBE3" type connection. The virtual node pair is managed in the shear directions by the shear stiffness contributions.

side: The type of side to make each surface.

average means no additional constraints are added to the surface. The two ends of the Joint2G element are attached to the surface using averaging type constraints.

rigid provides a means of constraining all the nodes on the surface in a rigid set (see users 2.22).

rrod provides a means of constraining all the nodes on the surface in a rrod set (see users 2.23).

Not all tied joint specifications are fully consistent. In particular, the specification of the "normal definition" and the "side" descriptions are not fully independent. Table 59 summarizes some of the dependencies between these two parameters.

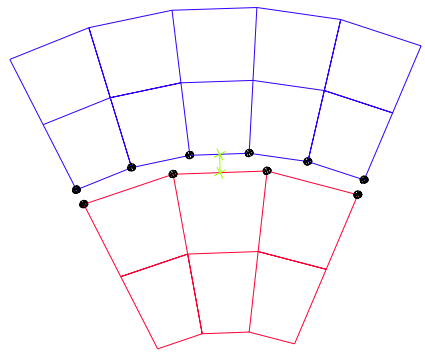


Figure 34: Tied Joint Geometry. The two side set surfaces are shown separated for clarity. A ghost element is created which connects only the shear components of the joint. Normal components are interconnected using Tied Data type structures.

Normal Definition	**Side**	**status**
none	average	The only constraints applied to the surface are from the RBE3.
	rrod	Adds only RRod type constraints to the surface.
	rigid	With a whole joint model, this recovers the legacy method of attaching whole joints.
slip	average	Does not add tangential constraints to the surface. Preferred method to tie to whole joint model.
	rrod	surface is allowed to flex, but is somewhat rigidized.
	rigid	Invalid. Overly Constrained.

Table 59: Tied Joint, "Normal" and "Side" dependencies

2.24.2 Output Specifications

Because the *Tied-Joint* is not fully represented in the **Exodus** database (except as a collection of surfaces), standard element output capabilities are insufficient to represent the data. The data is divided into two categories: configuration and results.

2.24.2.1 Configuration Output: The configuration output is only available in the text output of **Sierra/SD** , i.e. in the .rslt file. It is requested with the keyword "INPUT" in the "ECHO" section (see 2.8). This includes the following.

1. The type of the normal enforcement.
2. Surface information.
3. Centroid of the surface pairs (if applicable).

```
TIED Joint
    Normal Definition = slip
       surface 3,5
       search tolerance = 1e-6
    Shear distribution = uniform
    connect to Block 11  // Joint2G block
END

// definitions for the referenced joint2G block
Block 11
    Joint2G
    Kx=Iwan 1
    Ky=Elastic 1e6
    Coordinate=5         // for anisotropic shear parameters
END
```

Figure 35: Tied Joint Example

4. Owning processor for the shear elements (if applicable).
5. Shear models.

Results Output: The only results output that is currently available for a Tied Joint consists of the forces in the **Joint2G** element that connect the two surfaces of the tied joint together. Currently, these forces can be only obtained in the history (or frequency) file for a transient or nonlinear transient analysis. They cannot be written to the global exodus output file. If we consider the same example that is given in Figure 35, we could obtain the element forces as follows for a transient analysis

```
HISTORY
  block 11
  eforce
END
```

or, for a frequency domain analysis,

```
FREQUENCY
  block 11
  eforce
END
```

where in this example block 11 is the **Joint2G** block that connect the two surfaces of the tied joint together.

2.25 BLOCK

Each element block in the **Exodus** file, must have a corresponding **BLOCK** entry in the input file. The opposite is not true - there can be **BLOCK** entries in the input deck that do not have corresponding entries in the **Exodus** file. There are two cases where this can happen

- Virtual blocks. These are blocks that have entries in the input deck and are intended to be part of the model, but have no corresponding entries in the **Exodus** file. At this time, only Joint2G elements (see section 3.31) can be defined to be virtual blocks.
- Extra blocks that have entries in the input deck but are not intended to be part of the model. These blocks are silently ignored by **Sierra/SD** .

It is an error to have multiple definitions for the same block. However, **Sierra/SD** does not report the error. Which definition is used is not defined. This section contains information about the properties of the elements within the block.

2.25.1 Block Parameters

There are two main types of block parameters:

1. Parameters exist which are common to most elements. These include:
   - Material property references are required for most elements. The material reference is of the form, `material=material_id`, where `material_id` is a string representing the material identifier (see section 2.27).
   - coordinate frames - *optional*
   - nonlinear behavior - *optional*
   - block damping - *optional*
   - non-structural mass - *optional*
2. Element specific names and parameters. These properties depend on the element type. Clearly shells will require a thickness, while it is meaningless for solids.

An example is provided in Figure 36.

A list of the applicable attributes for some of the different element types is shown in Table 62. Each element type is outlined in section 3.

```
BLOCK 32
    material 2
    tria3
    thickness 0.01
END

BLOCK 34
        material aluminum
END

BLOCK 3
    Coordinate 1
    Spring
    Kx=1e6
    Ky=0
    Kz=0
    BLKBETA=0.0031
END

Material aluminum
    ....
END
```

Figure 36: Example Block input. Note that the material ID specified for BLOCK 32 uses an index (material 2), whereas BLOCK 34 uses a specified material ID string "aluminum" These refer to materials defined by blocks "MATERIAL 32 ... END" and "MATERIAL aluminum ... END" respectively (see Sec. 2.27 for details).

Table 60: General Block Parameters

Keyword	Values	Description
nonlinear	yes/no	blockwise nonlinear behavior
material	*string*	material identifier
rotational_type	eulerian/lagrangian/none	blockwise behavior for rotational dynamics terms
coordinate	*integer*	reference coordinate frame
blkalpha	*Real*	blockwise mass proportional damping
blkbeta	*Real*	blockwise stiffness proportional damping
nsm	*Real*	blockwise non-structural mass

2.25.2 General Block Parameters

Parameters that are generally applicable to almost all blocks are listed in Table 60. More detailed descriptions are available in the following paragraphs.

2.25.2.1 Nonlinear Behavior. The nonlinear behavior of the block in nonlinear solutions is controlled by the **nonlinear** keyword. The global default for block-level nonlinear behavior is set in the PARAMETERS section (2.3). Within each block, we can override that default value. For example, to set a block to default to linear behavior, we would have the following BLOCK definition.

```
BLOCK 3
     nonlinear=no
     material 2
     tria3
     thickness 0.01
END
```

Similarly, to turn on the nonlinear behavior for the block, we would have,

```
BLOCK 3
     nonlinear=yes
     material 2
     tria3
     thickness 0.01
END
```

Note that these block-level nonlinear flags override the global **nonlinear_default** keyword that is set in the PARAMETERS section.

LIMITATIONS:
Linear element behavior in a nonlinear solution is limited to the linear range of the element. For example, rotations are stored incrementally in nonlinear solutions. This permits us to use geometrically nonlinear element formulations (such a corotational formulation). However, it limits the *linear* behavior in such solutions to rotations less than 360°.

2.25.2.2 Rotational Loading Matrices. For problems involving rotational loads, the **rotational_type** keyword allows the analyst to specify which type of rotational formulation to use for a given block of elements. The Eulerian formulation involves a fixed (non-rotating) coordinate system. The Lagrangian formulation attaches a rotating coordinate system to the block. If the None options is chosen, then rotational loads are ignored for this block. Thus, a structure with a rotating disk would only have the rotational terms applied to the spinning disk, and not the entire structure. The default is for the **rotational_type** keyword is None.

2.25.2.3 Coordinate Frame Reference. The reference coordinate system may be defined in a block. This definition applies to all the elements of the block and the associated materials. At this point, the coordinate system is only recognized for a subset of the elements (solid elements and springs). Further information on coordinate systems may be found in section 2.28.

2.25.2.4 Block Specific Damping. In section 2.37, various methods of specifying the damping parameters for a model are identified. In addition to these methods, block specific damping parameters may be applied. These apply a stiffness (or mass) proportional damping matrix on an element by element basis within the block. Thus, if a model is made of steel and foam, one could apply a 5% stiffness proportional damping term to the foam, but leave the steel undamped.

There is no physical justification for proportional damping, and there is no expectation that it will accurately represent damping mechanisms in a structure. However, it is easy to apply, and there are cases where proportional damping may reveal a need for more accurate damping models. As with all damping models, the effects depend on the solution type. For example, both statics and eigen analysis ignore the damping matrix.

The damping matrix generated from block specific damping is defined as follows.

$$D = \sum_{i}^{nblks} \alpha_i M_i + \beta_i K_i \tag{42}$$

Where D is the real system damping matrix, and α_i and β_1 are the proportional mass and damping coefficients for block i. These coefficients are completely analogous to the system

Table 61: Non-Structural Mass Units

Element Type	Units	Example
One Dimensional	mass/length	lbs / in
Two Dimensional	mass/area	lbs / sq-in
Three Dimensional	mass/volume	lbs / cu-in

level coefficients described in section 2.37. The damping contributions from these block parameters are always added to the other contributions.

Block specific damping is applied using the **blkalpha** and **blkbeta** parameters. Block proportional damping generates a damping matrix that is would couple modal based solutions. It is not currently available in modal solutions such as **modaltrans**. Also see section 2.27.10 for material modal like damping.

2.25.2.5 Non-Structural Mass. An element block may define a non-structural mass (nsm) to be applied in addition to the elements' internal mass. This can be used to simulate an external load being placed on the elements. It is specified as a pseudo density, and the units depend on the type of element being used. Table 61 list these units.

The following is an example of how to use non-structural mass in the input file:

```
//nsm specified in pounds per square inch
BLOCK 3
     material 2
     tria3
     thickness 0.01
     nsm 0.005
END

MATERIAL 2
     density 0.5
END
```

2.26 Macroblock

It is possible to overload a single element block in the **Exodus** file to be used simultaneously as several different element types. To use this feature, the **BLOCK** entry should list the ids of the new macroblocks, which will share the same geometry from the **Exodus**

Table 62: Element Attributes

Element Type	attr	keyword	Description
ConMass	1	Mass	concentrated mass
	2	Ixx	xx moment of inertia
	3	Iyy	yy moment of inertia
	4	Izz	zz moment of inertia
	5	Ixy	xy moment of inertia
	6	Ixz	xz moment of inertia
	7	Iyz	yz moment of inertia
	8,9,10	offset	offset from node to CG
Beam	1	Area	Area of beam
	2,3,4	Orientation	orientation vector. For the orthogonal direction
	5	I1	First bending moment
	6	I2	Second bending moment
	7	J	Torsion moment
	9,10,11	offset	beam offset
Spring	1	Kx	spring constant in X
	2	Ky	spring constant in Y
	3	Kz	spring constant in Z
Triangle	1	thickness	thickness
	2	offset	shell offset in normal direction
Quad	1	thickness	thickness
	2	offset	shell offset in normal direction

file. Additional parameters should not be included in the **BLOCK** specification as the original element block will be treated as a "dead" element. For every macroblock listed, a **Macroblock** entry must be present in the input file. A **Macroblock** entry should look exactly like a normal **BLOCK** entry except for the keyword. The macroblock ids must be unique and different from any existing block ids.

2.26.0.1 Macroblock Example.

An example is provided below.

```
// the following element block is associated with block 1 in the
// exodus file.  It specifies which macroblocks use this block.
BLOCK 1
    macroblock 11 12
END

// the following macroblocks specify the types to use for the block
MACROBLOCK 11
    dashpot
    k=1e6
    c=1e4
    cid=1
END

MACROBLOCK 12
    spring
    Kx 1e+3
    Ky 1e+3
    Kz 3e-1
END
```

Macroblocks 11 and 12 will be used as though there are two distinct element blocks in the **Exodus** file, one treated as a dashpot and the other as a spring. Because the macroblocks do not actually exist in the **Exodus** file, element variables cannot be associated with them. However, it is still possible to obtain some element variable results from the `.rslt` file (section 2.8). Macroblock results will be specially labeled in this file because their elements' ids are not unique.

2.27 MATERIAL

Most element blocks must specify a material. Details of that material are included in the material section. The material section contains a material identifier (which is usually an

integer, but may be any string), an optional **name** keyword followed by a material name a material type keyword and the necessary parameters. The different material types and their parameters are summarized in Table 64.[1]

For example,

```
MATERIAL steel
    isotropic
    name "my model of steel"
    E 30e6
    nu .3
END
```

Deterministic materials may be input as **isotropic**, **orthotropic**, **orthotropic_prop**, **anisotropic**, or **isotropic_viscoelastic**. In addition, stochastic isotropic materials may be specified as **S_isotropic**.

2.27.1 Isotropic Material

Isotropic materials require specification of two of the following parameters.

Parameter	Description
E	Young's Modulus
nu	Poisson's Ratio
G	Shear Modulus
K	Bulk Modulus

Isotropic materials are the default, and the keyword **isotropic** is not required. Of the four parameters, *exactly two* must be supplied. They are related by the following equations.

$$
\begin{aligned}
E &= 3K(1-2\nu) \\
G &= \frac{3KE}{9K-E}
\end{aligned}
$$

Internally, **Sierra/SD** stores the values of **K** and **G**.

[1] The material must be uniquely identified by it's identifier. The "name" is used in reports and cannot be used as an identifier. Using a descriptive string, rather than a material number, can enhance readability to an input.

2.27.2 Anisotropic Material

Anisotropic materials require specification of a 21 element C_{ij} matrix corresponding to the upper triangle of the 6x6 stiffness matrix. Data is input in the order C_{11},C_{12}, C_{13}, C_{14}, C_{15}, C_{16}, C_{22}, etc. The C_{ij} must be preceded by the keyword **Cij**. The keyword **anisotropic** is also required. Materials are specified in the order xx, yy, zz, zy, zx, xy. Note that this ordering varies in the literature. It differs from the ordering in Nastran and Abaqus, but is consistent with much of the published materials science data. An example input file with an anisotropic material is found in section 1.2.

Each element defines a coordinate frame. The 1-direction for the material parameters is defined by the 1-direction of the coordinate frame. The default is the basic or Cartesian frame. In spherical coordinates (r, θ, ϕ), for example, the radial direction is the 1 direction.

For anisotropic (and orthotropic) materials, the check to make sure material properties are acceptable is skipped. A message is printed notifying the user that this check is skipped.

2.27.3 Orthotropic Material

Orthotropic material entry is similar to to the anisotropic case.

A difference is that the keyword **orthotropic** replaces **anisotropic**, and only 9 C_{ij} entries are specified. These entries correspond to C_{11}, C_{12}, C_{13}, C_{22}, C_{23}, C_{33}, C_{44}, C_{55} and C_{66}. Like the anisotropic material definition, the order is xx, yy, zz, zy, zx, xy.

Alternatively, an orthotropic material may be specified using **orthotropic_prop** and the material parameters `E1, E2, E3, nu23, nu13, nu12, G23, G13`, and `G12` as shown in the following example. Note that all elastic materials must satisfy requirements that the elasticity matrix is positive definite.

```
Material honeycomb
   orthotropic_prop
   name 'aluminum honeycomb in Mpa'
   E1 = 508.7
   E2 = 7641.0
   E3 = 14750.0
   Nu12 = .2
   Nu23 = .0825
   Nu13 = .1
   G12 = 115
   G23 = 2320.
   G13 = 450.
   density=0.5
END
```

A single orthotropic layer may be specified using **orthotropic_layer**. An orthotropic layer must specify 4 of the above parameters (`E1, E2, nu12, G12`). Here is an example:

```
Material 13
   orthotropic_layer
   name 'ortho layer 1'
   E1 = 508.7
   E2 = 7641.0
   Nu12 = 1.293
   G12 = 115
   density=0.5
END
```

If sensitivity analysis is being performed (see section 2.35), one indicates the parameters for analysis by following these parameters with the `+/-` characters. In the first entry method, a sensitivity analysis must be performed on all 9 parameters. In the second, each individual parameter must be requested individually. The concept is that the sensitivity is performed with respect to the labeled parameters, i.e. either the set of C_{ij} parameters, or each individually labeled `E1` term.

2.27.4 Stochastic Material

For stochastic materials, all material properties are determined by a table look-up, based on the element ID. The file name for the table look up is taken from the **name** identifier. The file is a standard text file with the first column corresponding to the element ID. The second column is the bulk modulus, K, and the third (and final) column is the shear modulus, G. The element IDs in the file need not be continuous, but they must be sorted in increasing order. Thus the **S_isotropic** data look up file contains the element ID, the bulk modulus and the shear modulus, with one line for each element. The stochastic material model is very preliminary and is expected to change significantly in the next few years. An example section from the input file is presented below.

```
MATERIAL 3
        s_isotropic
        name "mat3.txt"
        density 0.288
END
```

From within "`mat3.txt`" the data looks like the following. The last two columns are bulk and shear moduli respectively.[2]

[2] For relations between isotropic moduli, see the discussion in section 2.27.1.

```
1   40e6  20e6
2   40e6  20e6
4   40e6  20e6
9   40e6  20e6
10  40e6  20e6
11  40e6  20e6
```

2.27.5 Linear Viscoelastic Material

Linear visco elastic materials require the specification of the density, and the limiting moduli E_g, E_inf, G_g, G_inf. The subscript 'g' refers to the glassy modulus, which occurs at $t = 0$, or $\omega = \infty$. The subscript 'inf' refers to the rubbery modulus, which occurs at $t = \infty$, or $\omega = 0$. In addition the Prony series for the visco elastic materials have to be specified using keywords K_coeff, K_relax, G_coeff, and G_relax. All of these parameters are required.

Although in most cases the values of E_g, E_inf, G_g, G_inf are considered to be constants, there are cases where they actually depend on temperature. Temperature functions can specify the value for the limiting moduli, for a given value of T_current. For example, if the limiting moduli typically depend linearly on temperature, a linear function can be specified for the values of E_g, E_inf, G_g, G_inf. We refer to the example given below for the specifics on how to set this up.

For the bulk modulus K, the Prony series parameters are defined by the following equation:

$$K(t) = K_{inf} + (K_g - K_{inf}) \sum_i K_{coeff}[i] * e^{-\frac{t}{K_{relax}[i]}} \tag{43}$$

A similar equation holds for the shear modulus. Note that, the K_coeff and G_coeff MUST sum to 1.0 (individually). Otherwise, the formulation is inconsistent. That is,

$$\sum_i K_{coeff}[i] = \sum_i G_{coeff}[i] = 1.0 \tag{44}$$

Note that the number of terms in K_coeff and K_relax must be the same, and the number of terms in the G_coeff and G_relax must be the same. However, the number of terms in the K series does not have to equal the number of terms in the G series. Thus, one could simulate a case where the material shear modulus G is visco elastic, but the bulk modulus is not. In this case, the latter would have no terms in its series.

Optional parameters for visco elastic materials include reference (T_0) and current temperature (T_current), and the WLF constants C_1 and C_2. (more explanation of the Williams-Landel-Ferry (WLF) equation is given below). We note that any units of temperature can be used, as long as they are consistent with the values of the constants (e.g. C_1 and C_2). For Kelvin and Celsius units, the constants C_1, C_2, a_T1 and a_T2 are the same, since the expressions below involve differences of temperatures. Also, two constants

may be specified that describe the curve fit for the shift function, a_T1 and a_T2, in the case when T_current - T_0 is negative. The equation, provided by Terry Hinnerichs, is a good characterization of many visco elastic materials. Its form is

$$\log_{10}(a_T) = a_T1 * (1 - e^{a_T2*(T_current - T_0)}) \tag{45}$$

If these optional parameters are not specified, default values are used, as shown in the table below. Note that equation 45 will only be used to compute the shift functions if the

Table 63: Default Parameters for Viscoelastic Materials

parameter	default value
T_0	0.0
T_current	0.0
C_1	15.0
C_2	35.0
aT_1	6.0
aT_2	.0614

parameters aT_1 and aT_2 are specified. Otherwise, the standard WLF equation is used, as described below.

If the parameters aT_1 and aT_2 are not specified, then the shift factors are computed using the WLF equation. This equation is frequently used to determine an approximate set of shift factors when experimental data for a particular material is not at hand. The shift factors computed from this equation are used to scale the coefficients in the Prony series. The shift factors computed from the WLF equation are a strong function of temperature. The WLF equation is as follows

$$log(a_T) = -\frac{C_1(T_current - T_0)}{C_2 + T_current - T_0} \tag{46}$$

where T_current is the current temperature in the block, and T_0, C_1, and C_2 are material parameters that are determined experimentally. If C_1 and C_2 are not known for a particular material, then the default values given above are typically used. Typically, T_0 is the glass transition temperature of the material of interest. More explanation of the WLF equation can be found in the books by Aklonis,[27] and Ferry.[28]

After computing the shift factors using one of the two approaches given above, the relaxation times are shifted. This occurs before computations begin, using the relations

$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{47}$$
$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{48}$$

These shifts are automatically computed given T_0, T_current, C_1, and C_2, so that the user does not need to shift the relaxation times beforehand. Note that if these parameters

are not specified in the input file, then they are given default values that result in no shifting of relaxation times. In such a case, $a_T = 1$.

An example material block for a linear visco elastic material looks like:

```
MATERIAL 9
 isotropic_viscoelastic
 name "foam"
 T_0=0
 T_current=25
 C_1=1
 C_2=2
 aT_1=6.0
 aT_2=.06
 K_g = function 1
 K_inf 10.0e6
 G_g 10.0e1
 G_inf 12.0
 K_coeff .5 .5
 K_relax 3.0 2
 G_coeff .5 .5
 G_relax 1 3
 density 0.288
END
```

Note that the coefficients of both K and G sum to 1.0. This is necessary for a consistent formulation. Also, in this case we specify a temperature function for K_g. Thus, the value of K_g used in the simulations is the value of function 1, at the particular temperature T_current.

A note on visco elastic materials: when using visco elastic materials in a nonlinear transient simulation, it is necessary to specify "nonlinear=no" in the BLOCK section of the visco elastic block. This is because different internal force mechanisms are called for linear and nonlinear cases, and visco elastic materials in **Sierra/SD** only support linear constitutive model and small deformation.

We also note that if visco elastic materials are used in a statics simulation, then the material is assigned the properties G_{inf} and K_{inf}. This is because in a slow (static) loading, the material would respond with these material properties since they are the long-time or slow response properties.

2.27.6 Acoustic Material

Linear acoustic materials require the specification of the fluid density, and the linear speed of sound. In addition, the keyword **acoustic** must be in the material block.

```
MATERIAL
  name "air"
  acoustic
  density 1.293
  c0 332.0
END
```

Nonlinear acoustic materials require one additional parameter, B_over_A, which is a measure of fluid nonlinearity. For air, $B_over_A = 0.4$. Tables of B_over_A for various fluids can be found in.[29]

2.27.7 Temperature-Dependent Material Properties

Material properties in **Sierra/SD** can be specified to be temperature dependent. Temperature dependent material properties are supported when temperatures are read in from an **Exodus** file, or when they are specified on a block-by-block basis. In the case of exodus temperatures, the material properties would vary from element to element, since the temperatures vary with each element. When temperatures are specified on a block-by-block basis, the temperature-dependence of the material properties can be specified explicitly in the input deck. We note that when temperatures are specified both in the **Exodus** file as well as on a block-by-block basis in the input deck, the input deck values take precedence.

For linear elastic materials, an example of specifying temperature dependent properties is given below.

```
MATERIAL 1
        E function=1
        alphat .001
        tref 100
        nu 0.0
        density 7700.0
END

MATERIAL 2
        E function=2
        alphat .001
        tref 100
```

```
        nu 0.0
        density 7700.0
END

FUNCTION 1
       type LINEAR
       data 0.0 4.0
       data 5.0e9  4.0
END

FUNCTION 2
       type LINEAR
       data 0.0 3.0
       data 5.0e9  3.0
END
```

In this case, the elastic modulus of material 1 is specified by function 1, and the elastic modulus of material 2 is specified by function 2. The moduli of each element will be determined from its temperature and an interpolation on the function. In this example, the functions are trivial, and thus the moduli of materials 1 and 2 will be 4 and 3, respectively. Note that the moduli, density and any of the 4 elastic constants k, g, e, ν can be specified as temperature dependent, and can be given different functions. In this example, the Poisson ratio is constant and only the elastic modulus is temperature dependent.

For visco elastic materials, functions do not need to be specified in the **material** block to designate temperature dependence of the shift factors. This is accounted for automatically. See section 2.27.5 on visco elastic materials for more details.

Currently, only linear elastic and linear visco elastic materials can be given temperature-dependent material properties.

2.27.8 Density

For solutions requiring a mass matrix, all material specifications require a keyword **density** followed by a scalar *value*.

2.27.9 Specific Heat

Conversion of energy deposited in a structure to a change in temperature may be effected by a specific heat.

$$Q = \rho V \, C \, \Delta T \tag{49}$$

Table 64: Material Stiffness Parameters

material type	parameters
isotropic	any two of K, G, E or ν
orthotropic	nine C_{ij} entries
orthotropic_prop	$E1$, $E2$, $E3$, $nu23$, $nu13$, $nu12$, $G23$, $G13$, $G12$
anisotropic	21 C_{ij} entries
S_isotropic	file containing K and G

where Q is the total heat energy, ρ is the density, V is the volume, C is the specific heat and ΔT is the change in temperature. It is up to the analyst to ensure that consistent units are employed. Note also that the analyst must determine under what conditions the specific heat is applied (constant pressure or constant volume).

Specific heat is used only in applying boundary conditions. Energy deposited within a structure is converted to temperature using equation 49. Once converted to temperatures, thermal stresses and temperature dependent material properties may be applied. The specific heat defaults to 1.0, and must not be zero.

```
MATERIAL 'Steel-SI'
    E=200e9                 // Pa
    NU=0.28
    density=7850            // kg/m^3
    specific heat = 0.45    // J/(gK)
    tref = 300 // K
    alphat = 0.001
END
```

The reference temperature is used only for temperature dependent material properties, such as in visco elastic materials. In other words,

$$\Delta T = \frac{Q}{\rho V C} \tag{50}$$

$$T_{current} = T_{ref} + \Delta T \tag{51}$$

$$\epsilon_{thermal} = \alpha_T (T_{current} - T_{ref}). \tag{52}$$

Recall that the specific energy is used for energy loads (sec. 2.15.8). The specific energy is the energy per unit *mass*, $\tilde{E} = Q/(\rho V)$.

2.27.10 CJetaFunction

For the **CJdamp** solution method (see section 2.1.4), a frequency dependent damping coefficient, $\eta(f)$, may be specified[3]. All other solution methods will ignore this keyword. The **CJetaFunction** keyword requires as a parameter the identifier of a function. Its use is specified in the following example. See section 2.29 for details in specifying the function. If no function is specified, the block will be treated as if the function were identically zero everywhere.

```
MATERIAL 1
  E=10.0E6
  NU=0.28
  density=0.098
  cjetafunction=1
END

function 1
  name 'function to use for material 1 eta'
  type linear
  data 0.0 0.001
  data 100 0.010
  data 200 0.030
  data 400 0
end
```

The function specifies the frequency, amplitude pairs for η. The frequencies are in the same units as the modal frequencies (i.e. there is no factor of 2π, and they are usually supplied in Hertz). The **CJdamp** solution process interpolates the function at the eigen frequencies to determine the effective damping for any particular mode.

2.28 COORDINATE

Coordinate systems may be defined for reference to the materials and boundary conditions. As reported in the "history" section, nodal results may also be reported in arbitrary coordinate frames in the history file only (see section 2.10). Note that all nodal locations, outputs, etc. are always defined in the basic coordinate system in the standard **Exodus** files. These new coordinate systems are always defined based on three *locations*, which are defined in the basic coordinate system. These locations are illustrated in Figure 37.

1. The location the origin of the new coordinate system, P_1.

[3] η is twice the normal modal damping coefficient. Thus, if eta=0.02 for all materials, the equivalent modal damping will be 1 percent.

2. A point, P_2, on the Z axis of the new system. Note that the location is required, not the direction vector $\tilde{Z}$.

3. A point, P_3, in the $\tilde{X}\tilde{Z}$ plane of the new system. The vector from P_1 to P_2 need not be orthogonal to $\tilde{Z}$, but it may not be parallel to it.

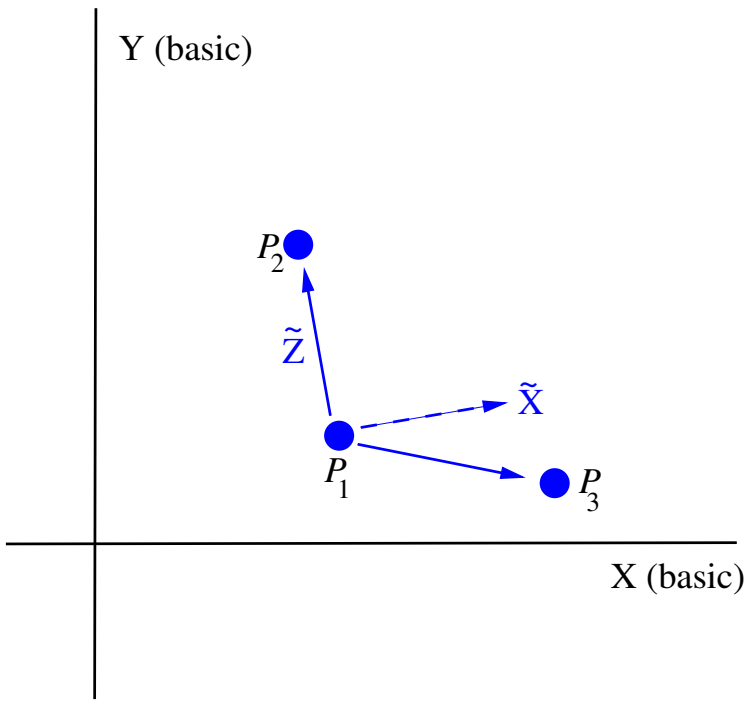


Figure 37: Coordinate System Definition Vectors. The origin of the new frame is at P_1. The new $\tilde{Z}$ axis is the vector from P_1 to P_2. Spherical frames align their polar axis with $\tilde{Z}$. The third point, P_3, defines the $\tilde{X}$ direction in a Cartesian frame, the $\theta = 0$ direction of a cylindrical frame, or the $\phi = 0$ direction of a spherical frame.

Coordinate systems for cartesian, cylindrical and spherical coordinates may be defined. In the case of noncartesian systems, the XZ plane is used for defining the origin of the θ direction only.

This example creates a cylindrical system located at a point (1,1,1) with the cylindrical axis in the (0,0,1) direction and the radial coordinate in the global Y direction.

```
Coordinate 7
    cylindrical
    1 1 1
    1 1 2
    1 2 1
END
```

The keywords for the coordinate system definitions are:

1. RECTANGULAR or CARTESIAN to define a cartesian system,
2. CYLINDRICAL for a cylindrical, i.e. polar system, and
3. SPHERICAL for a spherical system.

In spherical coordinates, it may help to think about the Cartesian frame $(\tilde{X}, \tilde{Y}, \tilde{Z})$ with the same orientation as (r, θ, ϕ):

$$\begin{aligned} \tilde{X} &= r\sin(\theta)\cos(\phi) \\ \tilde{Y} &= r\sin(\theta)\sin(\phi) \\ \tilde{Z} &= r\cos(\theta), \end{aligned}$$

$$0 < \theta < \pi, \quad 0 < \phi < 2\pi.$$

In the spherical coordinate frame the $\tilde{Z}$ direction is the North pole, $\theta = 0$. And the $\tilde{X}\tilde{Z}$ plane is $\theta \neq 0$ and $\phi = 0$.

If *input* is selected in the ECHO section then the transformation matrix will be output in the `.rslt` file (section 2.8). The transformation matrix is a unitary matrix which can be used to transform vectors from one system to another. If we let T be the matrix reported in the `.rslt` file, then the transformation from the basic system to the rotated frame is given by,

$$v_{new} = T^T v_{basic}$$

where	v_{new}	is the vector in the new coordinates,
	v_{basic}	is the vector in the basic system, and
	T^T	is the transpose of the `.rslt` matrix reported.

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node (see section 2.10). The history file will display variables as cartesian regardless of coordinate choice. Table 65 shows the corresponding values for cylindrical and spherical coordinates.

2.29 FUNCTION

Time, frequency and/or spatially dependent functions for transient and frequency response analysis can be defined using the **function** section. The following are simple examples of the use of a **function**.

Coordinate System	History Variable	Corresponding Coordinate
Cylindrical	X	r
	Y	θ
	Z	z
Spherical	X	r
	Y	θ
	Z	ϕ

Table 65: Coordinate Names for history files

```
FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 2
// This is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945.
// The equation is y(t)=-800*t^2 + 8.9943*sqrt(t)

  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END
```

The keywords for these function definitions are:

1. TYPE to define the functional form,
2. NAME for reference in echo and output, and
3. DATA for the functional parameters.

Other function definitions may require more parameters.

2.29.1 Linear Functions

For linear functions, the data elements are points of the function where the user defines the value of the independent variable (e.g. time) and the corresponding value of the function. Linear interpolation is used to find all other values of the function. In order to make the linear interpolation unique, the order of the input data is important. Input checks will ensure that time on subsequent data points is always greater than or equal to time on the previous data point so that curves cannot double back on themselves. For example,

```
FUNCTION 3
    name "illegal_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.05 1.
    data 0.04 0.   //illegal. the first column must never decrease
END
```

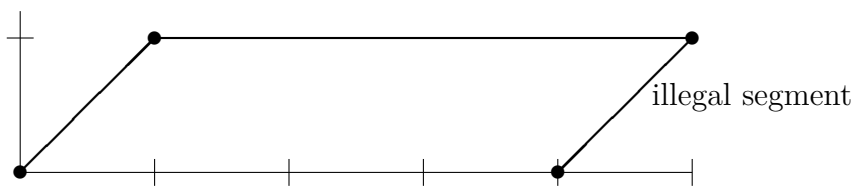


Figure 38: Linear function #3. "illegal_fun"

Linear functions will extrapolate by using the value of the nearest data point. For example, in the following function, f(t=0.3) = 0.5.

```
FUNCTION 5
    name "extrap_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.02 0.5
END
```

Note that while it is possible to have functions that have two values for the same time, this is not recommended. Such functions are very susceptible to round off. Solutions may vary depending on the platform or compiler used.

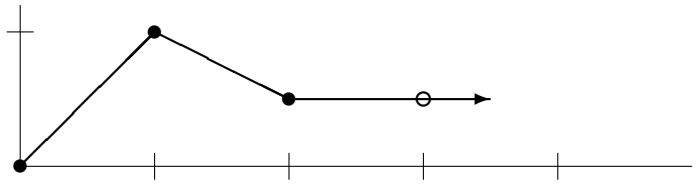


Figure 39: Linear function #5. "extrap_fun"

2.29.2 Functions using Tables

Functions may be specified by reference to a linearly interpolated **table** (as discussed in section 2.31). The table must be of dimension=1. Tables are very similar to the linear functions described above with several important differences.

1. Referencing a value of a table beyond the valid range is an error. This is seldom a problem in frequency domain analysis, but could often be an issue for time domain analysis.
2. Tables can be more memory efficient that linear functions in some cases where there is a large amount of data. This is especially important if only a few processors need access to that data.
3. Tables are almost always much faster than linear functions, especially as the data size grows.

The function in the following example is a tabular representation of the data of Figure 39 and Function 5 above.

```
FUNCTION 7
  type table
  tablename=example7
END

TABLE example7
  dimension=1
  size=5
  datafile='example7.txt'
  origin 0.0
  delta  .01
END
```

Within the datafile, “example7.txt”, the following data would be represented.

```
0.0
1.0
0.5
0.5
0.5
```

Of course, the linear function can be evaluated for any time, and the table is limited to the range 0-0.04. Table type functions require the **tablename** keyword.

2.29.3 Polynomial Functions

For polynomials, the data points given are the exponent of the independent variable and a scale factor for that term. The independent variable taken to any real power will always be evaluated as positive. If powers are repeated, their coefficients will sum. For example,

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.
    data 2.0 0.1
    data 1.0 0.5
END
```

is equivalent to

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.5
    data 2.0 0.1
END
```

The function value as a function of the independent variable t is,

$$f(t) = 1.5t + 0.1t^2.$$

2.29.4 LogLog Functions

In frequency domain analysis, log/log functions are commonly used for application of loads. This is particularly true for random vibration inputs which are commonly specified on log/log plots. The **loglog** option allows linear interpolation on a log/log plot so that only the corner frequencies need be specified. An example follows.

```
FUNCTION 1
   name "my_loglog"
   type loglog
   data 1.0 1e-8
   data 299 1e-8
   data 300 0.01
   data 2000 0.03
   data 8000 0.03
   data 10000 0.01
   data 10001 1e-8
END
```

2.29.5 Random Functions

There are two different types of internal random function distribution: a uniform and a Gaussian distribution. For both distribution types, the values are randomly generated according to the range that is input. For details of the **RandomLib** function, which has a more general capability see section 2.29.7.

For uniform distributions, the left range number is the lower bound and the right number is the upper bound, both inclusive. For a Gaussian distribution, the left number is the mean (or center of the distribution), and the right number is the standard deviation.

The *seed* determines the seed for a new sequence of pseudo-random numbers, either auto or a positive integer. The auto option seeds the generator using the computer clock, which will nearly always give an irreproducible string of random numbers. However for reproducible results, a manual seed may be given. The sequence of numbers is random, but the same random sequence of numbers generated from a specific seed is always the same. Please note that the number 0 acts the same as if you had entered auto as the seed.

Random functions use the pseudo-random number generator in the rand() function of the C library.

```
FUNCTION 2
      name "some_function"
```

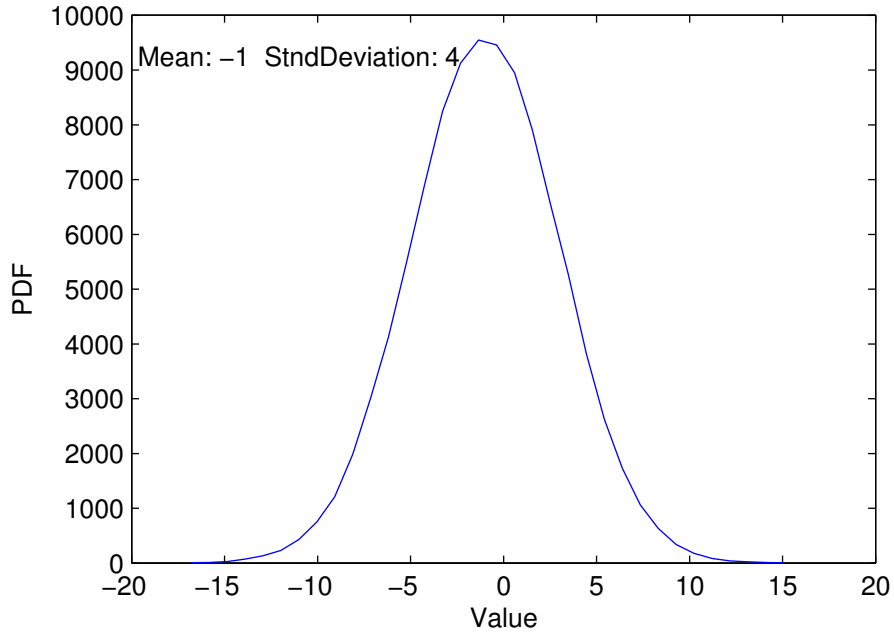


Figure 40: Example *Gaussian* output.

```
        type random
        distribution Gaussian
        range -1.0 4.0
        seed auto
END
```

That example would produce the distribution shown in figure 40:

Parameters are shown in Table 66:

Table 66: Random function parameters

Parameter	Type	Values
distribution	string	*uniform* or *Gaussian*
range	two Real numbers	lower and upper bound of distribution (uniform) OR mean and standard deviation (Gaussian)
seed	string/integer	*auto* OR any integer

2.29.6 SamplingRandom Functions

The random pressure loading defined in section 4.4 provides a means of applying a pressure load with a specified spatial and temporal correlation. In many cases, the desired

random function is a function of time only. In those cases the **SamplingRandom** function provides a mechanism for applying the load.

Keyword	**Values**	**Default**	**Description**
type	string	required	must be "SamplingRandom"
cutoff_freq	*Real*	required†	cutoff frequency (in Hz)
omega_c (ω_c)	*Real*	required†	cutoff frequency (in rad/s)
DeltaT	*Real*	π/ω_c	coarse time step
ntimes	*integer*	5	# of terms in time interpolation
correlation_function	*integer*	defaults to $\sin(x)/x$	function for time interpolation
scale_function	integer	defaults to $\sigma(z) = 1$	

†Either "cutoff_freq" or "omega_c" must be specified, but not both.

Table 67: SamplingRandom function parameters

Note that $\omega_c = 2\pi \times$cutoff_freq, and that only one of the two parameters omega_c and cutoff_freq can be specified. More detailed descriptions of these parameters are given in section 2.15.13. Random time functions can be used to specify any type of random load, including pressure loads, force loads, acoustic loads, etc. Below we give an example for the case of an acoustic load.

```
LOAD
  sideset 1
     function = 1
     acoustic_vel = 1.0
END

FUNCTION 1
   type SAMPLINGRANDOM
   cutoff_freq 1000
   deltaT 8.0e-4
   ntimes 5
END
```

The SamplingRandom function is a special case of a zero mean, unit variance Gaussian function. Sampling methods allow a reduced memory method of computing the time realization. In a transient analysis, the time integration step should be less than the coarse time step, "DeltaT". Statistics for the functions may be output by specifying "input" in the "echo" section of the input file (see section 2.8).

2.29.7 RandomLib Functions

The random functions defined in section 2.29.5 provide a simple, computationally efficient method of applying a random function. However, in many cases, a random load on a

structure may need some sort of spatial correlation with other loads on the structure. The **RandomLib** function was created to address this need.[4]

Run time parameters for "RandomLib" functions are listed in Table 68, and an example is provided in Figure 41. Each parameter is described in more detail below.

As currently implemented (starting with version 2.6), the randomlib function operates only by reading data from an external exodus data file. The data file is an **Exodus** file that contains nodal scalar loadings applied to a nodeset that covers the same nodes as the sideset. This sideset is assumed to have the name "surface_1_nodes" where the "1" in this case corresponds to sideset 1. These nodal loadings are typically generated within Matlab code and merged with the **Exodus** file definition. In the future, a more complete capability will be integrated within **Sierra/SD** directly.

Currently this function has been applied only to apply a scalar function on the nodal locations of a single sideset in the model. Such functions can be used to apply pressures (which are applied as piecewise linear functions within the elements). It can also be used to apply prescribed accelerations at the nodal locations.

Keyword	**Values**	**Description**
type	randomlib	required to specify function
interp		temporal interpolation scheme
		none=nearest
		linear=linear interpolation
sideset	*integer*	sideset where pressures are applied

Table 68: RandomLib function parameters

```
function 55
    type=randomlib
    interp=none
    sideset=1
end
```

Figure 41: Example **RandomLib** Function Specification

type The specification of "`type=randomlib`" is *required* to reference the randomlib function and its capabilities.

interp The restart file contains time samples of a random function. **Sierra/SD** references these values at each time step to properly load the function. The actual value returned depends on this interpolation, as is illustrated in Figure 42.

[4] The RandomLib function is an external library interface to **Sierra/SD** . Additional functionality as well as its interface to other applications is described in separate documentation.

sideset The pressure is applied over a single sideset of the model. This sideset must match the definition in the load section.[1]

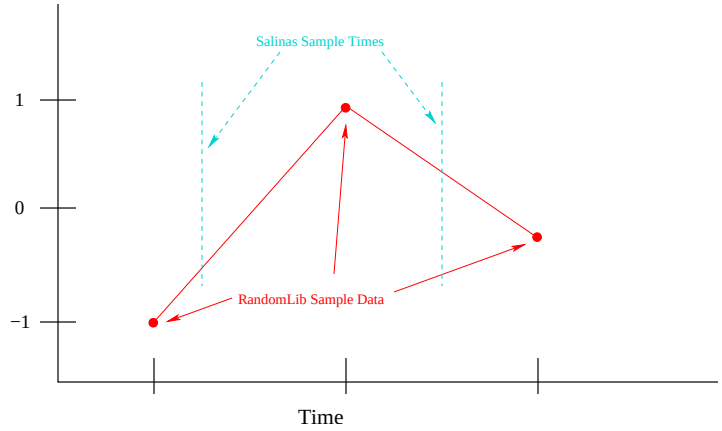


Figure 42: RandomLib Temporal Interpolation. Because of different time steps in the RandomLib data library and the **Sierra/SD** time step algorithm, the function value returned depends on the time interpolation algorithm. With "interp=none", the first value returned to **Sierra/SD** is about -1.0, as that is the nearest time sample in the data. With "interp=linear", the value returned is about -0.6. Note that round off can cause odd behavior with "interp=none", even if the two data sets have the same fundamental time step.

2.29.8 SpatialBC Functions

The Spatial boundary condition function is used when applying boundary conditions from a variable in the **Exodus** file. It is very similar to the randomlib function except that you can specify a nodeset directly instead of using a nodeset associated with the specified sideset.

The variable to read from the **Exodus** file is specified with the parameter "variable_name". For example to read accelerations in the Z direction:

```
function 1
    type = spatialBC
    nodeset 1
    variable_name AccZ
end
```

[1] The data for the **Exodus** data file is usually provided using specialized tools such as `mkrandloadrst`. The sideset provides information about the extent of the load, and for pressure loads, it is required to identify the faces upon which the load is applied. The actual time history data is associated with a nodeset which includes the same nodes as the sideset.

2.29.9 ReadNodal Functions

The **readnodal** function reads in time history data from the **Exodus** file, and applies that data as a time and spatially-dependent load. Currently this function has been applied only for reading in volume velocities for acoustic point source analysis. However, with minor modifications it could also be used for applying time and spatially-dependent force on the structure.

As currently implemented, the readnodal function operates only by reading data from an external exodus data file. The data file is an **Exodus** file that contains time histories of first or second derivatives of volume velocities applied to the entire mesh file. The name of the variable that is stored in the **Exodus** file is specified with the **exo_var** keyword. Run time parameters for **readnodal** functions are listed in Table 69, and an example is provided in Figure 43. The keyword **exo_var** must be followed by two keywords, the first one specifying whether the data is a scalar or a vector, and the second specifying the name of the variable on the **Exodus** database. Thus, Table 69 specifies that a scalar variable with the name **volume_acceleration** should be available on the **Exodus** database. The **interp** parameter is the same as was described for the **randomlib** functions. It specifies the type of temporal interpolation.

As currently implemented, the **readnodal** function first looks for a nodal variable with the specified name (i.e.,**volume_acceleration** in the example above). If no nodal variable is found with that name, it then looks for an element variable with the same name. If no element variable is found, it searches for a face variable. If none of these are found, the code will error out.

Keyword	Values	Description
type	readnodal	required to specify function
interp		temporal interpolation scheme
		none=nearest
		linear=linear interpolation
exo_var	scalar volume_acceleration	either scalar or vector, followed by variable name

Table 69: ReadNodal function parameters

```
function 55
    type=readnodal
    interp=none
    exo_var scalar volume_acceleration
end
```

Figure 43: Example **ReadNodal** Function Specification

2.29.10 ReadNodalSet Functions

The **ReadNodalSet** function reads in nodal data from a nodeset in the **Exodus** file, and applies that data as a time and spatially-dependent load. Note that this is the same functionality as the **readnodal** function, except that it only reads data on a nodeset instead of on the entire mesh. Currently this function has been applied only for reading in volume velocities for acoustic point source analysis. However, with minor modifications it could also be applied for applying time and spatially-dependent force on a structure.

Run time parameters for **ReadNodalSet** functions are listed in Table 69, and are the same as for the **readnodal**, except for the specification of a **nodeset** parameter. The force is applied over a single nodeset of the model. This nodeset must match the definition in the load section.

```
function 55
    type=readnodalset
    interp=none
    nodeset=1
    exo_var scalar volume_acceleration
end
```

Figure 44: Example **ReadNodalSet** Function Specification

2.29.11 ReadSurface Functions

The **ReadSurface** function reads in data from either the entire **Exodus** file, or a nodeset or sideset that covers the surface of interest. If a set is specified in the **FUNCTION** block, then data corresponding to that set is read in from the **Exodus** file. Otherwise, the nodal variable is read from the entire mesh as a nodal variable (rather than a nodeset variable). Once this data is read, **Sierra/SD** integrates the data over the surface to create a time and spatially-varying forcing function. One difference between this function and the **readnodal** function is that the data is from **ReadSurface** is used in a surface integration to generate the load, whereas the data from **readnodal** does not need to be integrated, and thus can be inserted directly into the force vector.

This function is used to read in surface velocities or accelerations which are used as a boundary condition for acoustic analysis. It can also be used for applying time and spatially-dependent pressure loads on a structure. For this case, the pressure output variable currently only outputs element data for pressures read from a sideset. Nodal pressures, like those used in **randomlib** functions, are output as 0.

As currently implemented, the ReadSurface function operates only by reading data from an external exodus data file. The name of the variable to be read in from the **Exodus** file must be specified in the input deck using the **exo_var** keyword. Also, the variable must

be specified to be a scalar or a vector, using the syntax given in Figure 45. An example for **ReadSurface** functions is given in Figure 45. The **interp** is the same as was described for the **randomlib** function. Note that the **sideset** keyword must match that which is specified in the Loads section.

In Figure 45, the keyword **exo_var** specifies the type of data (i.e. **vector** or **scalar**), and the name of that variable on the **Exodus** file. In the case of a vector, the name of the variable as given in the input deck should be the base name of the variable, without the suffix of 'x', 'y', or 'z'. For example, for the data given in Figure 45, a vector nodal variable with name 'vel' should be available in the **Exodus** file. Thus, the data in the **Exodus** file would have names **velx**, **vely**, and **velz**. In the case of scalar data, the base name given (i.e. **vel** in Figure 45), should match exactly the name of the nodal variable in the **Exodus** file.

```
function 55
    type=readsurface
    interp=none
    nodeset=1
    exo_var vector vel
end
```

Figure 45: Example **ReadSurface** Function Specification

2.29.12 User Defined Functions

A user defined function capability has been added to **Sierra/SD** to permit application of generic functions that cannot be readily evaluated using built in functions. Note the following.

1. User defined functions are typically quite slow.
2. By default, user defined functions are evaluated at each application point on the structure (i.e. each node in a nodeset). Thus, they must be evaluated many times. This with their slow evaluation can result in significant time for their evaluation. If you can do the problem another way, it is strongly recommended that you do so.
3. User defined functions are impossible to fully test in our test environment.
4. User defined functions may be less robust than other methods.

Sierra/SD uses the run time compiler (RTC) environment that was developed for Alegra.[30] This environment was chosen for several reasons, but the primary goals are to provide capabilities that cannot be performed in other ways, and to do with a simple, portable system.

The Alegra RTC is a library that compiles a subset of the "C" language in run time. The user is referred to the RTC documentation for details of the library. The RTC function is referenced in the **Sierra/SD** input just as any other function. For example,

```
LOADS
   nodeset 1
      function=1
      force=0 0 1
END

FUNCTION 1
   type=USER
   rtcfile='example.cc'
END
```

The function "type" is defined as "USER". In addition, the "rtcfile" parameter must be specified. The rtcfile points to the file containing the source for the function. Typically the file name has a ".cc" extension (to indicate that it is C++ source), but any filename is acceptable, and either a relative or a full path may be specified.

The permitted parameters are listed in the following table.

Parameter	**Argument**	**Description**
rtcfile	*string*	the file name containing source. *Required!*
timeonly	none	flag. If this exists, no spatial dependence is allowed in the function.

2.29.12.1 The Source File: The **rtcfile** points to a file containing source code to be compiled. This is a subset of the "C" language. There are some idiosyncrasies which we list here.

- Comment fields follow C (not c++) conventions.
- No function definitions are allowed.
- Data is passed to and from **Sierra/SD** using specifically named, predefined variables. These are listed in Table 70.

We provide an example below for the case of a force that is inversely proportional to the deformed Z coordinate of each node in a sideset. This distance is labeled R in the script, it is checked for divide by zero, a function value is computed, and the value returned in the *retvar* variable. This function will be run on all the nodes in a side set.[1]

[1] The HowTo document has an example of an RTC analysis together with suggestions on debugging.

Variable	I/O	Description
time	input	the current time for the function evaluation.
retvar	output	The function return value
coord	input	The undeformed coordinates of the node
disp	input	The deformation vector. Dimension=7
velocity	input	The velocity. Dimension=7
acceleration	input	The acceleration on this node.
nodeid	input	The global node ID of this node. *the unmapped* **Exodus** *node id (1:N)*

Table 70: Predefined RTC variables

```
double R=disp[2]+coord[2];
retval=1e10;
if ( abs(R) > 1e-10 ){
   retval = 1.0 / R;
}
```

Note that run time compile functions are currently not compatible with prescribed boundary conditions. A fatal error is encountered in **Sierra/SD** if one tries to use a run time compile function with prescribed boundary conditions.

2.29.13 Plane Wave

Plane wave functions are applied primarily in scattering problems where a load on a surface is analytically described as an incident plane wave. We define the wave in terms of the following parameters.

Keyword	Values	Description
type	plane_wave	identifier keyword
Direction	3 *reals*	direction of the wave, $\vec{d}$
material	*string*	acoustic material
K0	*real*	wavenumber, k_o
origin	3 *reals*	wave origin, $\vec{x}_o$

The material specification provides the parameters, c_o (the wave speed), and ρ_o (the fluid density). In terms of these parameters, and the pressure amplitude, P_o, which is specified in the loads section, the CW plane wave can be described as follows.

$$P = P_o \cdot \cos\left(k_o \hat{d} \cdot [\vec{x} - \vec{x}_o] - k_o c_o t\right) \tag{53}$$

Where $\hat{d} = \vec{d}/|\vec{d}|$ is the normalized direction vector. For scattering problems, a velocity is

also computed. The velocity for a plane wave is,

$$\vec{v} = \frac{P}{\rho_o c_o} \cdot \hat{d} \tag{54}$$

An example is shown in Figure 46.

```
function 65
    type = plane_wave
    Direction = 1 0 0
    Origin = 0 0 0
    K0 = 1000
    material = air
end

material air
    acoustic
    c0=332.0
    density=1.29
end
```

Figure 46: Example **PlaneWave** Function Specification

2.29.14 Planar Step Wave

The planar step wave, keyword="planar_step_wave" provides a means of applying a traveling exponential step wave to an acoustic scattering problem. The function provides both a pressure on a structure and a velocity load on an acoustic model. Parameters are listed in Table 71. The exponential step wave is useful for verification problems in scattering, but is not realizable physically. The pressure definition is similar to the plane wave, but employs a Heaviside step function, $H(t - t')$, where $t' = \frac{\hat{d} \cdot [\vec{x} - \vec{x}_o]}{c_o}$.

$$P = P_o \cdot e^{-\beta \cdot (t - t')} H(t - t') \tag{55}$$

A standard planar step wave function can be defined by using $\beta = 0$. This is the default behavior if no beta parameter is specified.

2.29.15 Spherically Spreading Wave

A spherically spreading wave, keyword=**spherical_wave**, computes the response of a point source excitation in an acoustic medium. The function applies both a pressure on

Keyword	**Values**	**Description**
type	planar_step_wave	identifier keyword
Direction	3 *reals*	direction of the wave, $\vec{d}$
material	*string*	acoustic material
origin	3 *reals*	wave origin, $\vec{x}_o$
beta	*real*	exponential decay factor, β

Table 71: Planar Step Wave Parameters

Keyword	**Values**	**Description**
Type	spherical_wave	identifier keyword
Origin	3 *reals*	wave origin, $\vec{x}_o$
C0	*real*	acoustic sound speed
reference_location	3 *reals*	reference location $\vec{R}$
material	*string*	acoustic material (alternate to C0)
pressure_function	*integer*	new function for user supplied pressures

Table 72: Spherical Wave Parameters

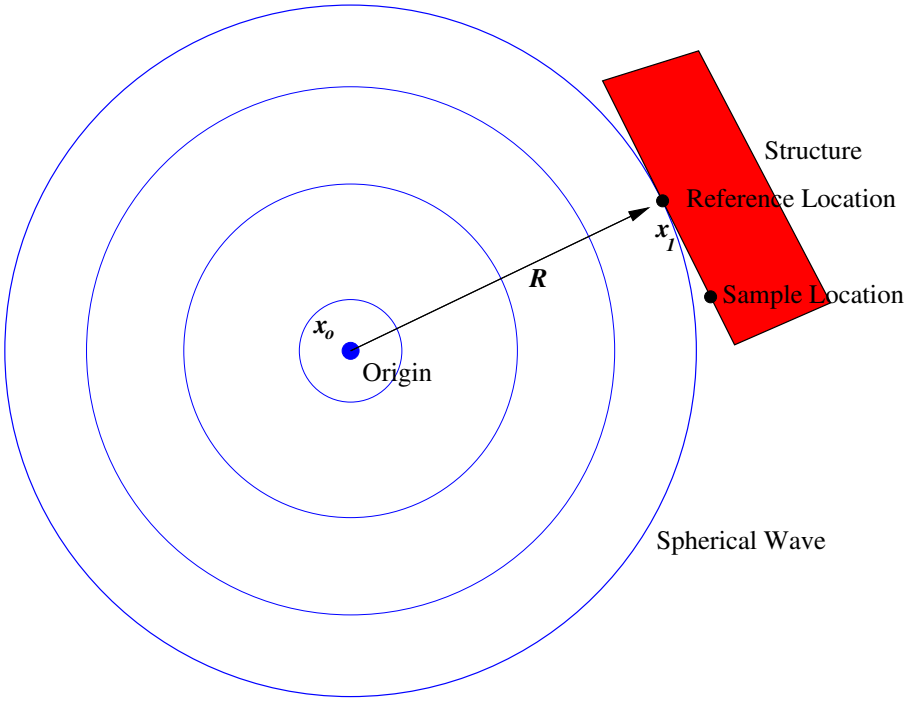


Figure 47: Spherical Wave Geometry

the structure and a velocity load on an acoustic model. Parameters are listed in Table 72. Figure 47 illustrates the geometry.

A spherical wave is used only in transient dynamics analyses. An example input is described in Figure 48. Function 1 in the example defines the spherical wave function, which describes the geometry of the loading. The time history of the loading is referenced in the function, function 11 in the example, must be a simple function of time. It could be a linear function, a runtime compiled function or a table. It cannot be a function of space and time.

```
LOAD 10
    sideset 1001
        acoustic_vel 1.0
        function = 1 //
    sideset 50000000
        pressure 1.0
        function = 1
END

FUNCTION 1
    type = spherical_wave
    origin = 0 1000 0
    pressure function = 11
    C0 = 4872 // wave velocity
    // material = 1000 //material for acoustic medium
END

FUNCTION 11
    Data 0.0 0.00000
    Data 1e-6 0.00001
    Data 2e-6 0.00002
    ...
END
```

Figure 48: Spherical Wave Example

2.29.16 Shock Wave

For Navy scattering applications, a "shock wave" function provides a numerical function for analysis of exterior shock loading. The parameters of the loading are listed in Table 73. Details of the theory and implementation are available from the Navy Surface Warfare Center, Carderock Division (NSWC/CD). An example input is shown in Figure 49.

The "free_surface_flag" indicates generation of an applicable image source above the surface of the water. The possible values of this flag are given in Table 74. In this routine, "z" is

Keyword	**Values**	**Default**	**Description**
type	shock_wave	required	identifier keyword
charge_weight	*real*	required	in pounds of TNT
charge_location	*3 reals*	required	explosive location
waterline_depth	*real*	0	
free_surface_flag	*integer*	1	see Table 74
material	*string*	required	acoustic material

Table 73: Shock Wave Parameters. input coordinates are in inches.

```
function 67
    type = shock_wave
    charge_weight = 10
    charge_location = 100.0 0.  50
    waterline_depth = 10
    free_surface_flag = 1
    material = water
end

material water
    acoustic
    c0=4872
    density=62.4
end
```

Figure 49: Example **Shock Wave** Function Specification

free_surface_flag	**Meaning**
1	similitude TNT shock without free surface
2	similitude TNT shock with free surface at waterline_depth
3	similitude TNT shock + Hick's bubble without free surface
4	similitude TNT shock + Hick's bubble with free surface

Table 74: Free Surface Flag Options

upwards and normal to the water surface. The depth is the distance below the water surface, i.e.

$$\text{waterline_depth} = z_{waterline} - z_{charge}$$

where z_{charge} is the z component of the charge location. If the free surface flag is not specified, no effects of the surface are included.

2.29.17 FSI

For fluid-structure interaction (FSI) applications, the "FSI" keyword provides a means of applying a prescribed nodal pressure load along the wetted surface. The FSI function is referenced in the **Sierra/SD** input as follows,

```
Loads
    sideset 1
        pressure 1
        scale 1
        function 1
End

Function 1
    type = FSI
End
```

The above input file assumes sideset 1 is the wetted surface. **Sierra/SD** will communicate nodal locations of the sideset to sigmaCFD. These are the locations at which pressures are sent to **Sierra/SD** . Then, **Sierra/SD** calculates a consistent load based on the values at the nodes. Finally, if restarts are needed, "restart=auto" is required in the solution section. **Sierra/SD** also supports two-way coupling for Fluid-Structure interaction. Interpolation from structural nodes to fluid nodes and from fluid nodes to structural nodes is implemented and unit tested. Figure 50 shows the infrastructure for FSI.

2.30 MATRIX-FUNCTION

This section provides for input of a matrix function as is used in a cross correlation matrix for input to a random vibration analysis. In the limit of a single input these reduce to a single function (as described in the previous section). Note that a matrix-function can have arbitrary symmetry and can be complex. An important feature of the matrix-function is that each entry of the matrix is a function of frequency (or time).

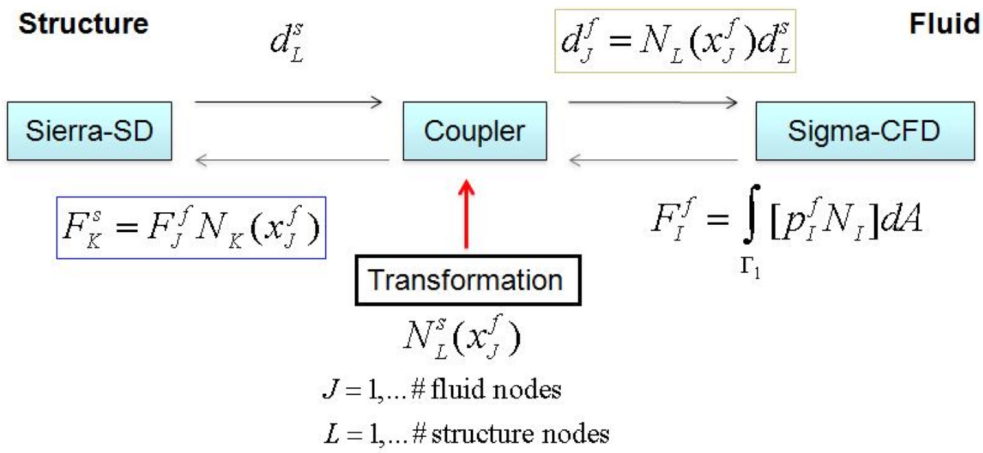


Figure 50: Fluid-Structure Interaction (FSI) Infrastructure

The Matrix-Function is illustrated in the following example.

```
MATRIX-FUNCTION 1
   name 'cross-spectral density'
   symmetry=Hermitian
   dimension=2x2
   nominalt=20.1
   data 1,1
     real function 1 scale 1.0
   data 1,2
     real function 12
     imag function 121 scale -3.0
   data 2,2
     real function 22 scale 0.5
 END
```

Matrix functions have the following parameters.

NAME allows you to optionally enter a string by which the matrix-function will be identified in subsequent messages.

SYMMETRY identifies the matrix symmetry. Options are "none", "symmetric", "asymmetric" and "Hermitian". If the matrix is not square, only "none" can apply. The default for this optional parameter is "symmetry=none".

DIMENSION specifies the dimension of the matrix. If not specified, it defaults to 1x1. The dimension is specified as the number of rows, an "x" and the number of columns. No space should be entered between the terms.

DATA Each data entry specifies one entry in the matrix-function. The *data* entry must be immediately followed by the matrix location specified as a row, column pair. Again, no spaces may be inserted in the location entry. The *data* parameters uses two keywords.

- "real" identifies the real component of the entry. It must be followed by a function reference (see section 2.29), and optionally by a scale factor.
- "imag" identifies the imaginary component of the entry. It must be followed by a function definition, and an optional scale factor.

NOMINALT Used only for echoing the matrix values. If *input* is specified as an *Echo* option (see section 2.8) general information from the matrix function are written to the log file (the *.rslt* file). If, a *nominalt* entry also exists, then the matrix entries are written for that nominal time (or frequency). Only one such output can be specified. It provides a means of checking the input to assure the matrix values are correct at a single time (or frequency) value.

2.30.1 Alternate Table Interface

For a large number of inputs, the individual specification of each function on each matrix element is both tedious and inefficient. An alternate **Table** input is provided. See section 2.31 for details about tables. Application of table input to matrix-functions requires three tables:

1. A table for the real valued data.
2. A table for the imaginary valued data.
3. A table which associates each nonzero row and column of the matrix-function with appropriate rows of the real valued and imaginary valued data.

Three new keywords are introduced:

Real Table which is a two dimensional table containing all the real valued entries for each entry in the matrix. Each column contains the frequency data for that entry.

Imag Table which is a two dimensional table containing all the imaginary (complex valued) entries for each entry in the matrix. Each column contains the frequency data for that entry.

Table Index which is a two dimensional table providing a map from the matrix elements to the data columns in the real and imag tables. This index is a 4 column table. Columns 1 and 2 are the row, column index of the matrix-function. Column 3 is the row index of the real data, while column 4 is the row index of the imaginary data. If the value in column 3 or 4 is zero then the corresponding data is zero See the example in Figure 51.

Note that the table entry has a fixed step, so the same number of values must be provided in each column.

```
Matrix-Function 1
    name 'spectral density'
    dimension=2x2
    symmetry=Hermitian
    REAL TABLE RealData
    IMAG TABLE ImagData
    TABLE INDEX idx
END

Table RealData
    size=3 550        // 550 freq samples, 3 matrix locations
    delta=1 0.5
    datafile='realdata.txt'
END

Table ImagData
    size=1 550        // 550 freq samples, 1 matrix location
    delta=1 0.5
    datafile='imagdata.txt'
END

Table idx
    size=3 4
    rowfirst          // transpose matrix for simpler input
    dataline
//  row  col  real imag
    1    1    1    0  // row=1, col=1 with 1st real data, no imag
    1    2    2    1  // row=1, col=2 2nd real data, 1st imag
    2    2    3    0  // row=2, col=2, 3rd real data row, no imag
END
```

Figure 51: Example Input for a Matrix-Function using Tables

Each matrix entry in the matrix-function must reference a row of a two dimensional table. The columns of data in the table contain the frequency response for that entry. The number of rows required for each table depends on the matrix symmetry and on the index in the "Table Index".

The table entry for matrix functions is an alternate means of providing input which is provided primarily for efficiency reasons. It cannot be mixed with the individual methods, i.e. if the table keywords are used, the "data" keyword must not be used.

2.31 Table

The **Table** section permits construction of tabular data in 1 to 4 dimensions. Tables must be referenced in other structures for their data to be useful. Tables are characterized by data structures which are sampled at uniform intervals. Tables offer the following benefits.

- They provide implicit linear interpolants for values between tables.
- They are fairly flexible structures which are more memory friendly than functions (for some applications).
- Tables are the only way to introduce multi-dimensional data.

Each Table includes a number of required and optional parameters, as shown below.

Table 75: TABLE Section Options

Parameter	Default	Description
dimension	*optional*	number of dimensions in the table
size	*required*	table size in each direction
datafile	*required*	ASCI file containing the values at each point
dataline	*required*	flag indicating that all data values will follow.
origin	zero	origin of the table (for scaling)
delta	1	interval between points in each direction
rowfirst		transpose data on input

The **dimension** identifies the shape of the table. For example, `dimension=2` indicates a table of xy values. All other quantities depend on this dimension. Note: after release 2.5, this parameter is no longer required. The dimension is extracted from the **size**.

The **size** parameters indicate the individual hypercube dimensions of the table. For example, in a table of `dimension=2`, the **size** parameter indicates the number of rows and columns in the table. The total number of entries is the product of all the terms in the size.

The **datafile** parameter contains the name of a text file containing all the data values in the table. The values are entered with the first dimension cycling faster. Thus, in a `dimension=2` table, all the entries for column 1 are first entered, followed by column 2, etc. The layout of the file is not important. Data values are read one at a time as they are separated by white space. There must be exactly the correct number entries in the file. No comments are permitted in the **datafile**, but white space is permitted.

The **dataline** parameter indicates that the tabular data is included in this file following the parameter. If **dataline** is specified then **datafile** must *not* be. The format is identical to the datafile. For performance reasons, it is best to use **dataline** for smaller data sets, and **datafile** for larger.

The **rowfirst** is provided to transpose the data on input. *It applies only to 2D tables.* If this keyword is present, then the table values will be interpreted as if the table had been transposed.

Both the **origin** and the **delta** parameters are optional values provided for interpolation. The implicit integer entries of the table are converted to real values for function evaluation by use of these parameters.

Function evaluations within the range of the table can be linearly interpolated. The range in each direction is determined by the following.

$$\text{origin}_i < \text{range}_i < \text{origin}_i + (\text{delta}_i \cdot \text{size}_i) \tag{56}$$

Evaluations of the table for regions outside the valid range result in a warning message.

In contrast to a **function** (see section 2.29), tables require memory only as needed. All processors store the full input file in memory. However, tables can store a large amount of data in the **datafile**. This file is opened and data is read from it only as needed. For this reason, tables are preferred over functions when only a few processors may need access to a large amount of data. Obviously, tables are the only option when a function of more than one variable is required.

An example of a two dimensional table definition is shown below.

```
Table example-2d-table
   dimension=2
   size    = 200 300    // note: don't put in an x
   origin    1.0 0.0    // optional. defaults to 0 0
   delta     1.0 0.9    // optional. defaults to 1 1
   datafile 'junk.txt'
 END
```

2.32 CBModel

The **CBModel** section provides a method of specifying information related to a Craig-Bampton model reduction of the entire structure. It is required by the CBR method (section 2.1.5).

The "interface" is that portion of the model which will interface to the external structure. The interface is defined by collections of nodes specified as nodesets or sidesets. After eliminating boundary conditions, the active degrees of freedom on the nodes become the interface.

Table 76: CBModel Parameters

Keyword	type	Description
nodeset	*integer*	**Exodus** nodeset. Must include the nodeset id.
sideset	*integer*	**Exodus** sideset. Must include the sideset id.
format	*string*	specifies the output format. **m**atlab - Matlab .m format **d**mig - nastran DMIG format **ne**tcdf - netcdf format†
file	*string*	specifies the file name for output.
GlobalSolution	*string*	'yes' to compute the eigen solution of the reduced system.
inertia_matrix	*string*	'yes' to compute the inertia tensor
sensitivity_method	*string*	specifies the method to compute cbr sensitivities. constant_vector - constant vector method finite_difference - finite difference method

†The netcdf format is the database upon which exodusII is built. A translator from this format to nastran `output4` format is available.

The supported keywords for the CBMODEL section are shown in Table 76. The keywords are described below.

nodeset: The **nodeset** keyword specifies the nodes to be placed in the interface. Nodesets are defined in the **Exodus** file. An integer nodeset ID must follow the nodeset keyword. Alternatively, a list of nodesets (in Matlab type format) can be specified. This is identical to the `history` file definition of section 2.10.

sideset: A **sideset** may also be used to specify the interface nodes. Any number of nodeset and sideset combinations are allowed. The interface is the union of all such entries.

format: The preferred format is the **netcdf** format. This is actually a superset of the **Exodus** format. It is the format that must be used if the reduced model is to be inserted into another **Sierra/SD** model as a superelement. The **dmig** format is for use with nastran, and will probably be dropped in the future. It contains only the reduced system matrices (no maps, coordinates, etc). The Matlab format is a convenience.

Note that the **netcdf** format may be converted to the other forms using a stand alone translator, **ncdfout**.

file: The **file** keyword is required to specify the output file name.

GlobalSolution: As a convenience, we will optionally compute the eigen values of the reduced system. It is strongly recommended that these values be compared with the eigenvalues of the full system to ensure that the model has converged over the frequency of interest.

inertia_matrix: The inertia matrix defined in section 2.1.5 is optionally computed and written to the super-element files with i the reduced mass and stiffness matrices.

sensitivity_method Currently the constant vector and finite difference methods are available for computing sensitivities for Craig-Bampton reduction. The default is the constant vector method.

The constraint modes and fixed interface modes that were used as a basis to generate the reduced order system may be output to the exodus file by specifying the "displacement" option to the **Outputs** section (2.9). The fixed interface modes are output first, followed by the constraint modes.[2] These modes may be visualized and evaluated using any of the standard tools.

Data in Table 77 will be written to a file. The Output Transfer Matrix (or OTM) depends on data in the **History** section (see section 2.10). Specifically, the output nodes and elements, and the output variables are specified in the history file *as if they were to be output to a history file.* For simplicity, and because the OTM describes a linear transfer matrix, only a limited subset of results are provided. In particular, displacements and the natural strains and stresses may be written. The transfer matrix provides the following computation.

$$\begin{aligned}\begin{bmatrix} u \\ \epsilon \\ \sigma \end{bmatrix}_{out} &= \begin{bmatrix} \text{OTM} \end{bmatrix} \begin{bmatrix} q \\ u_{int} \end{bmatrix} \\ &= \begin{bmatrix} \Phi_u & \Psi_u \\ \Phi_\epsilon & \Psi_\epsilon \\ \Phi_\sigma & \Psi_\sigma \end{bmatrix} \begin{bmatrix} q \\ u_{int} \end{bmatrix}\end{aligned}$$

Here q is the amplitude of the internal constraint modes (typically computed in the next level analysis), and u_{int} is a vector of interface displacements. The fixed interface modes (eigen modes of a clamped boundary) are represented by Φ, and the constraint modes by Ψ.

The left hand side vectors represents internal results (displacement, strain and stress) which are computed from the interface results. Any of the output results may be omitted, and the OTM will retain only nonzero components. For example, if only displacements are required, the matrix reduces to $[\Phi_u \ \Psi_u]$. The OTM matrix is a rectangular matrix, and it is typically full. An example **CBModel** section follows.

```
CBMODEL
   nodeset=1:2          // nodes from nodeset 1 and 2
   format=netcdf        // use a netcdf format file
```

[2] The eigenvalues of the fixed interface modes are associated with each mode. For the constraint modes an integer index replaces the eigenvalues.

Table 77: Data output for Craig-Bampton Reduction

<table>
<tr><th>Variable</th><th>Description</th></tr>
<tr><td>NumC</td><td>number of constraint modes</td></tr>
<tr><td>NumEig</td><td>number of fixed interface modes</td></tr>
<tr><td>Kr</td><td>Reduced stiffness matrix.</td></tr>
<tr><td>Mr</td><td>Reduced mass matrix.</td></tr>
<tr><td>Cr</td><td>Reduced damping matrix. Only available for dashpots and block proportional damping.</td></tr>
<tr><td>cbmap</td><td>A two column list providing a map from each interface degrees of freedom to the node and coordinate direction of the global model.
The first column of this list is the node number (1:N) in the structure. The second column indicates the coordinate direction as follows.
<table>
<tr><th>Number</th><th>Description</th></tr>
<tr><td>1</td><td>**x**</td></tr>
<tr><td>2</td><td>**y**</td></tr>
<tr><td>3</td><td>**z**</td></tr>
<tr><td>4</td><td>Rotation **x**</td></tr>
<tr><td>5</td><td>Rotation **y**</td></tr>
<tr><td>6</td><td>Rotation **z**</td></tr>
<tr><td>7</td><td>acoustic pressure</td></tr>
</table>
The “cbmap” has the same number of rows as Kr or Mr.</td></tr>
<tr><td>OutMap</td><td>A map of the nodes in the output transfer matrix. OutMap(i) is the global node number for each node in the output. There are always 6 rows of output for each node. Thus OutMap(1) corresponds to rows 1 through 6 in the OTM.</td></tr>
<tr><td>OTM</td><td>Output Transfer Matrix to provide a transfer function from the interface dofs to internal degrees of freedom or other results.</td></tr>
<tr><td>OutElemMap</td><td>A map of the *elements* in the output transfer matrix, OTME. OutElemMap(i) is the global element number for each element in the output. There are always 6 rows of output for each element.</td></tr>
<tr><td>OTME</td><td>Output Transfer Matrix to provide a transfer function from the interface dofs to internal elements.</td></tr>
</table>

```
   file='junk.ncf'
END
```

The reduced inertia matrix **I** for Craig-Bampton Reduction defined in section 2.1.5 may be computed and written to the super-element files with the reduced mass and stiffness matrices. **I** can be written to the results file in either netcdf, Matlab or DMIG format. In the CBMODEL section

```
CBMODEL
   nodeset 1
   format = netcdf
   file = cbmodel.ncf
   globalsolution = yes
   inertia_matrix = yes
END
```

the inertia_matrix = yes-no line requests the output of the inertia tensor. The default value of `inertia_tensor` is no.

> *NOTE:*
> *In release 2.2 we released the OTM output capability. This permits an analyst to output the reduced order model of the entire structure for use in another code that supports superelements (such as MSC/Nastran). In a subsequent release, we will add the capability to input these matrices as a superelement in* **Sierra/SD** *. At that point one could perform a Craig-Bampton reduction to generate a reduced order model of that portion of the structure. A follow up analysis could use this as a superelement. See details in Figure 52.*

2.32.1 Sensitivity Analysis for Craig-Bampton models

Sensitivity output for Craig-Bampton reduction requires both the **SENSITIVITY** block described in section 2.35, as well as the **sensitivity_method** keyword in the **CBModel** block. The **sensitivity_method** defaults to the constant vector method unless otherwise specified.

The sensitivity output from a Craig-Bampton reduction is different than that typically seen in eigenvalue or transient solutions. In the case of a Craig-Bampton reduction, the sensitivities that are output consist of partial derivatives of the reduced mass and stiffness matrices with respect to the parameters. We give a brief description here, and refer to the theory manual for further details (see section 1.14.2 of the theory manual).

The reduced stiffness matrices in Craig Bampton reduction is computed via the transformation

$$\kappa = T^T K T \tag{57}$$

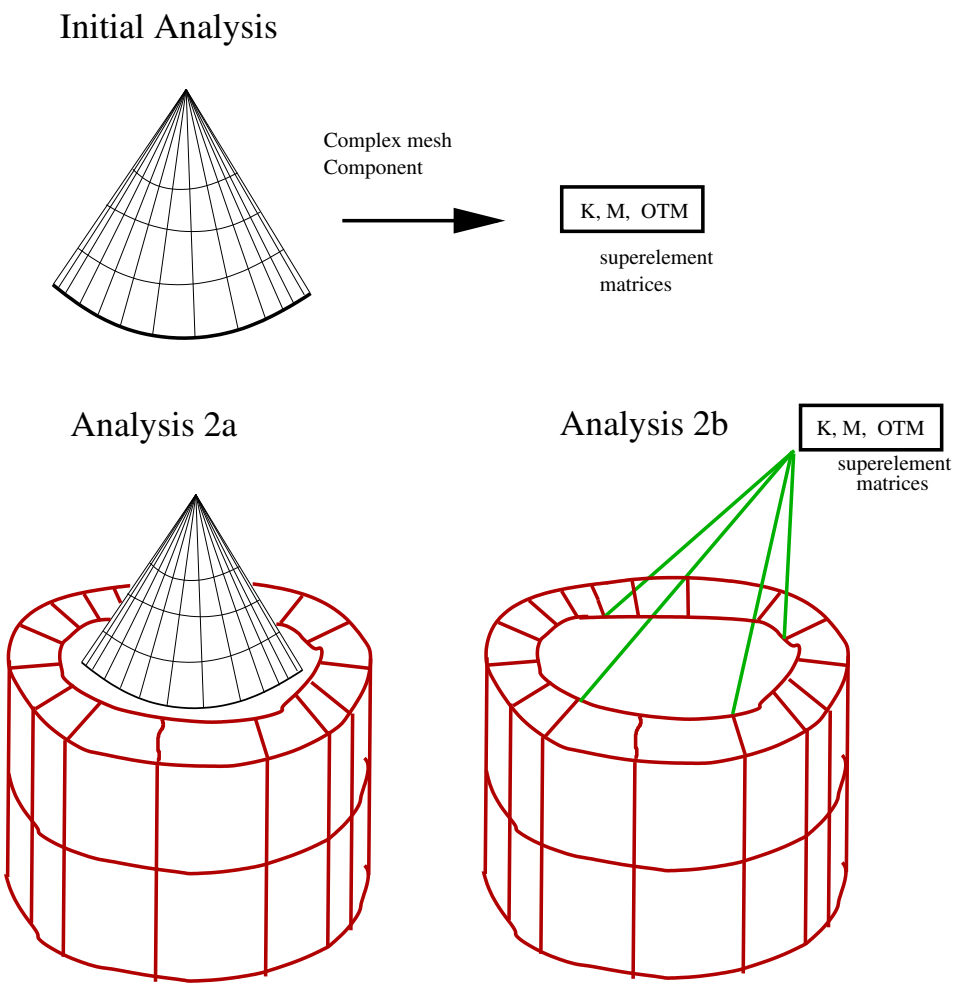


Figure 52: An initial analysis using CBR can be applied to reduce a complex component to much smaller matrices. In subsequent analyses the superelement replaces the complex component in the system analysis. There is little loss of accuracy, but significant computational benefit.

where T is the Craig-Bampton transformation matrix, K is the stiffness matrix, and κ is the reduced stiffness matrix. A similar expression can be written for the reduced mass matrix. Sensitivities of κ with respect to a parameter p can be computed with the constant vector and finite difference approaches.

The **constant_vector** approach makes the approximation that the transformation matrix $T = T_o$ does not change with the parameter p. Consequently, the sensitivity can be computed as

$$\frac{d\kappa}{dp} \approx \frac{T_o^T \left(K(p+\Delta p) - K(p)\right) T_o}{\Delta p} \tag{58}$$

Since T_o represents a truncated modal space, this approximation is not always accurate.

The **finite_difference** method uses a somewhat different approach. Given updated system stiffness $K_1 = K(p+\Delta p)$ and transformation matrix $T_1 = T(p+\Delta p)$, direct forward differences are used to evaluate the sensitivity

$$\frac{d\kappa}{dp} \approx \frac{T_1^T K(p+\Delta p) T_1 - T_o^T K(p) T_o}{\Delta p} \tag{59}$$

Unlike the constant vector approach, the finite difference method will converge to the exact sensitivities as Δp goes to zero. This is true provided that there are no repeated modes in the system.

One additional complexity that arises in sensitivity analysis of Craig-Bampton reduced models is when there are repeated modes in the transformation matrix T. Since the ordering of repeated modes in any eigenvalue problem is arbitrary, the perturbed transformation matrix T_1 could have a different ordering of the repeated modes than T_o. This would corrupt the difference operation in equation 59. However, in the case of the constant vector method this is not an issue since there is only one transformation matrix T_o. For these reasons, in the case where the user suspects repeated modes may be present in the structure, we currently recommend using the constant vector rather than the finite difference approach.

The output from a sensitivity analysis of a Craig-Bampton model is different than other types of sensitivity analysis. The output is written to either a matlab or netcdf format, depending on the **format** parameter (see Table 76). The outputted quantities are the derivatives of the stiffness and mass matrices with respect to the various parameters. Thus, if there were two sensitivity parameters p_1 and p_2, the output quantities would be

$$\frac{\partial \kappa}{\partial p_1}, \frac{\partial \kappa}{\partial p_2} \tag{60}$$

where κ would be the reduced mass and/or stiffness matrix. The dimensions of these sensitivity matrices would be the same as the dimensions of the corresponding reduced mass and stiffness matrices.

The output matrix derivatives given in equation 60 are useful for studying how the reduced matrices change with the parameters. These matrix derivatives can also be used in subsequent analysis with the corresponding superelements. For more details, we refer to section 3.41.

```
SOLUTION
    case eig
       eigen
       nmodes=500
       modalfilter=mpf1
END

MODALFILTER MPF1
    remove 1:999
    cumulative mef 0.8 0.8 0.8 0.2 0.2 0
    add 99:101,103
END
```

For this example, the following actions are performed in the filter.

1. all modes below 1000 are removed. Note, since only 500 modes are computed, all modes are removed from the list.

2. modes which contribute most to a cumulative modal effective mass are next added. Modes required for 80 percent contributions to the x, y, and z directions are added. In addition, modes needed to achieve 20 percent of the rotational terms for x and y are added. Since the contribution for rotation about z is zero, no modes are added there.

3. Finally, if modes 99, 100, 101 and 103 are not already included, they are also added.

Figure 53: Example ModalFilter Input

2.33 ModalFilter

The modal filter can be used to control the modes retained in a calculation for subsequent analysis. This can significantly improve performance with little effect on the desired response. For example, a shell structure may have many hundreds of modes contributing to the normal mode response, while only a few of these modes are likely to interact with the loads.[3]

During eigen analysis or CBR reduction, the usual number of modes (**nmodes**) are computed. These modes are filtered, and only a subset are written to the **Exodus** file or used in subsequent analysis. An example input is shown in Figure 53.

Each entry in the modal filter section consists of two parts: an action (like *remove* or *add*) and an application space. The *application space* for the "add" and "remove" space is an integer list with a format much like Matlab. See section 2.34 for more details. Valid action keywords are listed in Table 78.

[3]The modes of large ship are a good example. Only a few of the modes contribute to global bending or torsional modes. The remainder are local plate modes which may not be of interest to the analysis.

Keyword	Application Space
remove	integer list or "all"
add	integer list or "all"
cumulative mef	6 fractions
cumulative nmef	6 fractions

Table 78: Modal Filter Keywords

remove Removes modes in the application space from output.

add Adds modes in the action space.

cumulative mef Adds modes which contribute to the modal effective mass. Following this keyword sequence, 6 fractions are entered, one for each of the 6 rigid body mode contributions. The modes are sorted and modes are kept that contribute most to the modal effective mass. When the fractional contribution exceeds the threshold, no more modes are added for that direction. Contributions from each direction are combined (unioned) and added to the list of modes kept.

The 6 fractions following the keyword indicate the threshold for each coordinate direction. Each fraction must be between 0 and 1, inclusive. A value of zero means no modes are retained. A value of unity retains all modes.

cumulative nmef Adds modes which contribute to the *normalized* modal effective mass. This option is identical to the "cumulative mef" option except that the terms are normalized such that the total contribution from all computed modes sums to one.

Modal Filter Theory

Details of the computation for the modal participation factor are found in section 2.1.19. Specifically, there is one equation of interest.

$$\mathrm{MPF} = \sqrt{\sum_{i,j}(\sum_{k}\Gamma_{ij}^{k})^2} \tag{61}$$

where

$$\Gamma_{ij}^{k} = \frac{R_i^T M^k v_j}{\sqrt{(R_i^T M R_i)\,(v_j^T M v_j)}} \tag{62}$$

Here Γ_{ij}^{k} is the modal participation factor for eigenvector v_j with respect to rigid body vector, R_i, and measured in block k. Equation 61 is a measure of the system fractional modal participation factor.

2.34 Integer List

An integer list may be required as a parameter for a number of keywords. The list is of a format similar to that of Matlab. A simple list such as "1,2,3,4" is possible. One may also provide a sequence such as "1:4" which is completely equivalent to the previous example. A *step* value may also be provided, as in "2:2:20". The second term between the colons is the step. For this example, we list all even values between (and including) 2 and 20. Such combinations can also be combined, as in "1,2,3:2:7,11,13,17,19".

It is recommended that such lists have no spaces. Values can be separated with commas. However, if placed in quotes, the spaces are permitted, i.e. '1 2' is acceptable, but 1 2 without quotations is an error, and 1,2 is preferred.

2.35 SENSITIVITY

Sensitivity to parameters is available for modal analysis,[24] Craig-Bampton reduction (CBR), static solutions and transient solutions. An example input file for modal analysis is given in the Appendix 1. In the case of CBR analysis, we refer to sections 2.1.5 and 2.32 for a detailed discussion of how to perform sensitivity analysis. The **SENSITIVITY** section controls global parameters related to sensitivity analysis. Sensitivity analysis is not performed in **Sierra/SD** unless this section is present in the input file. The following example illustrates the legal keywords. Valid keywords are identified in Table 79.

```
SENSITIVITY
  values all
  vectors 1 thru 3  5  7 thru 9
  iterations 8
  tolerance 1e-7
  Attune
  AttuneNodeset 200
END
```

Keyword	argument	Description
values	string or numbers	eigenvalue selection
vectors	string or numbers	eigenvector selection
iterations	integer	number of evector iterations
tolerance	float	convergence tolerance for evectors
attune	n/a	enable attune output
attunenodeset	integer	nodeset for reduced model

Table 79: Sensitivity Analysis Keywords

The keywords **values** and **vectors** are used to control what types of sensitivities are computed for which cases in the analysis. In modal analysis, these refer to the eigenvalues and eigenvectors, respectively, and the case numbers represent the mode numbers. In static and **old_transient** analysis, **vectors** refers to the displacement vector results, and **values** has no meaning. Also, in modal analysis, eigenvalue sensitivities are always computed when eigenvector sensitivities are requested for a mode. Allowable values are:

```
vectors all              // compute for all cases/modes
vectors none             // compute for no cases/modes
vectors                  // default, same as all
vectors 1 2 3 5          // cases/modes 1,2,3,5
vectors 1 thru 3 5       // using thru to define range
```

Omitting the keyword **vectors** (or **values**) is equivalent to not requesting those sensitivities; in other words, it is equivalent to **vectors none**. The keywords **iterations** and **tolerance** are used in computing eigenvector derivatives. The default values are `10` and `1.0e-06`, respectively.

Sensitivity results are scaled by multiplying the derivative with respect to a parameter by the nominal value of that parameter. In this way, the units of the sensitivity coefficients are the same as the units of the nominal response results. Furthermore, in order to determine the absolute change in a response resulting from a relative change in a parameter, simply multiply the sensitivity of the response with respect to that parameter by the relative change. For example, multiply by 0.10 for the effect of a 10% change in the parameter.

For eigen analysis, eigenvalue sensitivity is output in the results file in two ways: as $(p)(\Delta\lambda/\Delta p)$, and in a normalized form, $(\frac{\Delta\lambda}{\Delta p})(\frac{p}{\lambda})$.

2.35.1 Attune

An interface is provided to the "Attune" test/analysis correlation code supplied by ATA engineering. The data is written to an external text file, with a file name based on the input (*.inp) file name. Attune uses a modally reduced model to provide an efficient surrogate of the full finite element model. Attune applies *only* to eigen sensitivity analysis, and the eigen modes must be selected using the **values** keyword. For output through this interface, the following two parameters must be defined.

attune: request interface output.

attunenodeset: identification of a nodeset to be associated with the test degrees of freedom. Note that even if test mode shapes are not available, "Attune" requires the definition of a reduced space model (using this nodeset). It is required for mode tracking.

For details on the use of **Attune**, please refer to the ATA website and on line documentation.[31]

2.35.2 Sensitivity Output

Sensitivity results are output to the same file as the nominal results. The arrangement of the output varies depending on the analysis. For statics, the nominal result is output, followed by the sensitivity result for each parameter. For eigenanalysis, the nominal frequencies and eigenvectors are output, followed by the eigenvalue and eigenvector sensitivities with respect to the first parameter, the second parameter, and so on. The eigenvalue sensitivities are placed in the time field of each output record, just as the frequencies are for the nominal modal parameters. For transient analysis, the nominal response for each time step is output, followed by the sensitivities for that time step. Then the nominal results for the next time step are output, and so on. See Figure 54 for an example of eigen sensitivity.

The change of parameter (or tolerance) may be specified in any of three ways.

1. Specify an absolute tolerance by entering "+/-" followed by the number, e.g. "+/- 1.05e-4".
2. Specify a relative tolerance by entering "+/-" followed by a number and the keyword "percent". Each field should be separated by a space. For example,

   56 +/- 2.0 percent
3. Use the default tolerance by entering only the "+/-" by itself. The default tolerance is 2 percent.

The selection of parameters is controlled by the inclusion of a **+/-** symbol following a parameter in the input deck. Examples of valid sensitivity parameter definitions are:

```
MATERIAL 1
  E 10e6 +/- 1e6                // absolute tolerance specified
  density 2.59e-4 +/-           // no tolerance, use default
END

BLOCK 1
  area 0.10 +/- 5 percent       // relative tolerance specified
END

BLOCK 2
  thickness  +/- 1 percent      // relative to \exodus attr
END
```

```
GROPE> select step 4
    Time = 3.504E+3 (time step 4 of 30)
GROPE> gvar
    Time = 3.504E+3 (time step 4 of 30)
    Global Time Step Variables
    ModeNumber                                  = 4.0000E+00
    EigenFrequency                              = 3.5042E+03
    EigenVectScale                              = 1.0000E+00
    dL/dElement Block 1000 area                 = 3.9362E+03
    dL/dElement Block 101 thickness             = 1.4426E+07
```

The order of parameters can be determined from the global variables. It is also available in the results file. The sensitivities may be extracted using the global variables.

```
GROPE> times
    Number of time steps = 18
            Step 1) 725.3E+0
            Step 2) 725.3E+0
            Step 3) 3.005E+3
            Step 4) 3.504E+3
            Step 5) 3.504E+3
            Step 6) 4.929E+3
            Step 7) 602.1E+0
            Step 8) 602.1E+0
            Step 9) 6.512E+0
            Step 10) 3.936E+3
            ...
```

The first NMODES (6 in this example) eigenvalues and vectors are associated with the nominal structure. The next NMODES values are the $d\lambda/dP_1$ *values associated with the first parameter,* P_1*. The corresponding vectors are* $d\phi_i/dP_1$*.*

Figure 54: Eigen Sensitivity Example Data. In this example, both eigen value and eigen vector sensitivities are computed. The data is probed using "grope". Global variable include sensitivities to area in block 1000 and thickness in block 101.

```
LOADS
  nodeset 1
  force 0. 0. 1000 +/- 0 0 10  // tolerance for vector param
END
```

Note that the tolerances are specified on the parameters where they normally appear in the input file. That is, these definitions do not appear in the **SENSITIVITY** section.

2.35.2.1 Solution Types: Sensitivity analysis is available only for the solution types shown in Table 80. The primary application is in eigen analysis where the semi-analytic solutions can provide significant computation and accuracy benefit over a finite difference approach. See 32. Transient analysis sensitivity is only available using the **old_transient** driver which solves for acceleration. The new, displacement based solver has been shown to have significantly better stability.

Table 80: Sensitivity Analysis Solution Type Availability

Name	Section	Description
eigen	2.1.11	Normal Modes
statics	2.1.30	Linear Statics
old_transient	2.1.34	Acceleration based linear transient

2.35.3 Sensitivity Limitations

As noted in the theory manual (sec 1.15), sensitivity analysis may be performed using most solvers. With the *sparsepak* solver, sensitivity analysis may *not* be performed on models with multipoint constraints.

2.36 Element Level Interface for UQ

For interaction with external optimization packages and with external uncertainty quantification (UQ) applications, it is sometimes necessary to modify design parameters on an element basis. In this mode, the structural analysis is best considered as a "black box" interaction, i.e. the UQ application knows nothing about the interior structure of the analysis. This type of analysis does not typically use the "SENSITIVITY" keyword (section 2.35), even though the UQ application may compute sensitivities.

Material properties (for linear elastic material) can be modified on an element by element basis as shown in the example of Figure 55. In this case, element variables representing the

```
Material example
    Isotropic
    E from file as "youngsmodulus"
    nu from file as "nu"
    rho 0.283
END
```

Figure 55: **UQ Element Interface.** This example would read Young's modulus and Poisson ratio from the exodus file for each element in each block using this material.

material properties are entered into an **Exodus** database, and are read and applied within **Sierra/SD** .

The keywords "from file" *must* be included as part of the input, or data in the **Exodus** file will be ignored. It is recommended that the data be entered into the **Exodus** database with a label. The keyword, "as" is used to associated the data in the **Exodus** database with the desired variable.

2.37 DAMPING

This section allows input of simple global viscous damping models, using either modal damping rates or stiffness and mass proportional damping. The various options for the DAMPING section are shown in Table 81.

Table 81: DAMPING Section Options

Parameter	Description
alpha	mass proportional damping parameter (real)
beta	stiffness proportional damping parameter (real)
gamma	uniform modal damping rate (fraction of critical) (real)
mode	individual modal damping ratio (fraction of critical) (integer, real)
ratiofun	index of function to define modal damping ratios
FilterRbm	remove rigid body mode contribution to damping
maxRatioFlexibleRbm	controls check for 6 RBM with FilterRbm

The damping matrix or modal damping coefficient is determined by summing contributions from all damping parameters given in Table 81. For modal superposition-based transient analysis, **modaltransient**, all the given parameters are defined. For linear direct implicit transient analysis, the modal damping parameters apply only to modes for which eigenvalues and eigenvectors have previously been computed. This depends on the presence of the keyword **nmodes** in the solution section of the input file. In the case of a **modalranvib** (or **modalfrf** analysis in the case of complex modes), modal damping is available, but the proportional damping parameters **alpha** and **beta** are currently ignored. We hope to lift this restriction in the future.

The effect of the mass and stiffness proportional parameters on modal damping depends on the frequencies of the modes. For modal-based analysis, the damping rate for mode i with radial frequency ω_i is given as

$$\zeta_i = \alpha/(2\omega_i) + \beta \cdot \omega_i/2 + \Gamma + \texttt{mode[i]} + ratiofun(i)$$

where the viscous damping term in the modal equilibrium equation is $2\zeta_i\omega_i$. For example the following damping input section could be used in a modal transient analysis [2].

```
DAMPING
  alpha  0.001     //
  beta 0.00005     //  C = .001 * M + .00005 * K
  gamma 0.005      // 0.5 % critical
  mode  1  0.01    // 1 % of critical
```

[2]Block specific proportional damping is also available. See section 2.25.2.

```
  mode  2  0.005  // 0.5 % critical
  mode  3  0.015  // 1.5 % critical
END
```

It produces the following damping ratios.

Mode	modal damping ratio	modal viscous damping term
1	$0.015 + 0.001/(2\omega_1) + 0.00005\omega_1/2$	$0.030\omega_1 + 0.001 + 0.00005\omega_1^2$
2	$0.010 + 0.001/(2\omega_2) + 0.00005\omega_2/2$	$0.020\omega_2 + 0.001 + 0.00005\omega_2^2$
3	$0.020 + 0.001/(2\omega_3) + 0.00005\omega_3/2$	$0.040\omega_3 + 0.001 + 0.00005\omega_3^2$

In direct transient analysis[3], the full mass and stiffness matrices are integrated for the solution. Specification of a modal damping method triggers construction of a damping contribution from the previous modal solution (using a method described in 33). This contribution is combined with other damping terms such as the proportional damping. Thus, the same damping input section would produce the damping ratios shown above for the first three modes. Modal damping is applied to modes computed in a previous solution case.[4]

The **ratiofun** keyword permits definition of modal damping terms based on a frequency dependent function. The associated function definition (see section 2.29) provides a table look up for damping ratios. For example, consider a system with modes at 200 and 500 Hz. The following example will establish modal damping ratios of .03 and .06 respectively. The function describes a line defined by $ratio(f) = 0.01 + 0.1/1000f$.

```
  DAMPING
     ratiofun=100
  END

  FUNCTION 100
     type=linear
     data 0 0.01
     data 1000 0.11
  END
```

The **FilterRbm** keyword permits proportional damping without damping the rigid body response. Thus, mass proportional damping can be used with no impact on the rigid body response. The theory behind this method of damping is described in § 1.18 of the Theory Manual. In order for this method of damping to work properly, the structure must have

[3] i.e. non-modal based, but linear transient

[4] If no previous solution case has been specified, then a default eigen analysis will be performed.

the conventional six rigid body modes of three translations and three rotations. A check of this condition is made inside of **Sierra/SD** , and a fatal error results if this condition is not satisfied. Specifically, the condition is met if

$$RatioFlexibleRbm = \frac{\|K\Phi_r\|_2}{\|K_d\|_\infty \|\Phi_r\|_2} \leq \epsilon \tag{63}$$

where K is the stiffness matrix, Φ_r is the matrix of six rigid body modes, and $\|K_d\|_\infty$ is the largest entry on the diagonal of K. The scalar tolerance ϵ can be specified using the **maxRatioFlexibleRbm** keyword.

```
DAMPING
   alpha=0.1
   FilterRbm
     maxRatioFlexibleRbm=0.001 // default is 1e-10
END
```

2.37.1 Nonlinear transient solutions with damping

Using the stiffness proportional damping parameter **beta** in a **nltransient** analysis will generate damping terms using the initial (or linear) stiffness matrix. The tangent stiffness matrix may not be used. Otherwise, the tangent matrix would be required to compute the damping terms at each iteration.

Nonlinear solutions do not support standard modal damping.

While nonlinear solutions do NOT currently support standard modal damping, they may be damping using the Distributed Damping method of the next section (2.37.2). Like modal damping, this is a system level damping model.

2.37.2 Nonlinear Distributed Damping using Modal Masing Formulation

The purpose of this formulation is to implement a subsystem or system level nonlinear distributed damping model into **Sierra/SD** . The theory on this method is found in the **Sierra/SD** Theory Manual.[24] Distributed damping is a method developed to model the nonlinear damping response of a subsystem. It implements the damping in a nonlinear manner with the use of an internal force term. The damping is modeled by either an Iwan model or a linear damper, and distributed to the subsystem by a modal expansion. This method augments the internal force vector through a modal Masing formulation.[2]

[2] Masing and Iwan models are used almost interchangeably in this document. Iwan models are a subset of more general Masing models.

Previous to the nonlinear transient solution which computes the distributed damping, eigenvectors must be computed. This is done in a previous solution 'case' option using either standard "eigen" methods or using "blk_eigen". The **blk_eigen** method is used to do an eigenvalue analysis on the subsystem blocks. This defines the subsystem by specifying the blocks in the **blk_eigen**, which is further explained in section 2.1.15.

The damping section is used to define the type of damping behavior. Currently, only two types of damping behavior are defined: a linear damper, **damper**, and an iwan model, **iwan**, see the theory manual.[24] Each mode will have a keyword defined after it with an associated parameter number. The parameters are used to define the damping behavior. If nothing is specified for a mode, then no damping for that mode is defined. An example input is shown below.

```
SOLUTION
    case 'blockeig'
      blk_eigen
        block 1:3, 5, 20
          shift -1e6
          nmodes 10
        block 4, 6:19
          shift -1e5
          nmodes 6
    case 'nonlinear'
      nltransient
        nsteps = 200
        time_step = 5.0e-3
        rho = 0.8
END


DAMPING
        mode 1 damper 1
        mode 2 damper 2
        mode 3 damper 2
        mode 4 damper 2
        mode 5 damper 2
        mode 6 damper 2
        mode 7 iwan 4
        mode 8 iwan 4
        mode 9 iwan 4
        mode 10 iwan 3
        mode 11 iwan 3
        mode 12 iwan 3
        mode 13 iwan 3
        mode 14 iwan 3
```

```
        mode 15 iwan 3
        mode 16 iwan 3
END

PROPERTY 1
    Mu = 0.001
    K = 0
END

PROPERTY 2
    Mu = 0.02
    K = 0
END

PROPERTY 3
  chi = -0.82139
  phi_max = 1.0325e-04
  R = 7.608594e+06
  S = 5.616950e+06
END

PROPERTY 4
  chi = -0.82139
  phi_max = 1.0325e-04
  R = 7.608594e+06
  S = 5.616950e+06
END
```

3 Element Library

Short descriptions of each of the types of elements follow. Most of the parameters for the element are supplied either in the database file (i.e. **Exodus** file) or in the text input file (*.inp). If parameters exist in both locations, the values specified in the text input will over ride the **Exodus** database specification.

3.1 Hex8

The **Hex8** is a standard 8 node hexahedral element with three degrees of freedom per node. The **Hex8** element has 8 integration points. The shape functions are trilinear. It supports isotropic and anisotropic materials.

There are three variations of **Hex8**. The default element is a bubble hex element, which can be specified by **Hex8b**, or by no specification at all. The bubble element still has 8 nodes and 3 degrees of freedom per node, and thus from a user's perspective it is no different than the standard **Hex8**. The **Hex8b** element uses bubble functions,[34],[35][36] to augment the standard element shape functions. It gives much better performance in bending than does the standard hex8.

The **Hex8u** specifies an under integrated Hex with properties similar to those of most commercial finite element codes. There are two versions of this element. The first is a standard under integrated element, and the second is a mean quadrature element with selective deviatoric control. In both cases, the under integration produces an element that is soft relative to a fully integrated element. Both elements are specified using by the keyword **Hex8u**, along with the parameter **sd_factor**. If **sd_factor** is specified, then the mean quadrature element with selective deviatoric is invoked with the value of **sd_factor** specified. If the parameter **sd_factor** is not specified, the standard under integrated element is invoked. For example, to use the under integrated element, one could specify

```
BLOCK
   hex8u
   material 1
END
```

On the other hand, the following block would use the mean quadrature element with a selective deviatoric parameter of 0.9

```
BLOCK
   hex8u
   material 1
   sd_factor 0.9
```

```
END
```

Note that **sd_factor** must be between 0 and 1. With a value of 0, the element is simply a mean quadrature element. With a value of 1, the element is again mean quadrature, but with fully integrated deviatoric component. More details on the theory behind these elements is given in the theory manual.

The fully integrated Hex is specified by **Hex8F**. While it performs adequately when the element shape is nearly cubic, it performs quite poorly for larger aspect ratios. For most problems involving bending the Hex8u is recommended.

The only *required* parameter for these elements is the material specification. Any material may be applied.

3.2 Hex20

The 20 node variety of Hex element provides quadratic shape functions. It is a far better element than the Hex8, and should be used if possible. The Hex20 element in **Sierra/SD** is very similar to elements found in most commercial codes. A material specification is required, and any structural material may be used.

Shape Function and Gauss point locations for the Hex20 are described in table 38, and in the theory manual (section 3.6.1).

The stress may be output at the Gauss points as described in section 2.9.12.

3.3 Wedge6

The Wedge6 is a compatibility element for the **Hex8**, it is not recommended that the entire mesh be built of **Wedge6** elements. They are primarily intended for applications where triangles are naturally generated in mesh generation. A material specification is required, and any structural material may be used.

3.4 Wedge15

The Wedge15 element adds mid-side nodes to the Wedge6. Like the Hex20 and Tet10, it has quadratic shape functions, and is a very good element. A material specification is required, and any structural material may be used.

3.5 Tet4

This is a standard 4 node tetrahedral element with three degrees of freedom per node. The **Tet4** element has one integration point. The shape functions are linear. It is not recommended to use only Tet4 elements for the entire mesh because standard, linear tetrahedron are typically much too stiff for structural applications. The **Tet4** is provided primarily for those applications where a mesh may be partially filled with these elements. If a model is constructed of all tetrahedral elements (as by an automatic mesh generator), the **Tet10** is strongly recommended over the **Tet4**.

A material specification is required, and any structural material may be used.

3.6 Tet10

This is a standard 10 node tetrahedral element with three degrees of freedom per node. The **Tet10** uses 4-point integration for the stiffness matrix and 16-point integration for the mass matrix. The shape functions are quadratic. This is a very good element for use in most structural analyses.

A material specification is required, and any structural material may be used.

3.7 QuadT

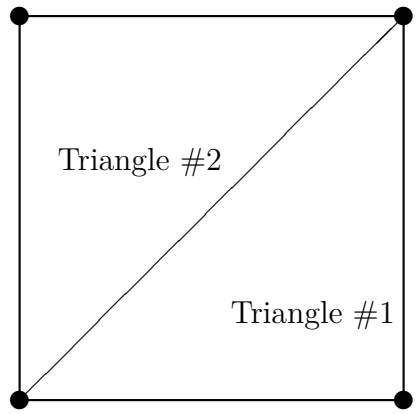
The **QuadT** is a 4-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated triangle elements, as illustrated in Figure 56. The quadrilateral is split along the shortest diagonal. It is not an optimal element, but is adequate for most applications. A more optimal element is currently under development. See the description of the **Tria3** and **TriaShell** for details on the element.

The **QuadT** may be based on either the **Tria3**, or on the **TriaShell** element depending on the material properties. The **Tria3** is used for isotropic, single layer elements. It is faster, and more robust than the **TriaShell**. However, more complex materials require use of the **TriaShell**. The underlying formulation is determined automatically by **Sierra/SD** , and cannot be selected by the user.

A material specification is required. Any linear elastic material may be used, including layered materials defined for the TriaShell.

Figure 56: QuadT Element

The element is generated by internally combining two Triangle elements.



3.8 QuadM

The **QuadM** is a 4-node quadrilateral membrane element. This element is similar to the **QuadT** element, except that it has no bending stiffness, and also in that it has no rotational degrees of freedom. It is not constructed from underlying triangles. Although it might seem that this element's behavior could be reproduced by using the **QuadT** and setting **bending_factor** to zero, it is not the case, since in that case the **QuadT** element would still retain the rotational degrees of freedom. For shell problems with very small bending stiffness, this element may be ideal, since it would not suffer from near-singularity.

The **QuadM** is a 4-noded element, where each node has three displacement degrees of freedom. For two-dimensional problems, it reduces to the standard plane elasticity element. For three-dimensional problems, it behaves like the plane elasticity element in the plane, and like a stretched balloon out-of-plane. This latter behavior results from an additional stiffness term that is applied to the out-of-plane degrees of freedom, which resembles the stiffness associated with Laplace's equation. This additional stiffness is derived in classical textbooks.[11] Note that this additional stiffness comes from the preload. Hence, if no preload is applied, the out-of-plane stiffness is zero and the element is singular.

The **QuadM** has a single required attribute, `thickness`. The remaining attributes[3] are listed in Tables 82 and 83.

Both full and selective integration methods are available for the membrane. The full integration is the default. Selective deviatoric integration can be specified by using the parameter **sd_factor**. For example, full integrated membrane, one would specify

[3] Recall that many attributes may be specified in the **Exodus** file, but may also be represented in the text input file.

Attribute	Keyword	Description
1	thickness	Thickness of the shell
2	sd_factor	selective deviatoric parameter used for numerical integration

Table 82: QuadM attributes

```
BLOCK
   QuadM
   material 1
   thickness 0.1
END
```

On the other hand, the following block would use the mean quadrature element with a selective deviatoric parameter of 0.9

```
BLOCK
   QuadM
   material 1
   sd_factor 0.9
   thickness 0.1
END
```

Note that **sd_factor** must be between 0 and 1. With a value of 0, the element is simply a mean quadrature element. With a value of 1, the element is again mean quadrature, but with fully integrated deviatoric component. More details on the theory behind these elements is given in the theory manual.

This element could be used in any situation where a preload is applied to the elements before the analysis of interest (i.e. a static preload followed by eigenanalysis), or even in cases where no preload is applied but the membranes are sufficiently constrained (i.e. two flat plates of hex elements with a layer of membrane elements in between).

This element can be used by simply specifying **QuadM** in the appropriate **Block** sections. The material input syntax for this element is the same as for the **QuadT** element. Also, this element can handle orthotropic material properties.

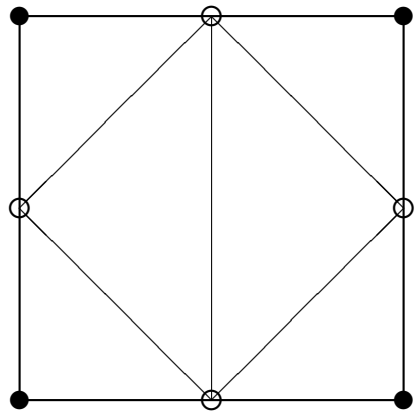
Sierra/SD example input files that use this element can be found in

```
Salinas_rtest/patchtests/quadt/quadt-patch8_test
Salinas_rtest/patchtests/quadt/quadt-patch9_test
```

This element is also commonly used in coupled simulations. In these cases, Adagio performs the preload calculation, and the preload information is passed to **Sierra/SD** for later analysis. In these cases the element is nonsingular due to the preload calculation. Examples that

Figure 57: Quad8T Element

The element is generated by internally combining six triangle elements.



use Adagio coupled with **Sierra/SD** and the **QuadM** element can be found in the sierra home area, under the following directories. These tests can be checked out by creating a tempo project under sierra.

```
tempo_rtest/tempo/membrane_free_free
tempo_rtest/tempo/membrane_free_free_par
tempo_rtest/tempo/membrane_clamped
tempo_rtest/tempo/membrane_clamped_par
```

3.9 Quad8T

The **Quad8T** is an 8-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated triangle elements (see Figure 57. It is not an optimal element, but is adequate for most applications. Shape functions are NOT quadratic. It is compatible with the **Tria6** element, as well as with other elements based on the **Tria3** or **TriaShell**. See the description of these triangle elements for details of the underlying formulation.

The **Quad8T** may be based on either the **Tria3**, or on the **TriaShell** element depending on the material properties. The **Tria3** is used for isotropic, single layer elements. It is faster, and more robust than the **TriaShell**. However, more complex materials require use of the **TriaShell**. The underlying formulation is determined automatically by **Sierra/SD** , and cannot be selected by the user.

A material specification is required. Any linear elastic material may be used, including layered anisotropic materials defined for the TriaShell.

3.10 Nquad/Ntria

The **Nquad** and **Ntria** elements are isoparametric shells with membrane and bending stiffness. They are shear-deformable elements with six degrees of freedom (DOF) per node which support the isotropic as well as the orthotropic layered cases. The formulation of the **Nquad/Ntria** is generated by decoupling the membrane and bending DOF. These elements, currently, only have linear behavior implemented. If using a non-linear solution method, these elements will not calculate a true internal force, but a linear force.

The **Nquad/Ntria** isotropic stiffness matrix is based on the plane elasticity and shear deformable (Mindlin) formulations as outlined in Reddy.[37] The layered shell stiffness matrix is based on the composite laminate formulation found in Ochoa and Reddy.[38]

The use of **Nquad/Ntria** elements requires a BLOCK definition with the **Nquad** or **Ntria** keyword, respectively. The BLOCK definition must also have a material keyword referencing the isotropic material properties (section 2.27) or orthotropic layer properties (section 2.27.3) with properties E_1, E_2, ν_{12}, and G_{12}). An example element block for a single layer isotropic material is shown below:

```
Block 2
  Nquad
  thickness 0.1
  material 2
end
Block 3
  Ntria
  thickness 0.4
  material 4
End
```

Thickness for single layer materials can be specified as attributes in the **Exodus** file or directly in the **Nquad/Ntria** section of the input file. If specified in the input file, these override the **Exodus** attribute specifications.

The stabilization method from Belytschko[39] is used for the **Nquad** element. Using single point integration $K_s^{[1x1]}$ for the shear stiffness matrix leads to hourglass modes for some problems. Using full integration $K_s^{[2x2]}$ can cause shear locking in some problems. Belytschko recommends a shear stiffness matrix given as $K_s = (1-\varepsilon)K_s^{[1x1]} + \varepsilon K_s^{[2x2]}$, a linear combination of the reduced integration and full integration shear stiffness matrices. The fraction, $\varepsilon = rt^2/A$ is a function of thickness and area. Here $r = 0.03$, t is the element thickness and A the area of the shell. This automatic selection of ε works well for very thin plates, but can be a problem for thicker elements; clearly, ε should never exceed 1. To limit shear locking, the fraction may be capped using **nquad_eps_max**, as shown in the example below.

```
Block 1
  nquad
  thickness 1
  nquad_eps_max 0.1
End
```

The value for ε is adjusted using the function $\hat{\varepsilon} = \frac{\varepsilon_{max}}{\sqrt[4]{1+\varepsilon^4}}$. This is done to address problems with 'elbow functions' in the code. Figure 58 shows this function for nquad_eps_max = 1.

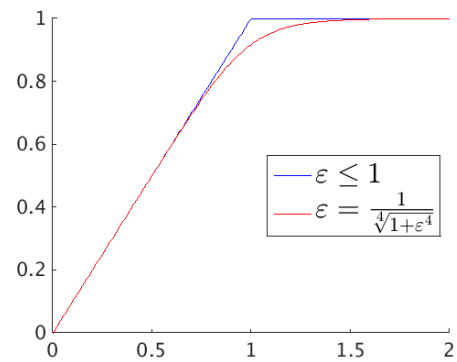


Figure 58: Function for nquad_eps_max

3.11 TriaShell

The **TriaShell** element has 3 nodes with 6 degrees of freedom (DOF) per node. The formulation of the **TriaShell** is generated by decoupling the membrane DOF and the bending DOF. Allman's Triangular (AT) element[40] models the membrane DOF, while the Discrete Kirchhoff Triangle[41] (DKT) models the bending DOF. These two elements are combined into the **TriaShell** element. The single layer shell supports only isotropic materials. A specification for a linear elastic material property is required.

The **TriaShell**, like the **Tria3**, has a single required attribute, `thickness`. The remaining attributes[4] are listed in Table 83.

Generally, users should use the **Tria3** element if possible because it is less prone to shear locking behavior, and it is somewhat faster. However, the TriaShell element must be used in these special cases.

- When the material is anything but isotropic.

[4] Recall that many attributes may be specified in the **Exodus** file, but may also be represented in the text input file. These rotational attributes are only available in the input file.

Attribute	Keyword	Description
1	thickness	Thickness of the shell
2	offset	offset for the shell
N/A	rotate about axis	
N/A	rotate about normal	
N/A	membrane_factor	scale factor for membrane
N/A	bending_factor	scale factor for bending

Table 83: TriaShell attributes

- If multiple material layers are required.

Additional attributes include two rotational parameters. The first is a rotation about a given axis, and the second is a rotation about the surface normal. The angles are specified in degrees and the axis is an integer 1, 2, or 3, representing the x, y, and z coordinate axes. The example below illustrates the use of these parameters. Figure 59 illustrates the concept.

3.12 Layered Shell

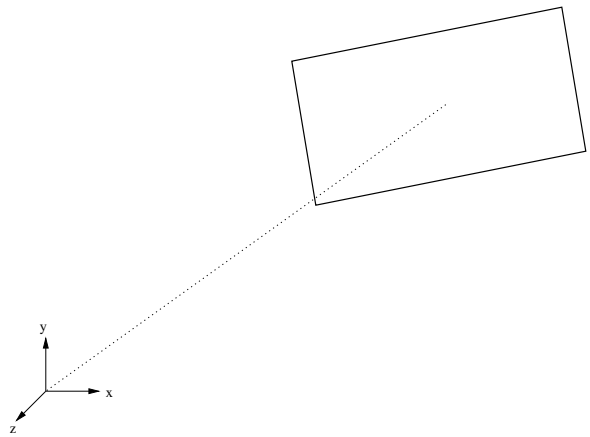
You may also specify layers for the **TriaShell** and **Nquad/Ntria** elements. When using layers, the available materials are *isotropic*, *s_isotropic* and *orthotropic_layer*. Each layer must specify a material, a thickness, and a fiber orientation. Thickness for a multilayer material must be specified layer by layer in the element section of the input file. The **Exodus** attribute may not be used. The **layer** keyword defines a new layer for the current shell. The layers of the shell will be stacked from the bottom to the top based on the order of the **layer** keyword on the input deck. The *layer_ID* input is an identifier provided by the user and is not used to select stacking order. A shell may have up to 250 different layers defined. Figure 60 shows a simple schematic explaining how layers are stacked in Sierra/SD. An example element block for a four layer orthotropic layered shell is shown below.[5]

An important parameter for the layered shells is the specification of a user-defined coordinate system with the **coordinate** option. In the example shown here, a cylindrical coordinate system is defined and the orthotropic material properties are defined using that cylindrical coordinate system and the additional **rotate** options. In the case that no user-defined coordinate system is defined, the default of the global cartesian system is used.

```
Block 2
   TriaShell
   coordinate 1
```

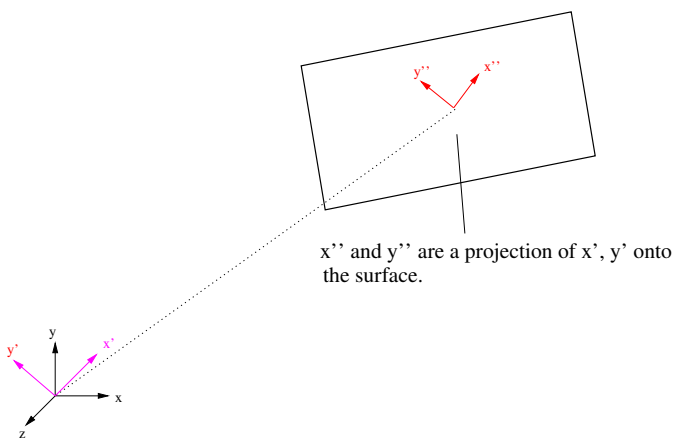
[5] For layered shells the "thickness" parameter specifies the actual thickness of that layer of the shell. This is in contrast to the HexShell which specifies a relative thickness. See section 3.16.

The coordinate frame is projected onto the surface of the shell.



A new coordinate frame is generated by rotating about the specified axis and projecting onto the element surface.

`Rotate 45`^{o} `about axis 3`



Finally, the axes may be rotated about the surface normal.

`Rotate 15`^{o} `about normal`

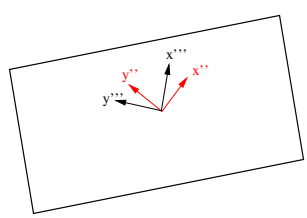


Figure 59: Shell Rotation Process

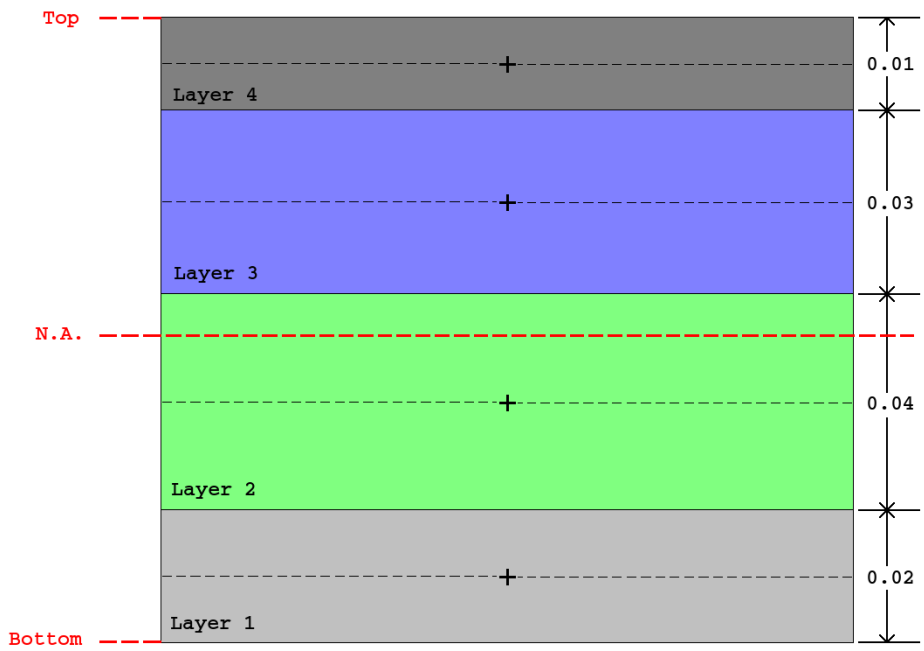


Figure 60: Stacking arrangement for a multilayer shell element.

```
   rotate 40 about axis 1
   rotate 15 about normal
   layer <layer_ID>
     material 1
     thickness 0.02
     fiber orientation 40
   layer <layer_ID>
     material 2
     thickness 0.04
     fiber orientation 44
   layer <layer_ID>
     material 3
     thickness 0.03
     fiber orientation 54
   layer <layer_ID>
     material 4
     thickness 0.01
     fiber orientation 4
End

Coordinate 1
   cylindrical
   0.0 1.0 1.0
   2.0 1.0 1.0
```

```
    0.0 1.0 10.0
END
```

Note that stress output for shells with more than one layer can be written to an **Exodus** file or can be obtained from the result file by specifying stress in the **Echo** section. The layer stresses will be computed only at the midpoint of each layer. Thus, layer stresses at the top and bottom of each layer are not currently supported.

3.13 Tria3

The **Tria3** is a three dimensional triangular shell with membrane and bending stiffness. There are 6 degrees of freedom per node. In most respects it is very similar to the **TriaShell**. It is the default element for triangular meshes. The **Tria3** was provided by Carlos Felippa of UC Boulder. It currently supports only isotropic materials. It has a single required attribute, `thickness`, which may be specified in either the **Exodus** file or the text input file.

The element stiffness matrix for triangles consists of the sum of two independent contributions from membrane and bending. These contributions may be arbitrarily scaled using the parameters **membrane_factor** and **bending_factor**. Each of these parameters default to 1.0. They must be specified in the text input file in the block definition.

Attribute	Keyword	Description
1	thickness	Thickness of the shell
2	offset	offset for the shell
N/A	membrane_factor	scale factor for membrane
N/A	bending_factor	scale factor for bending

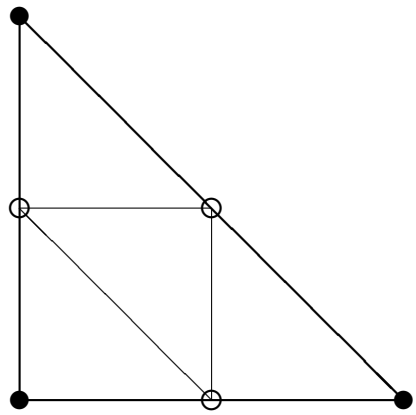
The thickness may either be entered in the **Exodus** file, or in the input file. If an attribute is entered in both locations, the value in the input file will be honored. An example element block is shown below.

```
Block 3
        Tria3
        Thickness 0.01
        material 71
        membrane_factor=0  // turns off membrane stiffness
End
```

A material for a linear elastic, isotropic material specification is required.

Figure 61: Tria6 Element

The element is generated by internally combining four Tria3 elements.



3.14 Tria6

The **Tria6** is a 6-node triangular shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements (see Figure 61. It is not an optimal element, but is adequate for most applications. Shape functions are NOT quadratic. It is compatible with the **Quad8T** element, as well as with other elements based on the **Tria3**. See the description of the **Tria3** for details on the attributes and properties required for the element.

3.15 Offset Shells

Any shell may be offset by specifying an offset. This single number is multiplied by the element normal to arrive at an offset vector. The resulting mass and stiffness properties are equivalent to the stiffness generated by translating the shell by the offset vector, and constraining the resulting offset nodes to the untranslated nodes using rigid links. The performance is vastly better than the constraint approach. Note that for curved surfaces there may be modeling issues with offset elements since there is no change in curvature with the change in radius. In the `.inp` file the element offset is specified as,

```
offset=-3.14e-2
```

Offsets may also be specified in the **Exodus** file. For shell elements these are specified in the attributes `2`. Note however, that at this time there are few tools to support model building. Refer to section 3.44 for limitations of element offsets.

3.16 HexShell

The 8 noded hexshell is a hybrid solid/shell element. It is meshed as a standard hex element, but the formulation of the element is similar to that of a shell. Unlike a shell element, the thickness is determined by the mesh. But, the element is designed to operate with many of the same features as shell elements even when it becomes very thin. Details of the element formulation are available in a separate report (Ref. 42), which can be obtained from the Sierra/SD website. An introduction to hexshells is readily available in,[24] and the verification manual[14] discusses the results of the verification problems from[42] for Sierra/SD.

The hexshell has a preferential thickness direction which is essential to it's correct operation. The thickness direction may be specified in any one of three ways.

1. Using the **tcoord**, it may be specified by a coordinate frame.
2. An **Exodus** side set may be attached to one face of all the elements in a block using the keyword **sideset**. The thickness direction will be defined to be the normal to the sideset's surface. For example, if the sideset is placed on a side of the structure that lies on the x-y plane, then the thickness direction of the hexshell will be defined as the z direction, since that is the normal to the x-y plane.
3. **Sierra/SD** may attempt to determine the thickness direction from the topology. This is the default option (because it is the easiest for the user), but it is also the least robust.

SierraSD attempts to identify the element orientation first using `tcoord`. The `tcoord` keyword abbreviates thickness coordinate, and is only defined for hexshells. If `tcoord` is not specified, then Sierra/SD attempts to identify the element orientation second from the corresponding sideset. These methods do not depend on the decomposition, but the third method does depend on the decomposition. Lastly if no sideset is specified, Sierra/SD attempts to determine the thickness direction from the topology.

The element orientation may be identified in the output using the **eorient** keyword. See section 2.9.25.

Thickness Determination by Topology

When the element thickness must be determined by the topology, the mesh must follow these requirements. The elements in the block must form a sheet. More than one disconnected portion of the sheet is possible, but all portions must adhere to these requirements.

- Every element in the sheet must have at least two neighbors, e.g. the sheet can't be a single element. NOTE... at this time, this is true for the parallel decomposed mesh as

well. The portions of the sheets found in each subdomain can not be a single element. We must be able to eliminate the thickness direction of each element by it's neighbor connectivity.

- The elements in the sheet may vary in thickness, but the sheet must be exactly one element thick.
- The elements must be connected as a single sheet. Thus, if the sheet turns a corner, it must do so gently. The algorithm will fail if any element in the sheet is connected on the top or bottom to another element in the sheet.

Determining element thickness from the topology has known limitations, and is not planned for an update. This topology method is the oldest and depends on the body being laid out in a layer one element thick. Unfortunately, it is not well parallelized, as we do not have ghost elements. The second and third methods do not depend on the decomposition.

HexShell Parameters

The **HexShell** requires a material specification. Optional parameters include the sideset or the coordinate frame and coordinate direction used to determine the thickness direction. The sideset keyword must be associated with a defined sideset in the model. The **tcoord** keyword requires two integer arguments. The first is the ID of the coordinate system referenced. The second is the direction (1, 2 or 3) associated with the coordinate system.

#	Keyword	Arguments	Description
1	sideset	ID	sideset to specify thickness direction
2	tcoord	ID and direction	coordinate frame and coordinate direction
3	autolayer	# of layers and material	creates specified number of uniform layers of specified material

An example specification for a multi-layer hexshell is shown in Figure 62.

HexShell Multilayers

The formulation of the HexShell supports multiple layers of orthotropic materials. Each layer has an associated material, normalized thickness and coordinate. The coordinate is provided to permit specification of the material coordinate. The thickness specifies the relative thickness of each layer. The total thickness is determined from the element topology, but relative thicknesses for each layer must be specified. If only one layer is specified, then the layer keyword is not required, and the relative thickness is irrelevant (and not required).[6]

[6] Layers for HexShells must specify the *relative* thickness of the layer. This is in contrast to layered shells which specify the absolute thickness (section 3.12.

```
Block 88
   HexShell
   sideset 88
   layer 1
     material 1
     coordinate 1
     thickness .4
   layer 2
     material 2
     coordinate 2
     thickness 0.6
End

BLOCK 89
   HEXSHELL
   tcoord 5 1  // use coordinate frame 5, "x" direction
   material 89
END

BLOCK 100
  HexShell
  sideset 1 // the normal to sideset 1 will be the thickness
            // direction for block 100
  material 1
END
```

Figure 62: Example HexShell Input. Block 88 is multilayer input with thickness direction determined using a sideset. Block 89 defines the thickness direction using a coordinate frame and the “tcoord” keyword.

There are two methods to specify multiple layers in a HexShell. The first, illustrated in Figure 62, provides complete flexibility over the material specification, orientation and thickness of each layer. An **autolayer** capability provides are much more limited specification that is useful for models of a single material with temperature dependence across the thickness. This `autolayer` feature creates the specified number of layers, of uniform thickness, of a single specified material. Figure 63 illustrates this specification.

Materials for all HexShell specifications can be defined as a function of temperature, with the temperatures defined through the exodus file as element variables. The temperature can vary over both the elements and layers in the block.

```
Block 1
   HexShell
   sideset 1
   autolayers 4
   material 1
End

Material 1
   name "steel1"
   E function=1
   nu .3
   density 0.288
END

FUNCTION 1
   type Linear
   data 0 30e6
   data 1e6 20e6
END
```

Figure 63: HexShell Autolayer Example. Here, exodus element variables define the temperature for each element on the block. Exodus layers must be of uniform thickness, and must be labeled "layer_temp1", "layer_temp2", etc.

When using temperature dependent materials, the temperature is obtained from the exodus file. The modulus is calculated as a function of temperature, and used in the element stiffness formulation. The temperature can vary both between layers and between elements. Any of the material parameters in either an isotropic or orthotropic material can be set to be temperature dependent. In the case of an isotropic material, any pair of two of the properties G, K, E, or ν can be temperature dependent.

The number of layers in the input file does not need to match the number of layers in the exodus file. The temperatures in the exodus file will be interpolated piecewise linearly to the center of the layer in the input file.

Temperature dependent orthotropic materials are supported for HexShells only. Temperature dependent densities are also supported.

Feature	Analytic Reference	Verification Section	Tested	Parallel Test	User Test
general	yes	4.5	Y	Y	some
multiple layers	no†	4.5	Y		

†Felippa's report contains some verification. It has not been carried into **Sierra/SD** .

Table 84: HexShell Verification Summary

The mass properties of a layered HexShell are computed approximately as follows.

1. The volume fraction, f_i, and density, ρ_i, of each layer is determined.
2. The contribution of the mass of the element is added to the nodes as if an element of density $\bar{\rho} = \sum_i \rho_i f_i$ filled the entire element.

The net affect of this is that the mass is computed as if an average density were applied. This could introduce minor errors if the element is thick and is much denser on one side than another.

For a hexshell if using `tcoord`, it is important to remember that the material definition may also use a non-default coordinate frame. In the next example, the thickness coordinate, `tcoord`, and the material defintion use the same coordinate system.

```
COORDINATE 1000
   cylindrical
   0.0 0.0 0.0
   1.0 0.0 0.0
   0.0 1.0 0.0
END
block 13
    hexshell
    tcoord 1000 1
    material 8
    coordinate 1000
end
```

3.17 Beam2

The **Beam2** element formulation is described in section 3.14 of.[24] This element is similar to the standard Nastran CBAR element, but it does not include a definition for a product

of inertia or area shear factors. A product of inertia and area shear factors are included in the CBAR element in Nastran and are supported by the **Nbeam** element described in section 3.18.

The use of a **Beam2** element requires a `BLOCK` definition with a `Beam2` keyword. The `BLOCK` definition must also have a `material` keyword referencing an isotropic material. Finally, the **Beam2** element must have a defined set of geometric parameters. Parameters for the **Beam2** element may be entered either as attributes in the mesh file or through keywords in the `BLOCK` definition. The general form of the `BLOCK` definition is as follows:

```
BLOCK block_id
   Beam2
   material = material_id
   Area = area
   I1 = inertia_about_1
   I2 = inertia_about_2
   J = polar_moment_inertia
   orientation = x_orient y_orient z_orient
   offset = x_offset y_offset z_offset
 END
```

We will discuss the various keywords in the above `BLOCK` definition in following sections.

Before describing the parameters for the **Beam2** element, it is necessary to define the local coordinate system that is set up for beam elements in general. The local coordinate system is defined by three axes – $\mathbf{x}_{elem}$, $\mathbf{y}_{elem}$, and $\mathbf{z}_{elem}$. The $\mathbf{x}_{elem}$-axis lies along the length of the beam. The other two axes, the $\mathbf{y}_{elem}$-axis and $\mathbf{z}_{elem}$-axis, are determined by an orientation vector, $\mathbf{V}$. The local coordinate system and the orientation vector are shown in Figure 64. The orientation vector $\mathbf{V}$ lies in the plane defined by the $\mathbf{x}_{elem}$-axis and the $\mathbf{y}_{elem}$-axis – plane 1 in Figure 64. The $\mathbf{z}_{elem}$-axis is derived from the orientation vector $\mathbf{V}$ and the $\mathbf{x}_{elem}$-axis by taking the cross-product $\mathbf{x}_{elem} \times \mathbf{V}$. Once the $\mathbf{z}_{elem}$-axis is calculated, the cross-product $\mathbf{z}_{elem} \times x_{elem}$ gives the $\mathbf{y}_{elem}$-axis.

The $\mathbf{x}_{elem}$-axis and $\mathbf{z}_{elem}$-axis define plane 2 in Figure 64. The $\mathbf{y}_{elem}$-axis, which lies in the 1-plane, corresponds to a local 1-axis defined in a cross-sectional plane, a plane normal to the $\mathbf{x}_{elem}$-axis. The $\mathbf{z}_{elem}$-axis, which lies in the 2-plane, corresponds to a local 2-axis defined in the cross-sectional plane.

The **Beam2** element requires that a number of geometric parameters be defined. A cross-sectional area, `area`, must be defined. The cross-sectional area can be defined with an `AREA` keyword. Two bending moments of inertia are also required. A bending moment of inertia for the 1-plane (bending about the $\mathbf{z}_{elem}$-axis) is defined by the **I1** keyword. Bending moments in the 2-plane (or bending about the $\mathbf{y}_{elem}$-axis) is defined using the **I2** keyword.[7]

[7] Note that the I1 and I2 are the bending moments in their corresponding planes, and NOT bending about their axes. This convention is consistent with many commercial codes including nastran.

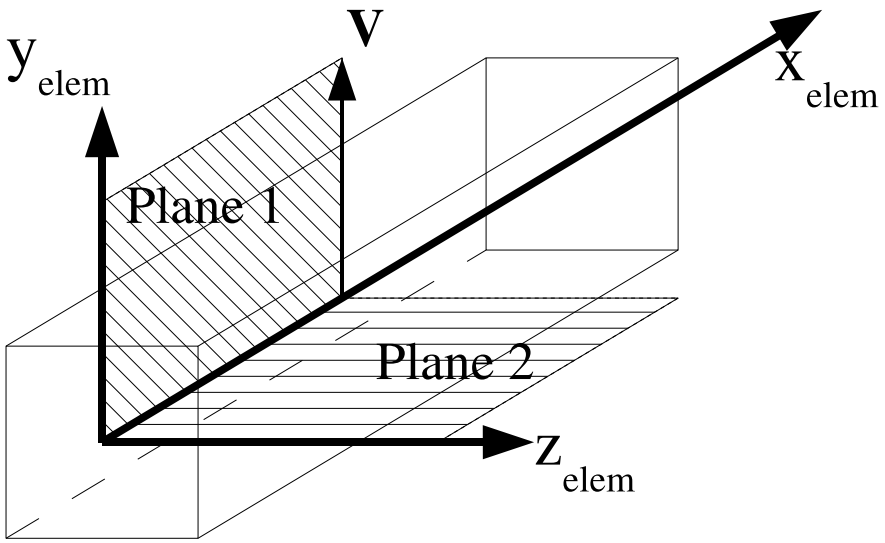


Figure 64: Beam Orientation and Local Coordinate System.

A polar moment of inertia, `polar_momnt_inertia`, for torsion about the $\mathbf{x}_{elem}$-axis is required. The polar moment of inertia can be defined with the `J` keyword.

The specification of the orientation vector **V** is optional if the cross-section of the beam is completely symmetric. Otherwise, the orientation vector must be specified to assure that the bending properties of the beam have the correct global orientation relative to the rest of the structure. The components of the orientation vector can be specified with the values `x_orient`, `y_orient`, and `z_orient` using an `ORIENT` key word.

By default, at the end of a beam, the point where the two bending axes cross (the origin of the 1,2 coordinate system at the end of the beam) coincides with the grid point at the end of the beam. We can shift the geometric location of the point where the two bending axes cross away from the grid point by specifying a an offset vector $\mathbf{V}_{off}$. This offset vector is shown in Figure 65. For the **Beam2** element, the same offset vector is applied to both ends of the beam. The `OFFSET` keyword is optional. The offset vectors move the beam neutral axis (the $\mathbf{x}_{elem}$-axis) off the line that passes between the two grid points defining the connectivity of the beam. An offset is defined by a vector with values `x_offset`, `y_offset`, and `z_offset`. These values are associated with an `OFFSET` keyword.

When the offset option is used, the offset stiffness properties are equivalent to the stiffness generated by translating the beam by the offset direction and constraining the resulting offset nodes back to the untranslated nodes using rigid links. In addition, the offset mass properties are equivalent to the mass generated by translating the beam by the offset direction and constraining the resulting offset nodes back to the untranslated nodes using rigid links. For

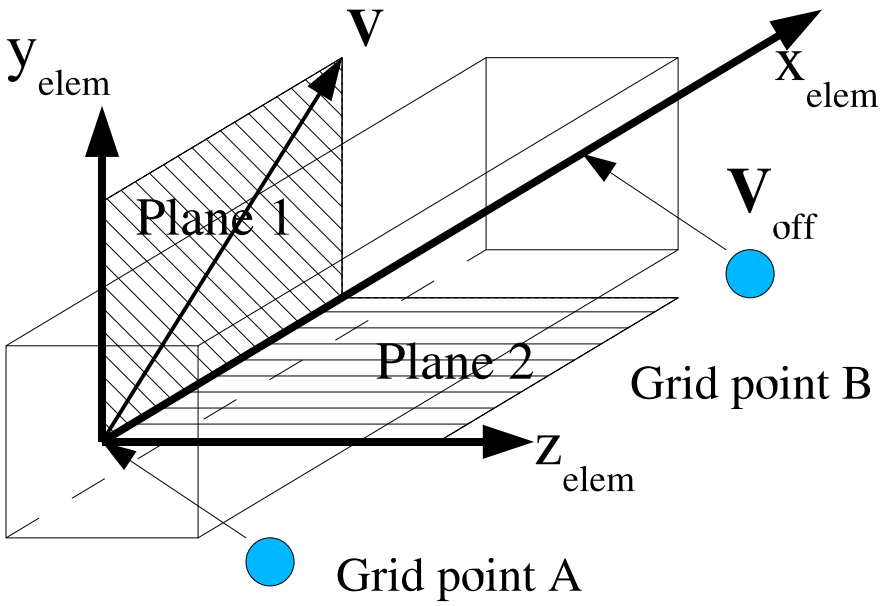


Figure 65: Beam Offset and Local Coordinate System.

the **Beam2** element, *only* the component of the offset vector orthogonal to the element is used to compute the offset behavior for both the stiffness and mass.

Note that for curved surfaces there may be modeling issues with offset elements, since there is no change in curvature with the change in radius. Refer to section 3.44 for limitations of element offsets.

The parameters just described, `area`, `inertia_about_1`, `inertia_about_2`, `polar_momnt_inertia`, `x_orient`, `y_orient`, `z_orient`, `x_offset`, `y_offset`, and `z_offset`, can also be defined as attributes in the mesh file. Attributes in the mesh file must be in the order specified in Table 85. If an attribute is entered in both the mesh file and the input file, the value in the input file will supersede the value in the mesh file. Two attribute orderings are currently supported in **Sierra/SD** because of inconsistencies in pre-processing tools. See the discussion on "OldBeam" in section 2.3.

The **Beam2** element is restricted to isotropic materials. No stress or strain output is available for the **Beam2** element.

The following section illustrates the use of the `Beam2` keyword in an element block definition. The element block has an integer block identifier of 3. This element block must consist of two node elements.

```
BLOCK 3
   Beam2
```

Table 85: Attributes for Beam2

#	old order #	Keyword	Description
1	1	Area	Area of beam
2	5	I1	First bending moment
3	6	I2	Second bending moment
4	7	J	Torsion moment
5,6,7	2,3,4	Orientation	orientation vector
8,9,10	8,9,10	offset	beam offset vector

```
   Material 7
   Area 0.71
   I1 .05
   I2 5e-2
   J 0.994
   Orientation 1.0 -1.0 0.9
   Offset -3.14e-2 0.11 0.99
END
```

3.18 Nbeam

The **Nbeam** element was developed from COSMIC/Nastran's open source CBAR element. Unlike the **Beam2** element discussed in the previous section, the **Nbeam** element includes a definition for a product of inertia and definitions for area shear factors. The Nbeam element, currently, only has linear behavior implemented. If using a non-linear solution method, the Nbeam element will not calculate a true internal force, but a linear force.

The use of a **Nbeam** element requires a `BLOCK` definition with a `Nbeam` keyword. The `BLOCK` definition must also have a `material` keyword referencing an isotropic material. Finally, the **Nbeam** element must have a defined set of geometric parameters. Most parameters for the **Nbeam** element may be entered either as attributes in the mesh file or through keywords in the `BLOCK` definition. Some parameters can be reset from default values only by use of the keyword definitions in the `BLOCK` definition. The general form of the `BLOCK` definition is as follows:

```
BLOCK block_id
   Nbeam
   material = material_id
   Area = area
   I1 = inertia_about_1
```

```
    I2 = inertia_about_2
    J = polar_momnt_inertia
    I12 = product_inertia_12
    Shear_factor_1 = sfactor1
    Shear_factor_2 = sfactor2
    orientation = x_orient y_orient z_orient
    offset = x_offset y_offset z_offset
  END
```

The various keywords in the above `BLOCK` definition are described in following paragraphs.

Local Coordinate Frame: Before describing the parameters for the **Nbeam** element, it is necessary to define the local coordinate system that is set up for beam elements in general. The local coordinate system is defined by three axes – $\mathbf{x}_{elem}$, $\mathbf{y}_{elem}$, and $\mathbf{z}_{elem}$. The $\mathbf{x}_{elem}$-axis lies along the length of the offset beam. The other two axes, the $\mathbf{y}_{elem}$-axis and $\mathbf{z}_{elem}$-axis, are determined by an orientation vector, $\mathbf{V}$. The local coordinate system and the orientation vector are shown in Figure 66. The orientation vector $\mathbf{V}$ lies in the plane defined by the $\mathbf{x}_{elem}$-axis and the $\mathbf{y}_{elem}$-axis – plane 1 in Figure 64. The $\mathbf{z}_{elem}$-axis is derived from the orientation vector $\mathbf{V}$ and the $\mathbf{x}_{elem}$-axis by taking the cross-product $\mathbf{x}_{elem} \times \mathbf{V}$. Once the $\mathbf{z}_{elem}$-axis is calculated, the cross-product $\mathbf{z}_{elem} \times x_{elem}$ gives the $\mathbf{y}_{elem}$-axis. As the NBEAM supports arbitrary vector offsets at each end, the orientation of the *offset beam* may differ from the orientation of the unoffset geometry (see "offset" below).

The $\mathbf{x}_{elem}$-axis and $\mathbf{z}_{elem}$-axis define plane 2 in Figure 64. The $\mathbf{y}_{elem}$-axis, which lies in the 1-plane, corresponds to a local 1-axis defined in a cross-sectional plane, a plane normal to the $\mathbf{x}_{elem}$-axis. The $\mathbf{z}_{elem}$-axis, which lies in the 2-plane, corresponds to a local 2-axis defined in the cross-sectional plane.

Area: The cross-sectional area, `area`, must be defined either as exodus attributes or in the "block" section. The cross-sectional area can defined with an `AREA` keyword.

Bending Moments: The bending moments of inertia about orientation axes must be defined either in the exodus file, or the "block" section. A bending moment of inertia about the 1-axis (the local cross-sectional axis corresponding to the $\mathbf{y}_{elem}$-axis), `inertia_about_1`, can be defined with the `I1` keyword. A bending moment of inertia about the 2-axis (the local cross-sectional axis corresponding to the $\mathbf{z}_{elem}$-axis), `inertia_about_2`, can be defined with the `I2` keyword. Finally, a polar moment of inertia, `polar_momnt_inertia`, for torsion about the $\mathbf{x}_{elem}$-axis is required. The polar moment of inertia can be defined with the `J` keyword.

The **Nbeam** element supports a product of inertia specification. The product of inertia about the 1,2-axes, `product_inertia_12`, is specified with the keyword `I12`. If the `I12` keyword does not appear, the value for `product_inertia_12` defaults to zero.

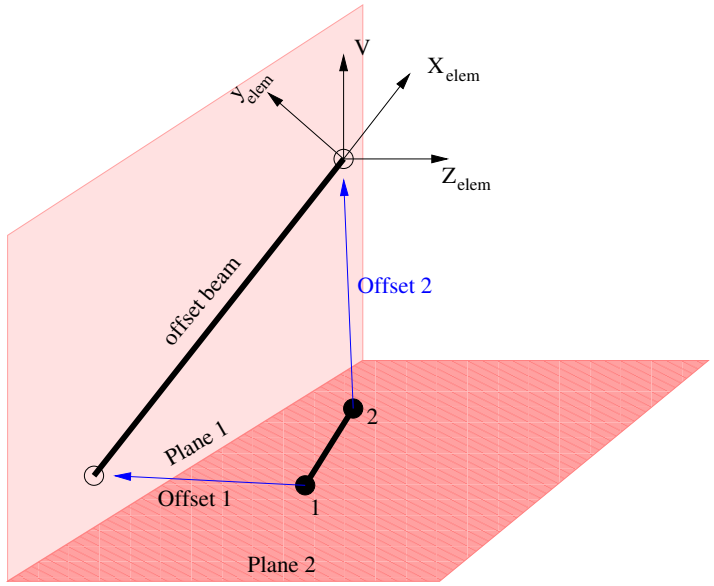


Figure 66: NBeam Orientation, Offset and Local Coordinate System. The coordinate system is in the plane of the *offset* beam. The plane is defined by the offset beam and the orientation vector, $\vec{V}$.

Shear Factor: The **Nbeam** element also has two area shear factor specifications. An area shear factor is a constant by which an average shearing strain on a beam cross-section must be multiplied in order to obtain the same transverse shear displacement as the transverse shear displacement that will be obtained from the actual shear strain distribution for the cross-section. Typically, the shearing strain will vary over a cross-section rather than being uniform distribution. See Oden (Ref. 43) for a discussion of shear factors. An area shear factor for shear in the 1-direction, `sfactor1`, is specified with a `Shear_factor_1` keyword. If no `Shear_factor_1` keyword appears, the value for `sfactor1` defaults to 1.0. An area shear factor for shear in the 2-direction, `sfactor2`, is specified with a `Shear_factor_2` keyword. If no `Shear_factor_2` keyword appears, the value for `sfactor2` defaults to 1.0.

Orientation: The orientation vector **V** must be specified to assure that the bending properties of the beam have the correct global orientation relative to the rest of the structure. The components of the orientation vector can be specified with the values `x_orient`, `y_orient`, and `z_orient` using an `Orientation` keyword.

Offset: By default, at the end of a beam, the point where the two bending axes cross (the origin of the 1,2 coordinate system at the end of the beam) coincides with the grid point at the end of the beam. We can shift the geometric location of the point where the two bending axes cross away from the grid point by specifying a an offset vector $\mathbf{V}_{off}$. This offset vector is shown in Figure 65. For the **Nbeam** element, the same offset vector is applied to both ends of the beam. The `OFFSET` keyword is optional. The offset vectors move the beam neutral axis (the $\mathbf{x}_{elem}$-axis) off the line that passes

Table 86: Attributes and Parameters for Nbeam

#	Keyword	Description
1	Area	Area of beam
2	I1	First bending moment
3	I2	Second bending moment
4	J	Torsion moment
5,6,7	Orientation	orientation vector
8,9,10	offset	beam offset vector
11,12,13	–	offset of second node
–	I12	product of inertia
–	Shear_factor_1	shear factor 1-direction
–	Shear_factor_2	shear factor 2-direction

between the two grid points defining the connectivity of the beam. An offset is defined by a vector with values `x_offset`, `y_offset`, and `z_offset`. These values are associated with an `OFFSET` keyword.

When the offset option is used, the offset stiffness properties are equivalent to the stiffness generated by translating the beam by the offset direction and constraining the resulting offset nodes back to the untranslated nodes using rigid links. For the **Nbeam** element, the full offset vector is used to compute the offset behavior, and different offsets may be applied at each end. (This behavior is different from the **Beam2** element, in which only the component of the offset vector orthogonal to the element is used to compute the offset behavior).

Note that for curved surfaces there may be modeling issues with offset elements, since there is no change in curvature with the change in radius. Refer to section 3.44 for limitations of element offsets.

Many of the parameters just described can also be defined as attributes in the mesh file. Attributes in the mesh file must be in the order specified in Table 86. If an attribute is entered in both the mesh file and the input file, the value in the input file will supersede the value in the mesh file.

The **Nbeam** element is restricted to isotropic materials. No stress or strain output is available for **Nbeam** elements.

The following section illustrates the use of the `Nbeam` keyword in an element block definition. The element block has an integer block identifier of 3. This element block must consist of two node elements.

```
BLOCK 3
   Nbeam
```

```
    Material 7
    Area 1.92
    I1 2.57375
    I2 4.81277
    J 0.025816
    I12 -1.45983
    Shear_factor_1 0.44021
    Shear_factor_2 0.33313
    Orientation 1.0 0.0 0.0
    Offset 0.5 0.5 0.5
END
```

3.19 OBeam

These beams are provided by Carlos Felippa of UC Boulder. They are similar to the simple beams of **Beam2**. They use identical parameters. Because of this duplication, these beams will probably be eliminated in the future.

3.20 Truss

This is the definition for a **Truss** element based on Cook (Ref. 18). Trusses have stiffness in extension only. The **Truss** has 1 attribute as shown in the table. A linear elastic, isotropic material is required.

#	Keyword	Description
1	Area	Area of truss

No stress or strain output is available for trusses.

3.21 Ftruss

The **Ftruss** is a simple truss with a stiffness that is defined using a function. Typically the function is a user defined function (also called a run time compiled function or RTC). See section 2.29.12.

Trusses have stiffness in only the axial direction. While they exist in a 3 dimensional world, forces orthogonal to the axial direction result in no resistance, i.e. they are singular. The axial force for an **Ftruss** element is defined as,

$$\vec{F}(\vec{L}_n, t) = -K(|\vec{L}_n|, t_n)\,\vec{L}_n \tag{64}$$

where $\vec{L}_n$ is the vector from the first point to the second at time t_n. Note that $|\vec{L}_n|$ is the instantaneous length of the truss. The force is *always* in the direction of the instantaneous element.

Note that K a constant is NOT the expression for a standard truss. Rather, $F = -K_o\,dx$ implies that $K = \frac{K_o dx}{L_o + dx}$, where L_o is the nominal truss length and K_o is the stiffness of a standard truss. The definition in equation 64 is necessary so a force may be applied when dx is zero, as in an electrostatic force for example.

If a standard (non user) function is used, the stiffness is a function of truss extension only. It may not be a function of both extension and time.

Input to the **Ftruss** element is similar to that for the truss element. The attributes and parameters are listed in table 87, and a demonstration example is provided below.[8]

```
BLOCK 88
     Ftruss
     function 88
     scale 1.0
     material 17      //optional material
     area 0.01        //area required iff material defined
END
```

If the material keyword is not found, no mass matrix is generated for the element. If a material is found, then *area* must also be defined. Like a standard truss, *area* is the first **Exodus** attribute. The area and material properties are used only to compute the mass properties of the element, and may be omitted.

The *scale* term can be defined using the input file, or alternatively, it may be defined using the second **Exodus** attribute.

3.22 ConMass

Concentrated masses are used to apply a known amount of mass at a point location. Because many meshing tools build beams as a building block for **ConMass**, the geometry definition may be either a line or a point, i.e. the **Exodus** file element types are **BEAM**, **BAR**,

[8] Recall that attributes are ordered data that may be specified in the **Exodus** file, providing a variable which changes with each element. Parameters may be specified in the input file, and are applied uniformly to all elements in the block.

Table 87: Ftruss Attributes and Parameters

#	Name	Type	Default	Comment
1	Area	*Real*	0	required if a material is specified.
2	Scale	*Real*	1	multiplier for the function
-	Function	*int*	*required*	function identifier (see section 2.29)
-	Material	*string*	*optional*	If the material specification is provided, it must point to a valid material (sec. 2.27), and an area must also be provided.

TRUSS or **SPHERE**. If a line-type element is used, all the mass is associated with the *first* node of the element.

Parameters for the **ConMass** are listed below. Because of difficulties in translation or generation of the model, the parameters found in the **Exodus** file are not normally used for a **ConMass**. This avoids the confusion generated when mass constant defaults may have been taken from beams for example. As a result, all parameters must be specified in the input or the analysis will fail.

This behavior can be tedious however, if many concentrated masses are found in the model, and if the analyst is confident that the attributes are appropriate for these elements. In this case, use the **ConMassA** element. It is identical to the ConMass, but uses the default attributes from the **Exodus** file. Typically seven attributes would be specified there.

A concentrated mass must have a mass. If no inertia tensor is specified, then the concentrated mass has 3 degrees of freedom, displacements. On the other hand a concentrated mass with an inertia tensor has both displacement and rotational degrees of freedom.

#	keyword	Description
1	Mass	concentrated mass
2	Ixx	xx moment of inertia
3	Iyy	yy moment of inertia
4	Izz	zz moment of inertia
5	Ixy	xy moment of inertia
6	Ixz	xz moment of inertia
7	Iyz	yz moment of inertia
8,9,10	offset	offset from node to CG

As an example element block,

```
Block 5
        ConMass
        Mass 1000.0
        Ixx 1.0
```

```
        Iyy 2.0
        IZZ 1.5
        offset 30.0 40.0 50.0
End
```

The **ConMass** moments of inertia are defined at the location of the **ConMass**. The offset can be used to specify inertial terms about a different point.

A **ConMass** element will activate either 3 or 6 degrees of freedom on the node the mass is located. Every **ConMass** element will activate "DispX", "DispY", and "DispZ". A **ConMass** element with non-zero inertial terms or an offset will activate "RotX", "RotY", and "RotZ". In a case such as a spring-mass system where only one translational degree of freedom is desired, the mass should be constrained in the other directions. If **ConMass** elements are attached to solid elements, through shared nodes or a 2D element, either the inertial terms should be set to zero or the rotational degrees of freedom should be constrained. Failing to properly constrain the **ConMass** may result in a solver out of bounds error or incorrect results.

3.23 Spring

The **Spring** element provides a simple spring connection between two nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the **Exodus** database using `BEAM` or `BAR` elements.

The **Spring** element has three required parameters (the translational spring stiffnesses). Rotational parameters are supported using the **RSpring** element described in section 3.24. Currently there is no way to attach off-diagonal elements, i.e. there is no K_{xy} spring element. If that is required, a combination of a spring and a multi-point constraint must be used.

Springs can be defined in user defined coordinate systems.

#	Keyword	Description
1	Kx	translational spring constant in X
2	Ky	translational spring constant in Y
3	Kz	translational spring constant in Z

As an example element block,

```
Block 51
      Spring
      Coordinate 7
```

```
      Kx 1e6
      Ky 1.11E7
      Kz 1000
End
```

3.23.1 Spring Parameter Values

It is strongly recommended that all three values of the spring constants be nonzero. This is especially important in parallel analysis performed using domain decomposition. Many domain decomposition tools may partition the model such that zero spring constants lead to singular domain stiffness matrices. This is true even if other elements may eliminate the singularity. This can cause the solver (particularly FETI) to fail.

While setting nonzero spring stiffness helps to avoid solver problems, the underlying domain decomposition problems still exist for parallel calculations. At the time of this writing, all available domain decomposition tools have difficulty with linear elements and particularly with springs. This invariably leads to load balance problems, and may introduce other problems. In many cases in large models, it may be better to replace the spring elements by solid element meshes which more accurately represent the physical connection. While there are more degrees of freedom in the calculation, the accuracy is enhanced, and domain decomposition problems are largely eliminated.

3.24 RSpring

The **RSpring** element provides a simple rotational spring connection between two nodes in a model. It is usually preferable to have the nodes of the spring be coincident. RSprings are defined in the **Exodus** database using `BEAM` or `BAR` elements.

The **RSpring** element has three required parameters (the rotational spring stiffnesses). It is strongly recommended that all three components have some stiffness. This is particularly important when doing parallel analysis (see the discussion in section 3.23.1). Translational stiffness require the use of the **Spring** element described in section 3.23. Currently there is no way to attach off diagonal elements, i.e. there is no K_{xy} spring element. If that is required, a combination of an **RSpring** and a multi-point constraint must be used.

RSprings can be defined in user defined coordinate systems. The relevant parameters are listed in the table.

#	Keyword	Description
1	Krx	rotational spring constant in X
2	Kry	rotational spring constant in Y
3	Krz	rotational spring constant in Z

As an example element block,

```
Block 52
      RSpring
      Coordinate 7
      Krx=1e6
      Kry = 1.11E7
      Krz 0.1
End
```

3.25 Spring3 - nonlinear cubic spring

The **Spring3** element provides a nonlinear spring connection between nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the **Exodus** database using `BEAM` or `BAR` elements.

The **Spring3** element has nine required parameters (the translational spring stiffnesses). There is no way to attach off diagonal elements, i.e. there are no K_{xy} spring elements. If that is required, a combination of a spring and a multi-point constraint must be used.

The force applied by the **Spring3** is defined as a cubic polynomial in each of the coordinate directions. Thus,

$$F_x = Kx1 \cdot u_x + Kx2 \cdot u_x^2 + Kx3 \cdot u_x^3 \tag{65}$$

For linear analyses, only the first term is used.

Cubic springs may be defined in user defined coordinate system.

#	Keyword	Description
1	Kx1	translational linear spring constant in X
2	Ky1	translational linear spring constant in Y
3	Kz1	translational linear spring constant in Z
4	Kx2	translational quadratic spring constant in X
5	Ky2	translational quadratic spring constant in Y
6	Kz2	translational quadratic spring constant in Z
7	Kx3	translational cubic spring constant in X
8	Ky3	translational cubic spring constant in Y
9	Kz3	translational cubic spring constant in Z

As an example element block,

```
Block 51
      Spring3
      Coordinate 7
      Kx1 1e6
      Ky1 1.11E7
      Kz1 0
      Kx2 0
      Ky2 0
      Kz2 0
      Kx3 1e4
      Ky3 1.11E5
      Kz3 0
End
```

3.26 Dashpot

A dashpot represents a damping term proportional to velocity. Dashpot elements combine a viscous friction damper with a simple linear spring. The spring is included to avoid singular stiffness matrices when dashpots are connected without springs. Dashpots are currently only used in transient dynamic, direct frf and complex eigen analyses. For other analyses only the spring term will be used.

The damping factor is the damping matrix entry. It has units of *force·time/length*. For a single degree of freedom system with a mass=M, the following equation is satisfied.

$$K \cdot u + c \cdot \dot{u} + M \cdot \ddot{u} = f(t) \tag{66}$$

Currently dashpots are defined in the basic coordinate system only. Because they are single degree of freedom elements, the direction must also be defined (i.e. cid=1, 2 or 3). There are three parameters. All are required.

#	Keyword	Description
1	K	translational linear spring constant
2	c	damping factor
3	cid	coordinate direction (1, 2 or 3)

As an example element block,

```
      Block 51
            Dashpot
            cid=1  // dashpot is in the X direction
```

```
        K=1e6
        c=1e5
End
```

Dashpots may be represented in the **Exodus** file with any linear element. The Truss element most closely mimics the dashpot's single degree of freedom behavior, and may be the best definition for domain decomposition tools.

Caution should be exercised when using dashpots (or any single degree of freedom element). The remaining degrees of freedom must be properly accounted for, or the system matrices will be singular. Care should also be exercised to ensure that if the nodes of the dashpot are not coincident, that the constraint force lies along the axis of the element - failure to do this can result in models that have nonzero rotational modes. There may also be important domain decomposition issues with dashpots. See section 3.23 for a discussion.

3.27 SpringDashpot

The **SpringDashpot** element provides a general, fully coupled spring and dashpot connected to a pair of nodes. It is a linear element only, and is not corotational. It supports stiffness and damping in the translational and/or rotational degrees of freedom. The relevant parameters are described in Table 88.

As shown in the table, all the elements of the matrices may be entered for this element. An example follows.

```
Block 100
    SpringDashpot
        Kxx = 1e4
        Kyy = 1e4
        Kzz = 1e4
        Kxy = -1e4
        Kyz = -1e4
        Byz = 3.2
END
```

3.28 Hys

The **Hys** element provides a simple, one dimensional approximation of a joint going through microslip. Many simple joints can be represented by their *hysteresis* loop, a curve in the displacement vs. force plane. The relevant parameters of this element are indicated in the table, and illustrated in Figure 67.

#	Name	Description
1	Kxx	Translation Stiffness, K_{xx}
2	Kyy	Translation Stiffness, K_{yy}
3	Kzz	Translation Stiffness, K_{zz}
4	Kxy	Translation Stiffness, K_{xy}
5	Kxz	Translation Stiffness, K_{xz}
6	Kyz	Translation Stiffness, K_{yz}
7	Krxx	Rotation Stiffness, Kr_{xx}
8	Kryy	Rotation Stiffness, Kr_{yy}
9	Krzz	Rotation Stiffness, Kr_{zz}
10	Krxy	Rotation Stiffness, Kr_{xy}
11	Krxz	Rotation Stiffness, Kr_{xz}
12	Kryz	Rotation Stiffness, Kr_{yz}
13	Bxx	Translation Damping
14	Byy	Translation Damping
15	Bzz	Translation Damping
16	Bxy	Translation Damping
17	Bxz	Translation Damping
18	Byz	Translation Damping
19	Brxx	Rotation Damping
20	Bryy	Rotation Damping
21	Brzz	Rotation Damping
22	Brxy	Rotation Damping
23	Brxz	Rotation Damping
24	Bryz	Rotation Damping
25	coordinate	coordinate frame

Table 88: SpringDashpot Parameters

#	Keyword	Description
1	Kmax	maximum slope of f vs u curve
2	Kmin	minimum slope of f vs u curve
3	fmax	maximum possible force
4	dmax	maximum possible displacement

The **fmax**, **dmax** pair define the limits of applicability of the element. The element will fail if the internal force exceeds **fmax** or the displacement exceeds **dmax**. The slope of the curve at the origin is **kmax**. It represents the small amplitude response of the system. The slope at the extremum, i.e. at (**dmax,kmax**) is **kmin**.

A **Hys** element uses a Beam or truss element in the **Exodus** file. At the current time, the element may only be defined in the X direction. An example of the **Sierra/SD** input is shown below.

```
BLOCK 2
     Hys
     Kmax 4.5e+7
     Kmin 3.0e6
     fmax 5.92
     dmax 0.9833e-6
END
```

3.29 Shys

A **Shys** is the whole joint model developed by Smallwood and is an element which uses a Beam or truss element in the exodus file. The element is a 2.5 dimensional element with an **Shys** element in both the X and Y directions and a linear spring element in the Z direction. The **Shys** element is assumed identical in both the X and Y directions in this formulation. A coordinate system can be defined to orient the element correctly.

This element is being phased out in favor of the Joint2g element, where similar constitutive behavior can be specified if desired.

An example of the Sierra/SD input is shown below.

#	Keyword	Description
1	n	Exponent describing slope of force-dissipation curve at very small amplitudes
2	k	Linear stiffness of Smallwood's element
3	kNL	Coefficient for non-linear stiffness
4	kz	Linear translational stiffness in the Z direction
5	k_r	Linear rotational stiffness (optional, default = 0)

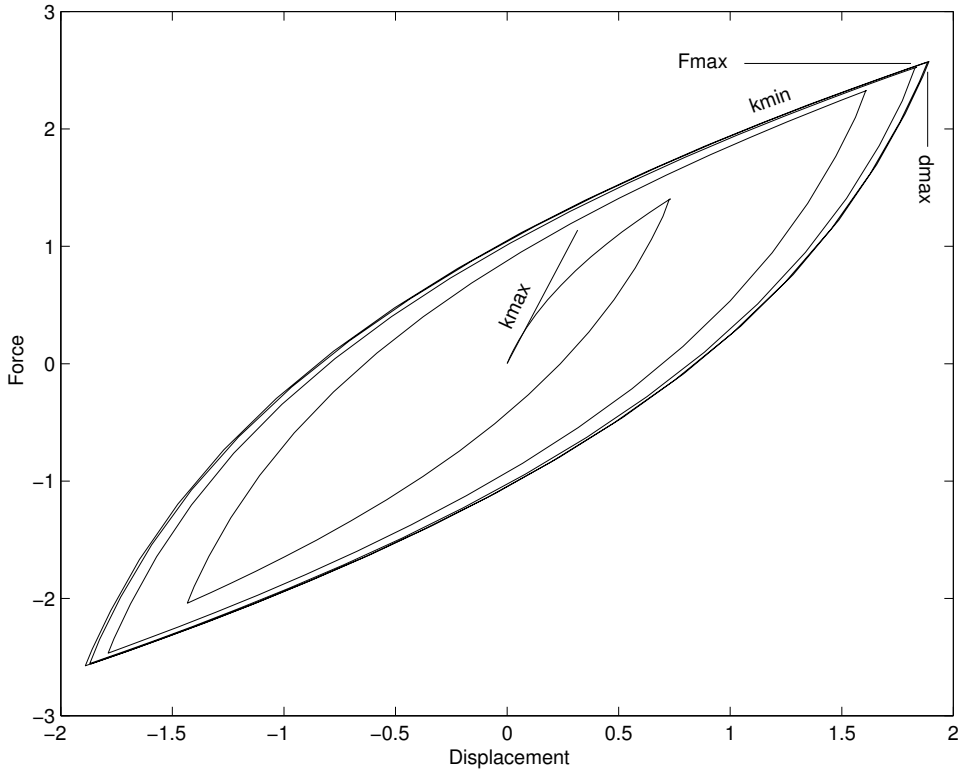


Figure 67: **Hys** element parameters

The **Shys** element does not use the attributes defined in the exodus file for default values of the optional parameters. A detailed discussion of the theory of the **Shys** element as well as how to determine the parameters can be found in the reports by Smallwood (Ref. 44).

```
BLOCK 2
     shys
     coordinate 2
     n = 1.39
     k = 1.3167e6
     kNL = 1.8499e6
     k_z = 1.6e6
     k_rot = 1.e9
END
```

3.30 Iwan

The Iwan model as a stand alone element has been phased out. Instead use the Joint2G element with an Iwan constitutive model.

3.31 Joint2G

The **Joint2G** element[2] was devised to facilitate the implementation of "whole joint" models in **Sierra/SD** . Beyond that it offers a workbench of considerable flexibility for specifying the nature of adherence between surfaces.

Each **Joint2G** element connects a pair of nodes (or *grids*, hence the "G" in *Joint2G*); it is a member of the geometrically one-dimensional class of elements *OneDim*. It's unique advantage is that it permits users to specify independently the constitutive behavior of each of the degrees of freedom connecting its node pair.

The constitutive behavior is implemented through a constitutive class that provides generalized scalar forces in response to corresponding generalized displacements. Though the class name is *Axial*, members of the class provide responses that do not make reference to the axial or rotational nature of the deformation.

The decoupling of the constitutive response from the element machinery facilitates creating additional constitutive classes without having to recreate the whole element machinery.

The **Macroblock** provides a complementary functionality which may be used to specify the mechanically parallel behavior through the use of multiple, co-located **Joint2G** elements. See section 2.26.

[2] Joint2G elements are supported and documented by Dan Segalman.

3.31.1 Specification

The meshed objects that map into the **Joint2G** element are defined in the **Exodus** database using BEAM or BAR elements. The Joint2G element does not make use of attributes defined in the **Exodus** file; all properties must be specified in the BLOCK and PROPERTY cards. In the example below, properties are assigned to element block "2".

```
BLOCK 2
  coordinate 5
  shear_axis 2
  joint2g
  kx=iwan    1
  ky=elastic 1.0e6
  kz=elastic 1.0e6
  krx=null
  kry=null
  krz=null
END
```

The above statement declares "BLOCK 2" to be of type **Joint2G**. It also declares the constitutive response in the "x" direction to be that of Segalman's 4-parameter Iwan model (SAND2002-3828). The parameters to be used in this model are those specified in "Property 1" defined below. In this case, the four parameters chosen are chi, phi_max, R, and S (χ, $\phi_{\max}$, R, and S in the SANDIA report). The Iwan properties can be specified alternatively by the parameter set chi, phi_max, F_S, and beta (χ, $\phi_{\max}$, F_S, and β).

```
property 1
    chi     =  -0.82139
    phi_max =   1.0325e-04
    R       =   7.608594e+06
    S       =   5.616950e+06
END
```

The constitutive behavior in the "y" and "z" directions is elastic with stiffness specified by the third argument - 1.0×10^6 in this case.

In this example, there is no specification for constitutive behavior in the three rotational directions. The *NULL* specification merely means that those degrees of freedom in the relevant nodes are are not activated ("touched") by this element. Because of artifacts associated with parallelization, it is recommended that if any of the rotational degrees of freedom are active (not *NULL*), they all should be active.

The directions ("x", "y", and "z") employed above are those associated with the coordinate system declared for the block. In the example shown, there is an explicit reference to

coordinate system 5. If there is no such explicit reference to a coordinate system, then the "x", "y", and "z" directions are those of the global coordinate system.

In the case when the joint2G element is used in conjunction with a tied joint, then the **shear_axis** can be used to specify the "x" direction for the constitutive response of the joint2G. Note that the **shear_axis** parameter is only meaningful when the joint2G is used in conjunction with a tied joint.

The **shear_axis** parameter allows the user to specify the "x" direction for the constitutive behavior. Since **shear_axis** is set to 2 in the above example, the "x" direction will be derived from the second component of coordinate 5. For more information on the **shear_axis** parameter, we refer to Figure 33 and section 2.24.

3.31.2 Constitutive Behavior

Elastic: Undamped, linear elastic behavior is defined by the keyword "elastic" followed by the value of the parameter. No "property" section is required.

Damper: Linear, damped behavior is obtained using a keyword "damper" in the joint2g definition, and using a property definition to specify the stiffness and damping terms. Typically each direction will require a different property definition.

```
BLOCK 3
    JOINT2G
    kx=damper 1
    ky=damper 2
    kz=damper 3
    krx=null
    kry=null
    krz=null
END

PROPERTY 1
    DAMPER
    K=1e6
    MU=.2
END
```

4-Parameter Iwan Model (iwan): The Iwan element is a collection of spring slider elements designed to provide a predicted model of joint behavior (including energy loss).

A detailed discussion of the theory of the Iwan element as well as how to determine the parameters can be found in the reports by Segalman (Ref. 45). Information about the Iwan element, and its relationship to other joint elements may be found in the Sandia internal report by Segalman and Starr (see 46).

The schematic of the Iwan model is shown in figure 68. Parameters for the behavior may be specified using either an older definition (Table 89), or a newer set (Table 90). The newer parameters are described briefly below, but the analyst is referred to the documentation for more detail.

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at very small amplitudes
2	R	Constant coefficient in distribution
3	phi_max	Maximum break free pseudo-force
4	S	Strength of singularity in break free force distribution
	alpha	Geometric factor specifying nonuniform spacing of dphi (optional, default = 1.2)
	sliders	Number of slider elements (optional, default = 50)

Table 89: Older Iwan 4-parameter model

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at very small amplitudes
2	beta	shape parameter of force/dissipation curve
3	K_T	Tangent stiffness at very low loads
4	FS	Maximum break free pseudo-force
	alpha	Geometric factor specifying nonuniform spacing of dphi (optional, default = 1.2)
	sliders	Number of slider elements (optional, default = 50)

Table 90: Revised Iwan 4-parameter model

chi: determines the slope of the dissipation-force curve. Typically $0 < \chi < -1$. A value of zero corresponds to a coulomb type loss in Mindlin solutions. A value of $\chi = -1$ corresponds to a viscous like (but amplitude dependent) loss with dissipation proportional to the square of the amplitude. Dissipation follows the relation,

$$\text{Dissipation} \approx (\text{Amplitude})^{\chi+3}$$

beta: determines the shape of the dissipation-force curve. Larger β (say 5), produces power law behavior over all amplitudes. Beta affects both the shape of the hysteresis curve within microslip (Figure 69), and the abruptness of the transition from microslip to macroslip as shown in Figure 70. $0 \leq \beta < \infty$.

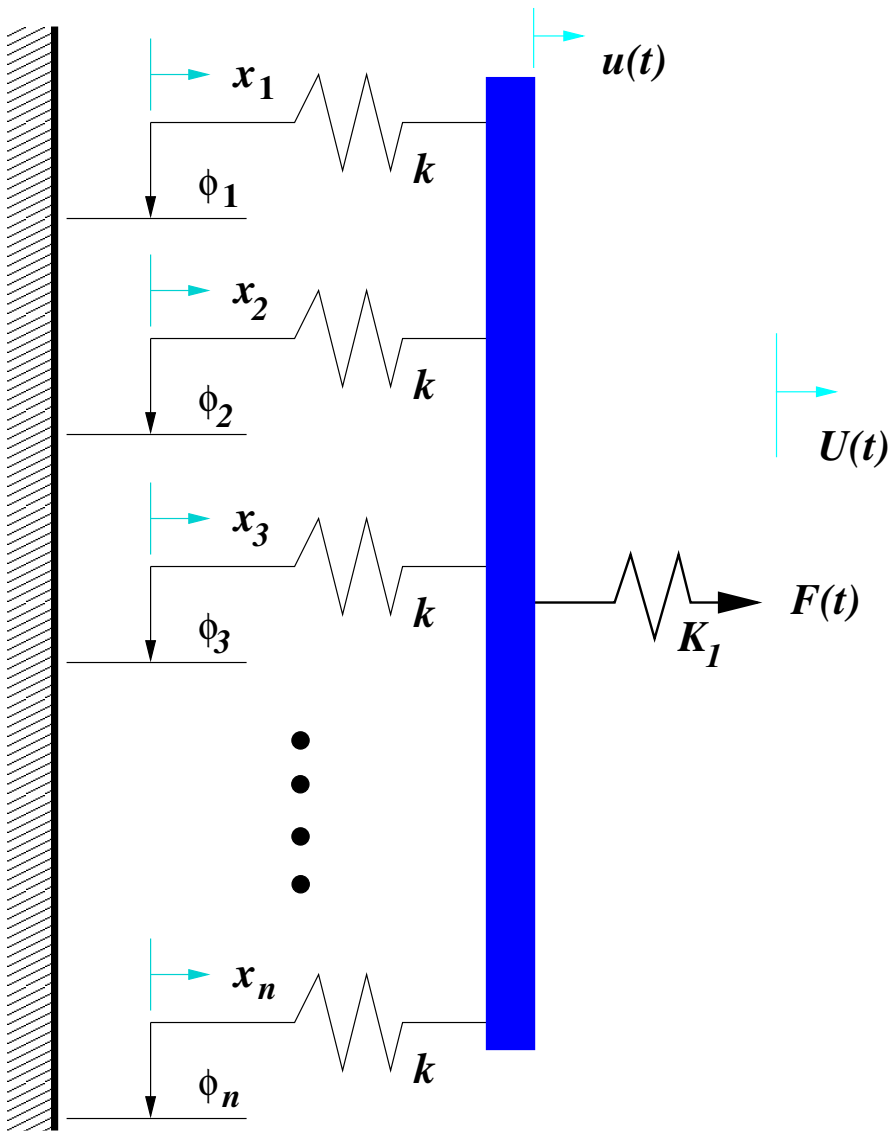


Figure 68: **Iwan** Constitutive Model

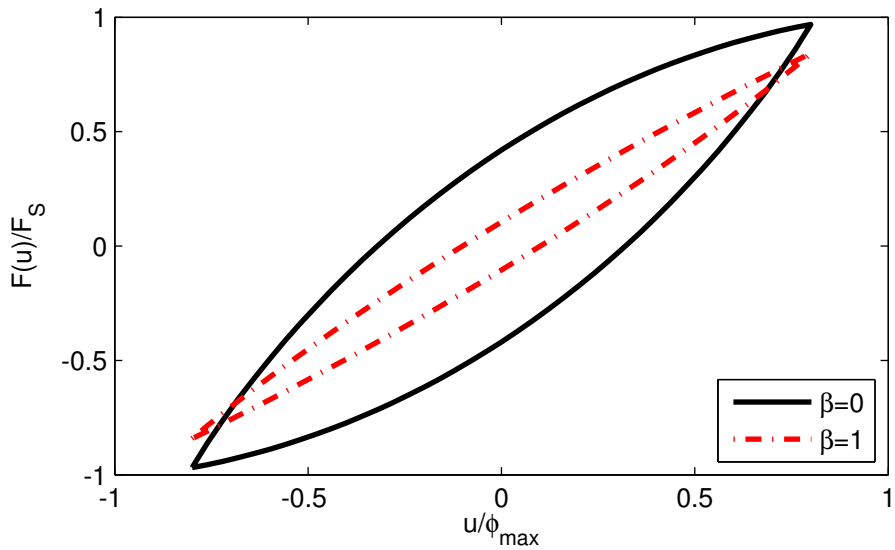


Figure 69: Dimensionless hysteresis curves for the four-parameter Iwan model with $\chi = -1/2$ and two values of β.

KT: determines the slope of the force-displacement curve at low amplitudes. This is equivalent to a spring constant, and is used as such in analyses for which the element is treated linearly.

FS: determines the force at which the last slider gives out, and element goes entirely into macroslip. The Iwan element is a statistical distribution of spring/slider elements. This is a point on that distribution.

Smallwood's Hysteresis Model (shys): D.O. Smallwood developed a three parameter model that captures the power-law behavior of energy loss with force amplitude. The model parameterizes the hysteresis loop determined from experimental data in such a way that the power law behavior is preserved.

#	Keyword	Description
1	n	Exponent describing the slope of the force dissipation curve at small amplitudes
2	k	Coefficient for the linear stiffness
3	knl	Coefficient for the non-linear stiffness

A detailed discussion of the theory of the **shys** model as well as how to determine the parameters can be found in reference 47.

PROPERTY 1

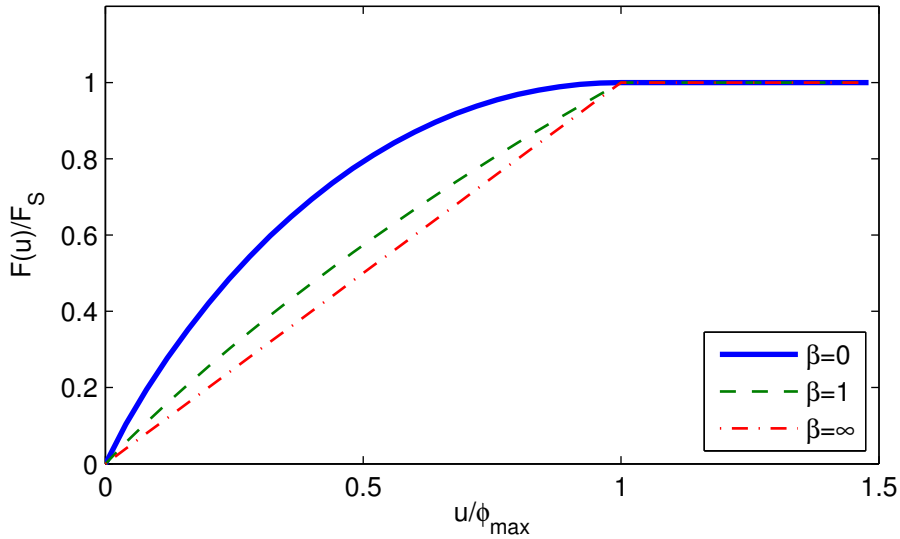


Figure 70: Dimensionless static loading curves for the four-parameter Iwan model with $\chi = -1/2$ and three values of β, as the model goes into macroslip.

```
    n   =  1.39
    k   =  1.3167e6
    knl =  1.8499e6
END
```

One Dimensional Gap Model (gap): The **gap** model attempts to represent the behavior of a gap closure with a bilinear elastic element. For proper numerical behavior, the stiffness of the open gap should not be more than a few orders of magnitude less than the stiffness when the gap is closed. The **Joint2G** implementation of the **gap** model is identical to the axial behavior of NASTRANS cgap/pgap element as well as the axial behavior of the stand alone version of the gap element implemented in **Sierra/SD** (section 3.32).

#	Keyword	Description
1	Ku	Unloaded Stiffness
2	Kl	Loaded Stiffness
3	U0	Initial Gap Opening
4	F0	Preload (force at U0)

```
PROPERTY 1
    ku  =  1e5
    kl  =  1e6
    U0  =  0.01
    F0  =  200
```

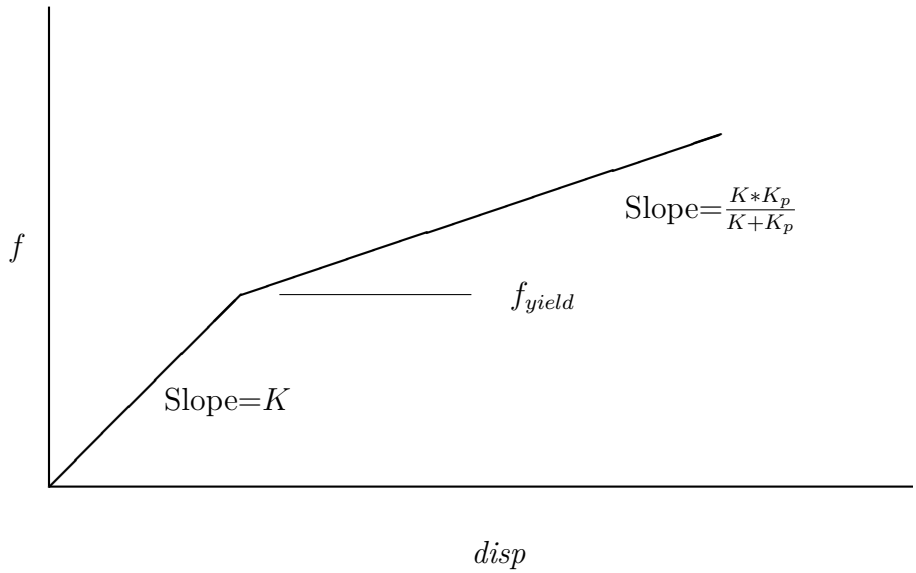


Figure 71: Eplas Model

```
END
```

Elastic Plastic Hardening Model (eplas): The **eplas** element is an elastic-plastic 1-dimensional element with linear isotropic hardening. Both the plastic strain and the hardening variable are initialized to zero. The parameters are illustrated in Figure 71.

#	Keyword	Description
1	k	Linear Stiffness
2	kp	Hardening Stiffness
3	fyield	Force at Yield

```
PROPERTY 1
    k       =  1e6
    kp      =  1e5
    fyield  =  1e4
END
```

One Dimensional Spring-Dashpot Model (damper): A **damper** represents a damping term proportional to velocity. Damper elements combine a viscous friction damper with a linear or cubic spring. The spring is included to avoid singular stiffness matrices when dampers are connected without springs. Dampers are currently only used in transient dynamic, direct frf and complex eigen analyses. For other analyses only the spring terms will be used. The behavior of this element is similar to **dashpot**, but also includes cubic terms.

The damping factor is the damping matrix entry. It has units of *force·time/length*. For a single degree of freedom system with a mass=M, the following equation is satisfied.

$$K \cdot u + \mu \cdot \dot{u} + M \cdot \ddot{u} + K_3 \cdot u^3 + \mu_3 \cdot \dot{u}^3 = f(t) \tag{67}$$

#	Keyword	Description
1	K	Stiffness
2	Mu	Viscous Damper Coefficient
3	K3	*Optional* cubic stiffness coefficient
4	Mu3	*Optional* cubic damping coefficient

```
PROPERTY 1
    K   =  1e6
    Mu  =  1e2
    K3  =  1e4   // optional, default=0
    Mu3 =  0.1   // optional, default=0
END
```

Additional Constitutive Behavior: The philosophy employed in the implementation of the **Joint2G** element of decoupling the constitutive behavior from the element machinery should facilitate the implementation of other constitutive models. Among those whose implementation is foreseen are the following:

- Bouc-Wen hysteresis model
- Preisach hysteresis model

3.32 Gap

Gap elements are modeled after the non-adaptive nastran `CGAP/PGAP` elements. They are intended to provide a simple, penalty type element suitable for modeling simple connections. Note that these elements (like all beam-like elements) when embedded in solid meshes can result in difficult domain decompositions, and lead to load imbalance.

The **Gap** element is inherently nonlinear. In linear analysis, the element behaves approximately like a spring with the stiffness determined by `KL` and `KU` and a transverse stiffness, KT. The parameters of the element are listed in the table below and shown graphically in Figure 72.

#	Keyword	Description
1	KU	unloaded stiffness
2	KL	loaded stiffness
3	KT	transverse stiffness (closed)
4	U0	initial gap opening
5	F0	Preload, i.e. force at U0
6	coordinate	Required coordinate frame.

The unloaded stiffness, KU, represents the stiffness of the element when the gap is open. It must be greater than zero. The loaded stiffness, KL, represents the stiffness when the gap is closed (as shown in the figure). The stiffness is KL when UA - UB is greater than U0.

The initial gap opening and preload define the corner point in the force/deflection curve as shown in Figure 72. Typically these will be zero.

A gap element provides for transverse stiffness and friction. When the gap is closed, the transverse stiffness is KT. If the gap is open, the transverse stiffness is reduced to $KT' = KT \times KU/KL$.

The coordinate frame is an optional attribute of the gap element. The gap open and closes along the X axis of the frame. Note that the direction of the coordinate frame is quite important. The element determines a quantity $UA - UB$ along this coordinate axis. *This axis may not align with the coordinate alignment of the elements, which can lead to confusion.* If the coordinate frame is not provided, each gap element will have a coordinate frame generated such that the gap opens and closes along the line between the two points. If the points are coincident, then a coordinate frame *must* be provided.

The gap element is a simple penalty type element that somewhat mimics the effect of a physical gap. Choice of the value of KL is very important to success of the element. Good values are somewhat in the range of the neighboring element stiffness. Too large a value can lead to matrix condition problems. Too small a value results in excessive softness and penetration in the gap.

Because the element is nonlinear, it has a significant impact on solutions. As described in section 2.1.27 (and the **update_tangent** keyword), the default behavior for the nonlinear solver is a partial Newton iteration. This means that the tangent stiffness matrix is not updated between iterations. Thus, if KL and KU are quite different, the solver will be using the wrong slope in the newton loop. Many, many iterations may be required for convergence. You may want to turn on the 'nlresidual' option in the echo section (see 2.8) which will put convergence information into the results file.

An example is shown below.

```
BLOCK 2
     GAP
     KL 4.5e+7
```

```
      KU 3.0e6
      KT=1e6
      f0 5.92
      u0=0.9833e-6
      coordinate 5
END
```

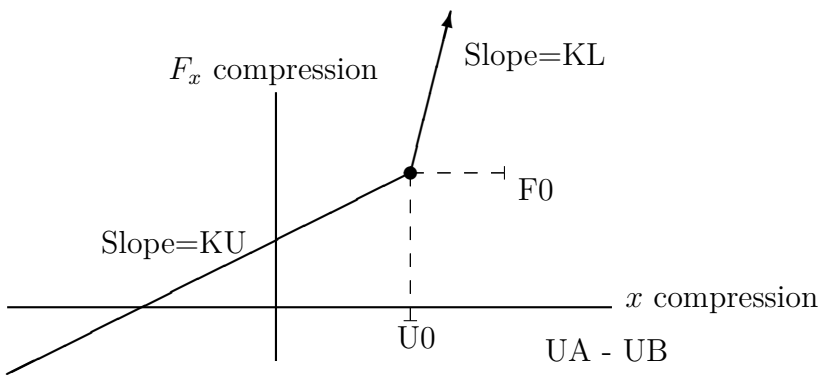


Figure 72: Gap element Force-Deflection Curve

Gap Issues. The gap element is definitely more complex than most elastic elements. Here is a partial list of "gotchas" that we have observed.

- Gaps should normally be zero length elements. Like springs, a gap that has a physical length will not be invariant to rigid body rotation. See section 6.3.5. One approach to this would be to use a combination of beam and gap elements. Note however, that if `KT` is zero, and the gap opens and closes along the line between the beam endpoints, the element is invariant to rotation.

- The gap element may use a coordinate frame to define its direction. *In this case the direction is NOT set by the nodal coordinates.*

- The direction of the gap element must correlate to the displacement difference from $UB - UA$. It is very easy to get this direction reversed.

- If you set **U0**, you must also set **F0**. This element does not constrain the force/displacement curve to go through zero. The input must do this. The gap element may thus be used to enforce an initial displacement or force. That may not be what you want. It can cause very slow convergence on the initial time step.

- Significant numerical damping may be required for convergence. Closing the gap can cause energy to be moved into higher frequencies. Without numerical damping, this energy can multiply until the solution becomes unstable. Numerical damping is best

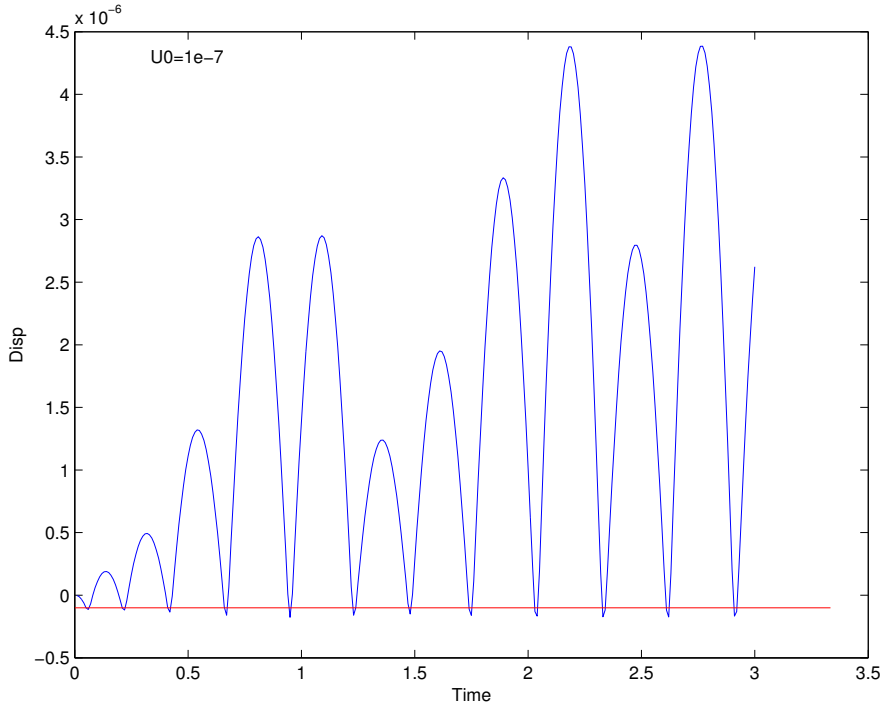


Figure 73: Mass bouncing off a Gap. With this large time step the model is not conserving energy. Reducing the time step is required to correct the problem.

introduced by setting "`rho`" in the time integrator. Values of "`rho=0.2`" to "`rho=0.7`" have worked well. It is problem dependent.

Physically closing a gap would cause some energy loss, either by microslip, or by a small amount of local plastic deformation. Numerical damping can dissipate this energy that is removed from the physical system by means that are not included in the finite element model.

- This gap element may not conserve energy. This is demonstrated in Figure 73, where a mass is dropped onto a gap. A completely elastic rebound would take the mass back to zero. Instead, it rebounds significantly above zero. This issue comes about because of time discretization. The mass "penetrates" the gap region too far, which stores too much energy in the element. It is then expelled with too much velocity. The only solution with this element is to reduce the integration step.

- Setting either `KU` or `KL` to zero is a recipe for disaster in parallel. Use a small positive value even if physically the unloaded stiffness may be zero.

3.33 Gap2D

The **Gap** element of the previous section provides a useful construct for planar type interactions. A common modeling issue is a bolt in an oversized hole. To model this interaction an ellipsoidal gap element (or **Gap2D**) may be required.

The **Gap2D** element operates just like the Gap element except that the gap could open in 2 dimensions. The gap is open provided that the element displacement is within an ellipse defined by the major and minor axes.

$$\left(\frac{u_x}{\mathrm{U0X}}\right)^2 + \left(\frac{u_y}{\mathrm{U0Y}}\right)^2 < 1 \tag{68}$$

The major and minor axes of the ellipse are defined in the x and y direction of the *required* coordinate frame.

Parameters of the **Gap2D** element are listed below.

#	Keyword	Description
1	KU	unloaded stiffness
2	KL	loaded stiffness
3	KT	transverse stiffness (z direction)
4	U0X	initial gap opening, major direction
5	U0Y	initial gap opening, minor direction
6	coordinate	Required coordinate frame.

While the gap geometry is defined as an ellipse, the stiffnesses are not. In the open section of the element, the in-plane stiffness is `KU`, and is independent of direction. Likewise, in the closed gap region, the in-plane stiffness is independent of direction, and is defined by `KL`. The out of plane stiffness for this element is *always* `KT`. Note that the transverse stiffness behavior is significantly different than that of the standard **Gap** element.

The definitions above define the gradient of the force only, and for this nonlinear force, the value of the force depends on the path chosen for integration. For this element, we define the force as the integral along the shortest line from the origin.

In Figure 74, two possible integration paths are shown for arriving at the point (x_1, y_1). In the first path, we integrate to $(x_1, 0)$ and then up to (x_1, y_1). The y component of force is $f_y^{(1)} = \mathtt{KL} \cdot y_1$. In path 2, we follow the straight line through (x_b, y_b). The associated force is $f_y^{(2)} = \mathtt{KU} \cdot y_b + \mathtt{KL}(y_1 - y_b)$. For this element, we always choose the shortest line path (path 2). This ensures that the force is not history dependent.

3.34 GasDmp

The **GasDmp** element is a nonlinear, beam-like element that simulates the damping forces on MEMS devices due to gas pressure as MEMS beams vibrate. The element has no

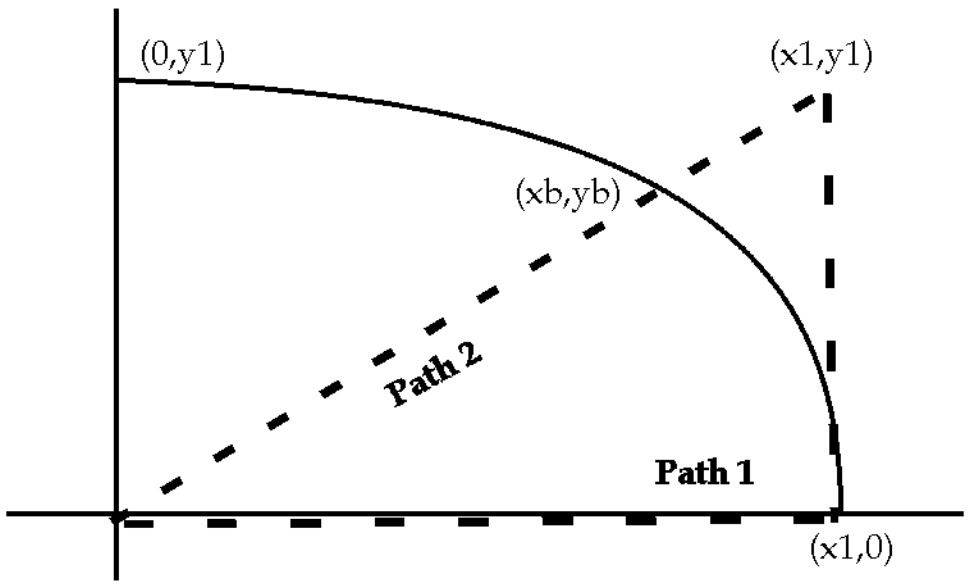


Figure 74: Gap2D force diagram

stiffness, but has damping roughly proportional to velocity/L^3, where L is the distance from the beam to the substrate. The element is very experimental, and still under development. Contact Troy Skousen or Burak Ozdoganlar at 845-0427 for details.

Inputs to the **GasDmp** element are as follows.

#	Keyword	Description
1	W	Beam width (length units)
2	dL	Considered length of beam (length)
3	mm	Molecular mass of gas (mass)
4	p0	Ambient pressure of gas (pressure)
5	T	Ambient temperature of gas (temperature)
6	muRef	Reference viscosity (pressure * time)
7	TRef	Reference temperature (temperature)
8	ww	Viscous temperature exponent

Currently all of the parameters are implemented through the input file and not through the Exodus_II file. At a future date the beam width and length will be tied to the mesh.

The theory for the development can be found in an internal Sandia draft report available on the Sandia internal web at,
http://www.jal.sandia.gov/Salinas/external-reports/microbeam2.pdf
Most of the implementation is associated with equations 9 and 10 of this report.

3.35 Nmount

The **Nmount** element is a Navy-specific mount element that provides an external force at user-specified points in the model. These forces are formed from a constitutive equation that is supplied by the user in the form of a subroutine. The Nmount capability provides an interface capability that allows the user to input their own subroutine that evaluates the constitutive equation.

An example of the user interface is shown in Figure 75. Mount orthogonal directions must be provided either as attributes in the **Exodus** file, or using the "Orientation" keyword in the "Block" section. The relation between the orientation vector and internal element coordinates is shown in Figure 76. Remaining information is provided in the "Block" section. Each mount type requires a separate block entry. Mount parameters are provided as text input in the Block section.

Each mount type may require a different number of mount parameters. If more parameters are provided than required for this mount, the additional parameters are ignored *without warning*. If less parameters are provided than are anticipated for the mount, the last parameters are set to zero, a warning is printed, and the analysis continues.

```
BLOCK 41378
      NMOUNT
      MOUNT TYPE = 99
            Parameters = 1.414 3.141 2.713
            Orientation = 0 0.7 1
END
```

Figure 75: **Sierra/SD** Mount Interface

Stability: The Nmount element applies a force to the joining nodes in much the same way as an externally applied force. It provides no contribution to the stiffness matrix, and as such resembles an explicit element. Thus, stability issues can arise with this formulation. For certain models, damping has been shown to stabilize the formulation. The user may need to experiment with time step and damping levels to determine appropriate parameters for a stable solution.

3.36 MPC

Multi-Point Constraints (or **MPC**s) are constraint equations applied directly to the stiffness matrix. They are not elements, and are not available from an **Exodus** database. However, in many respects they look like elements, and can be thought of as elements. Some analysis codes treat them as pseudo elements.

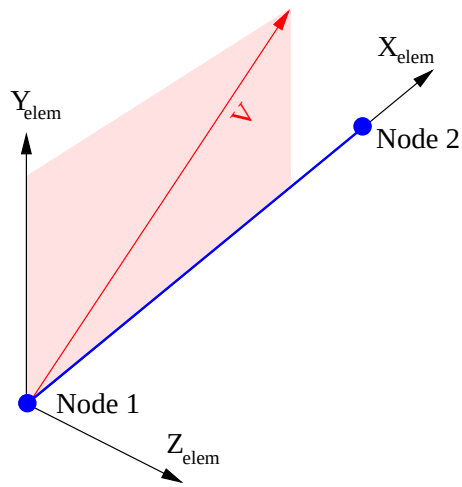


Figure 76: Nmount Orientation

X_{elem} Normalized vector from node 1 to node 2. May change as the structure deforms.

$\vec{V}$ User provided orientation vector.

$Z_{elem} = \frac{\hat{X}_{elem} \times \vec{V}}{|\hat{X}_{elem} \times \vec{V}|}$

$Y_{elem} = \hat{Z}_{elem} \times \hat{X}_{elem}$. A normalized vector in the $X_{elem}\vec{V}$ plane, and orthogonal to X_{elem}.

All **MPC**s describe constraint equations of the form,

$$\sum_i C_i u_i = 0$$

where C_i is a real coefficient, and u_i represents the displacement of degree of freedom i.

Unlike many Finite Element programs, **Sierra/SD** does not support user specification of constraint and residual degrees of freedom (DOF). In serial solvers the partition of constrained and retained degrees of freedom is performed simultaneously by Gauss elimination with full pivoting so the constrained degrees of freedom are guaranteed to be independent. In parallel solvers (such as FETI), the constraints are specified as Lagrange multipliers which involve no such partitioning. Redundant specification of constraint equations is handled by elimination of the redundant equations and issue of a warning. User selection of constrained DOF in Nastran has led to significant headaches for analysts who must ensure that the constrained DOF are independent and never specified more than once.

Each **MPC** is specified in the input file with a section descriptor. Note that a separate section is required for each equation (or degree of freedom eliminated). An optional coordinate system may be specified on the input, but must be the first entry in the section[3]. The **MPC** will be stored internally in the basic coordinate system (coordinate frame 0). The input consists of a triplet listing the global ID of the node, a degree of freedom string, and the coefficient of that degree of freedom. The degree of free strings are x, y, z, Rx, Ry, Rz. They are case insensitive. If the global ID of the node in the MPC does not exist in the model, the code will exit with a fatal error.

#	Keyword	Description
1	coordinate	optional coordinate frame with *integer* id
2	*integer*	*integer* node number in global model (The node number MUST USE 1 TO N ORDERING like **Exodus** file numbering).
3	dof string	*string* x, y, z, Rx, Ry, or Rz
4	coefficients	*Real* weight associated with this dof
items 2-4 may be repeated as many times as needed		

In the following example, the x and y degrees of freedom in coordinate system 1 are constrained to be equal for node 4.

```
MPC
    coordinate 1
    4 x 1.0
    4 y -1.0
END
```

[3]At this time, all the nodes in an MPC must be associated with the *same* coordinate system.

IMPORTANT

Constraints are handled in various ways by the linear solvers. In the serial solver, the dependent degrees of freedom are eliminated before the matrices are passed to the solver. In parallel, we use Lagrange multipliers to handle the constraints. There is currently no user control of constraint handling methods.
Note also that there are practical differences between rigid elements (described in the following sections) and constraint equations that are nominally identical. For parallel solutions, we are currently using an augmented Lagrange type solution method with the rigid links. This means that terms are added to the stiffness matrix in parallel with the constraints. In most cases, this renders the matrices positive definite, and greatly increases robustness and solution performance with no penalty for accuracy. Thus, rigid links are recommended whenever possible in parallel solutions.
Finally note that replacing rigid links with very stiff beams can be a bad thing to do. The condition of the resulting matrices can be severely degraded which can lead to significant loss of accuracy.

3.37 RROD

An **RROD** is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for an **RROD** may be conveniently stated that the dot product of the translation and the beam axial direction for a **RROD** is zero. There is one constraint equation per **RROD**.

The **RROD** is specified using beams or trusses in the **Exodus** database, with a corresponding **Block** section in the **Sierra/SD** text input file. No material is required and any number of connected or disconnected **RROD**s may be placed in a block. The following is an example of the input file specification for **RROD**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RROD
END
```

Attribute	default	description
RB_ID	-	translation identifier
CID_FLAG_INDEP	123456	independent coordinate flag
CID_FLAG_DEPEND	123456	dependent coordinate flag

Table 91: Rbar Exodus Attributes

3.38 RBar

An **RBAR** is a *pseudo*element which is infinitely stiff in extension, bending and torsion. The constraints for an **RBAR** may be summarized as follows.

1. the rotations at either end of the **RBAR** are identical,
2. there is no extension of the bar, and
3. translations at one end of the bar are consistent with rotations.

The **RBAR** is specified using beams or trusses in the **Exodus** database, with a corresponding Block section in the input file. No material is required and any number of connected or disconnected **RBAR**s may be placed in a block. The following is an example of the input file specification for **RBAR**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBAR
END
```

RBARs can be reordered so that the number of **RBAR**s connected to a single node is minimized. Having a large number connected to the same node results in a highly populated matrix and a slow computation. Therefore, reducing the number of connections can shorten run time. (see the *reorder_rbar* parameter in the **PARAMETERS** section 2.3).

The **RBAR** attributes are listed in Table 91, and are described below.

RB_ID Sometimes a collection of RBARS is a description of a rigid body. This occurs for example when translating a nastran model containing RBE2 elements. During translation these bars are grouped into rigid bodies based on their connectivity. The RB_ID is an index to that grouping.

CID_FLAG_INDEP By default, all degrees of freedom are active on both nodes of the RBAR. Independent dofs are activated on the first node. The "CID_FLAG_INDEP"

allows control over which degrees of freedom are activated. The flag is specified as an integer which is sum of components.[4]

100000	X degree of freedom
20000	Y degree of freedom
3000	Z degree of freedom
400	R_x degree of freedom
50	R_y degree of freedom
6	R_z degree of freedom

Thus, '123456' activates all dofs, and '123000' activates only translations.

CID_FLAG_DEPEND By default six dofs are eliminated from the bar. By setting this attribute to a non-default value, constraint equations may be skipped. The values are the same as the "CID_FLAG_INDEP" described above. If all 6 dofs are included in the constraint then a link stiffness will (optionally) be applied. If any dofs are not constrained, then the link stiffness is not used. See section 2.3.

3.39 RBE2

Sierra/SD has no support for the Nastran **RBE2** element. However, in most cases there is little difference between the **RBE2** element and a collection of **RBAR**s.

3.40 RBE3

The **RBE3** pseudo-element's behavior is taken from Nastran's element of the same name. Two distinct versions of the element are available, but the older version will be deprecated sometime in the future. Each method is each described below, with significantly more detail found in section 3.21.3 of the theory manual . The element is used to apply distributed forces to many nodes while not stiffening the structure as an **RBAR** would. The **RBE3** uses the concept of a slave node.

Because all the nodes in an **RBE3** are not equivalent, each **RBE3** requires its own block ID. In the **Exodus** file, all links connecting to a single **RBE3** are defined in a single element block. The input file then specifies that this is an **RBE3** element block, as shown in the example below. If the model requires many **RBE3**s, a separate block must be specified for each.

Usage. The optional parameters for the Rbe3 pseudo-element are shown in the table below. These parameters must be specified in the input file, not as attributes of the **Exodus** file.

[4] It is a rather unusual descriptor, but it was designed to somewhat mimic the nastran cid flag.

Keyword	**value**	Description
refc	*string*	reference coordinates for slave
method	*new* or *old*	Constraint computation method
WT	6 *reals*	relative weight of coordinates

refc. The **REFC** parameter sets the degrees of freedom to activate on the slave node. The keyword **REFC** provides a text representation of the active degrees of freedom involved in the constraints. Thus, `REFC='12'` provides 2 equations that constrain degrees of freedom associated with X and Y translations. No other degrees of freedom are affected. If the **REFC** keyword is not provided, it defaults to `REFC='123456'`, i.e. constraint relations will be provided for all 6 structural degrees of freedom on the slave node.

method. This parameter determines which formulation is used to determine the constraint relations. By default, the *new* method is used in versions of **Sierra/SD** newer than 2.0. See below.

WT. The contributions of each of the coordinates of the independent nodes may be scaled by **WT**. Most typically this would be used to determine the relative weight of rotational degrees of freedom on the independent nodes to the computation of the slave node rotations. The default value is **WT**= 1 1 1 0 0 0 which means that the rotations do not contribute to the Rbe3.

Generally we recommend there be no contribution from the rotations. The rotation of the element may then be determined solely from the translational degrees of freedom on the independent nodes.

The parameter applies only to the new method. In the old method rotations on the independent nodes are always ignored.

New Method Rbe3. The new formulation of the Rbe3 is based directly on the published method from MSC nastran. Details of the method are described in section 3.21.3 of the *theory manual*.

Old Method Rbe3. Previous to version 2.0, a version of the RBE3 was generated based on an *ad hoc* mathematical approach. This element should act like a Nastran **RBE3** for most applications, but its use is discouraged.[5]

[5] These elements are not identical. In particular, RBE3 elements that have 2 or fewer dependent nodes, or for which the dependent nodes are colinear will either not work, or not work as anticipated. As outlined in the theory manual, the rotational degrees of freedom on the independent nodes are ignored. Further, the old formulation will differ from the new approach if the slave node is far from the centroid of the element.

Cautions in using RBE3. While a very convenient construct, the **RBE3** is not a true element, and it can introduce complexity in the solution. Following are a few things to bear in mind in using the element.

- Very large RBE3 elements may spread across a large portion of the model. This affects linear solvers that are typically designed to propagate error locally. As a consequence convergence may be slow.
- Large RBE3 elements may require a lot of memory. This memory is stored on a single processor.
- No MPC should be linked to another. Many of our solvers will fail if one MPC type element shares nodes with another.
- Prescribed accelerations (see section 2.14.2) cannot be applied on an **RBE3** or any other MPC.
- The element has no logic to determine which degrees of freedom of the independent nodes are active. Thus, if you specify $WT = 1\,1\,1\,1\,1\,1$ the element will try to determine it's rotation based on a combination of the translational and rotational degrees of freedom on the independent nodes. If the rotational degrees of freedom are inactive, they are treated as zero. This is rarely what is wanted.
- Care must be taken to ensure that only one node of the **RBE3** has multiple connections to its links. Further, all links in the **RBE3** must be connected to the slave node.
- We note that many of our trouble tickets come from Rbe3 elements.

Example Rbe3. The following is an example of the input file specification for an **RBE3** if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBE3
    refc=123456
    method=new
    wt=1 1 1 0 0 0
END
```

3.41 Superelement

Superelements have various meanings in commercial codes. **Sierra/SD** does not support a full automatic superelement capability. In section 2.1.5 the procedure for reducing an

entire model to a reduced order model is outlined. Import of a such a reduced model (or superelement) into **Sierra/SD** is also supported. The **superelement** described in this model involves import of a mass and stiffness matrix into a full system model. This linearized approach complements a Craig-Bampton (and other) reductions, and may be used in any type of analysis.

Limitations

- The superelement must be small enough to fit on a single processor in a parallel run. No consideration for superelements which span processors is made.
- Nodes on the superelement interface may be shared across processors. Interior degrees of freedom are local to a single processor.
- Output of the interface node degrees of freedom will be made in the base model in the usual way. Output of internal superelement quantities will be made in the superelement database file.
- No automatic data recovery is available.
- Only a single level of superelement is supported.
- The mass properties report is computed by lumping mass to the interface dofs.
- No geometric stiffness effects are currently accounted for in superelements. The default at this time is for these elements to return zero geometric stiffness.

User Input

Each superelement must be placed a unique block, i.e. there is one superelement per block. The following input is provided by the user.

connectivity: To provide the geometric connectivity to the model, the connectivity must be added to the **Exodus** file. If the superelement has the same number of nodes as a standard element, the analyst may choose to use such an element to provide the connectivity. This can facilitate visualization of the model. When the model is larger, a tool is provided to directly add the superelement to the **Exodus** database.[6]

Note that codes such as nastran input superelements by connecting to the nodes directly. In a parallel environment, it is critical that the superelement remain on a processor. As a consequence, the decomposition tool must have knowledge of the superelement. It must therefore be in the finite element database. This is also consistent

[6]The tool is named "**mksuper**". It is part of our standard tools distribution.

with the other tools used with **Sierra/SD** where node numbers are not typically provided directly. This permits insertion of the superelement in a part, with a subsequent node reordering from `gjoin` for example.

Sierra/SD does support an element with *more* nodes than required for the connectivity map. Thus, a Hex-8 could be used to define the connectivity for a superelement with 7 nodes on the interface. Obviously the connectivity map cannot have more nodes than the element.

connectivity map: The equations for the system matrices must be associated with the nodes and degrees of freedom in the model. The following example creates a map for an eight degree of freedom reduced order matrix. The first column of the map is associated with the node index in the element. The second degree of freedom defines the coordinate direction (typically 1 to 6 for x, y, etc).

```
// node  cid
 map 0    0
     0    0
     1    1
     1    2
     1    3
     2    1
     2    2
     2    3
```

In this example, the first two rows of the system matrices are associated with internal degrees of freedom. These interior dofs are indicated by a zero for both the node index, and the coordinate direction. Row 3 of the matrix is associated with the first node in the element connectivity, and with the x coordinate direction. Row 8 is associated with the second node, and the z coordinate direction.

There must be exactly as many rows in the connectivity map as there are rows in the system mass and stiffness matrices.

If the node index is less than zero, the row of the matrix associated with that degree of freedom will not be mapped to the system matrix. This can be used to "clamp" a generalized degree of freedom.

The node index is NOT the node number in the **Exodus** *file. Rather it is the index into the element connectivity. Thus, for a four node element, the index must never exceed 4. This permits the use of* `gjoin` *and other tools without the need to reorder these terms in the input file.*

Alternate formats may be used to provide the map between rows of the system matrices and degrees of freedom of the residual structure. For these alternate formats to be used, the netcdf file containing the superelement data must include the `cbmap` data, which

provides an *internal* mapping between internal rows and columns and the internal nodes. These methods include the following.

map ascending_id or sorted If the user specifies the node number connectivity of the superelement in an ascending node order, then we can automatically generate the map.[2] Note, either *ascending_id* or *sorted* may be used here, they refer to identical algorithms.

map locations If the nodal coordinates of the superelement are stored in the netcdf reduced order model file, then the best match between coordinates of the residual and the superelement can be used to determine the map. This method works best if the superelement and residual have the same coordinate locations and if there are no collocated nodes in the interface.

system matrices: The system matrices must be provided in a **netcdf** file. These matrices are available as output of the CBR reduction process (section 2.1.5) and may also be generated with other tools such as nasgen. The file must contain the following.

Kr. The reduced stiffness matrix. This is required for all analysis.

Mr. Most analyses require a reduced mass matrix as well. It's dimension must match that of the stiffness matrix.

Cr. A reduced damping matrix may be used for some analyses. It is entirely optional, but if present, must be of the same dimension as **Kr**.

maps that connect the degrees of freedom of the superelement to the degrees of freedom of the residual structure.

A good reduced **Kr** for 3D analysis should have exactly 6 zero energy modes. It must be symmetric (**Sierra/SD** will try to symmetrize it). Typically **Mr** would be nonsingular. Failure to meet these requirements can confuse the entire solution procedure, and lead to erroneous solutions.

transfer matrices: Output of results on interior points in the superelement are facilitated using optional output transfer matrices (OTM). These are described in some detail in the section on model reduction (2.1.5). These matrices are used *only* if superelement output is requested in the output specification. The following matrices apply.

OTM Nodal output transfer matrix.

OTME Element output transfer matrix.

OutMap An optional node map for the OTM.

OutElemMap An optional element number map for OTME.

[2] With the "mksuper" application, it is easy for the user to set up an element with ascending order, but most tools do not know how to visualize the element. Visualization may be easier using standard elements, but the the restriction that the connectivity have ascending node ids is confusing.

skip_output: Optionally provides a means of disabling all output to the netcdf results files. This is particularly useful if the analyst wishes to use the same netcdf data for multiple superelements in the model. Without this keyword, each of these superelement blocks would be writing to the same file location, resulting in corrupted data.

output specifications: Output from superelements may be requested in the "ECHO" or the "OUTPUT" sections. If requested in the "OUTPUT" section, then a new **Exodus** file will be generated based on the information and name of the netcdf file. The number of nodes in the new file is the sum of the number of nodes on the interface and the number of nodes in the output transfer matrix, OTM. The number of elements is the number of elements in the OTME. All elements will be placed in a single element block. For either the echo or the output sections, output of superelement data is specified by the **superelement** keyword.

Because we don't know the connectivity of the elements in the OTME, all such elements will be defined as sphere elements, and will be collocated on a single node in the model. This makes visualization pretty much useless, but the element data is preserved for other types of post processing.

Likewise, no coordinate information is available for the interior nodes of the model. These elements will be located at the origin of the system.

sensitivity_param: When the **cbr** analysis that generated the superelement included a sensitivity analysis, the netcdf file containing the superelement matrices also contains derivatives of the reduced matrices with respect to the parameters. This information can then be used in the superelement block to set the superelement parameters to whatever values are desired. This uses the linear Taylor series expansion of the sensitivity information of the Craig-Bampton model to compute the updated reduced matrices, and thus by-passes the need to go back and re-generate the Craig-Bampton model when the parameters are perturbed. The **sensitivity_param** allows the user to input specific values of the parameters for the superelement.

Parameters

The parameters for the superelement block are listed in the table.

Keyword	**value**	Description
file	*string*	netcdf file containing matrices
savememory	*yes* or *no*	controls storage of matrices in memory
diagnostic	int	0 = run no diagnostics 1 = compute Kr * RBM 2 = compute eig(Kr,Mr)
map	*ints* *string*	table of node/cid pairs, OR "ascending_id" or "sorted" or "locations"
skip_output	N/A	optionally disable netcdf output
sensitivity_param	*Int Real*	parameter index and value of sensitivity parameter

Although the previous table of parameters only had one **sensitivity_param** line, multiply **sensitivity_param** parameters can be input in the superelement block, using the index following the **sensitivity_param** parameter to specify the parameter number.

Block Example

The above parameters are entered in the **block** section of the input file. For example,

```
BLOCK 10
  superelement
  file='example.cdf'
  // node  cid
   map 0    0
       0    0
       1    1
       1    2
       1    3
       2    1
       2    2
       2    3
  diagnostic=1
  sensitivity_param 1 0.01 // thickness in CBR shell model
  sensitivity_param 2 30e6 // modulus in CBR shell model
END
```

In this case, there are two sensitivity parameters, one for the thickness of a shell block in the Craig-Bampton model, and the other for the Young's modulus in that same block.

3.42 Interface Elements

The Interface Element (**InterfaceElement**), provides a means of connecting two mismatched meshes while still allowing compliance between them. The element is typically meshed into the geometry as a flattened tetrahedron. Initial implementation is with a node on face interaction, and with the face defined as a 6 noded triangle. Currently, only tet10 elements can be used for the element blocks on the contacting surfaces, since the interface elements themselves are written out at tet10 elements. Later extensions to this will include 3 noded triangle faces and 4 and 8 noded quad master faces. The mortar type connection to these faces should also be possible because the dual mortar connection does not couple the nodes on the slave side.

The element is a collection of springs and dashpots – no material model is used. The spring constants are described in the example below.

```
BLOCK 10
  interfaceelement
  normalstiffness 1.0
  tangentialstiffness 0.1
END
```

where **normalstiffness** and **tangentialstiffness** denote the spring constants for the normal and tangential springs, respectively.

The interface elements can also be used when **Sierra/SD** is to be run in a linear analysis (modal or transient) of a preloaded structure that has contact surfaces. The syntax is the same as described above, except that the solution block would have multiple cases corresponding to the preload and linear analysis, respectively. For example, a model with nonlinear contact may be preloaded in Adagio, and the results could then be transferred to **Sierra/SD** for a modal analysis. In this case, Adagio would write out the interface elements to the results file, and this results file would be used as the incoming mesh file for **Sierra/SD** . These elements would only be written to the parts of the interfaces where the contact is active, and thus **Sierra/SD** would correctly account for which nodes on the surfaces are in contact and which are not. Note that for the purposes of the linear analysis in **Sierra/SD** , the stresses in the incoming interface elements are all set to zero.

3.43 Dead

A **dead** element has no mass and no stiffness. It may be of any dimensionality, solid, planar, line or point. Interior nodes to a block of **Dead** elements will not be included in the computation of the model. There are no parameters for **Dead** elements.

3.44 Offset Elements and Lumped Mass

Offset elements can provide a tremendous advantage in modeling some structures including stiffened plates. Offset elements necessarily involve coupling between the rotational and translational degrees of freedom. This results in off diagonal coupling terms in the element stiffness and mass matrices.

Generally the element stiffness matrix is fully populated and seldom is reduced. However, the mass matrix may be diagonalized for a number of reasons. For example, the user may specify the "lumped" parameter in the solution section. Lumped mass matrices are always generated when running explicit analysis.

A lumped mass matrix loses all coupling between translational and rotational degrees of freedom. The model is changed significantly. Specifically, while the total mass is conserved the center of gravity and mass moments are not. The mass looks as if it had not been offset. This is true even with mesh refinement. The models of the consistent and lumped mass are fundamentally different when element offsets are included.

4 Stress/Strain Recovery

Stresses and strains are recovered at the centroids of the finite elements using standard finite element procedures. Stress and strain recovery is not implemented for 1-D elements. The stresses/strains calculated for shell elements are calculated in element space and not global space.

4.1 Stress/Strain Truth Table

The **Exodus** data format provides an element truth table. Element variables are defined globally (for all element blocks), but output data is stored only for those blocks that have entries in the truth table. Thus, in **Sierra/SD** if stress output is requested (see section 2.9.10), then stress variables are defined for solids and shells.[3] Space is allocated in the output **Exodus** file, and data is written only if it is applicable. Table 92 illustrates this for stresses. A similar table can be generated for strains. Note that volume stresses always start with "V" and surface stresses start with "S". Note that "vonmises" is the only entry that applies to both solids and shells.

4.2 Solid Element Stress/Strain

If stresses are requested, solid elements will output the values of stress at the element centroid.[4] The values reported are the engineering stresses in the global coordinate frame. That is,

$$\sigma_{ij} = \sum_{k,l} D_{ijkl}\epsilon_{kl} \tag{69}$$

Where D_{ijkl} is the material tangent modulus tensor, and

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

Here u and x are the displacement and coordinates in the basic coordinate frame.

4.3 Shell Element Stress/Strain

Shell elements introduce two complexities to stress/strain recovery. First, it is often important to recover data from the virtual surfaces of the elements (where the stresses are highest). This requires data recovery at the top, midplane and bottom surfaces. Second,

[3] The variables are defined for solids and shells even if only one or the other occurs in the model

[4] There is little point in reporting stresses elsewhere in the element as none of the post processing tools currently available properly manage stresses except at the centroids.

Table 92: Element Stress Truth Table

Variable Name	**Element**		
	Solid	**Shell**	**Beam**
SStressX1		σ_{xx}^{top}	
SStressY1		σ_{yy}^{top}	
SStressXY1		τ_{xy}^{top}	
SvonMises1		σ_{vm}^{top}	
SStressX2		σ_{xx}^{mid}	
SStressY2		σ_{yy}^{mid}	
SStressXY2		τ_{xy}^{mid}	
SvonMises2		σ_{vm}^{mid}	
SStressX3		σ_{xx}^{bottom}	
SStressY3		σ_{yy}^{bottom}	
SStressXY3		τ_{xy}^{bottom}	
SvonMises3		σ_{vm}^{bottom}	
VStressX	σ_{xx}		
VStressY	σ_{yy}		
VStressZ	σ_{zz}		
VStressYZ	σ_{yz}		
VStressXZ	σ_{xz}		
VStressXY	σ_{xy}		
VonMises	σ_{vm}	$\max(\sigma_{vm})$	
ElemForce			forces

there are no stresses or strains normal to the surface. Thus, stresses are naturally reported in the surface of the element. This can also introduce confusion about the inplane coordinate frames. As shown in Figure 77, the stresses and strains are recovered in the physical space x_1, x_2 coordinate frame, which has been mapped from the η_1, η_2 frame in element space. Note that the direction of the x_1 vector depends on the ordering of the mesh, and may vary from element to element in the same surface mesh. The element orientation vectors can be obtained with the **eorient** keyword described in section 2.9.25. The von mises stress, will of course be independent of the element orientation vectors (as it is an invariant).

The **TriaShell** stress recovery is described here. The **TriaShell** is a shell element created by combining Allman's triangle[40] and the DKT element.[41] The stress vector for the element is $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_{xy})^T$. This can be further broken down as:

$$\vec{\sigma} = \vec{\sigma}_{at} + \vec{\sigma}_{dkt} \tag{70}$$

where $\vec{\sigma}_{at}$ is the stress vector for Allmans's triangle and $\vec{\sigma}_{dkt}$ is the stress vector for the DKT element. Since Allman's triangle represents the membrane d.o.f., i.e., (u, v, θ_z), the stresses through the three surfaces of the shell element are the same. Therefore,

$$\vec{\sigma}_{at} = [D]\{\epsilon\} \tag{71}$$

where $\{\epsilon\}$ is the strain vector, and $[D]$ is the elasticity matrix for Allman's triangle. For the DKT element,

$$\vec{\sigma}_{dkt} = z[D]\{\kappa\} \tag{72}$$

where z is the coordinate direction normal to the element, with $z = 0$ representing the mid-plane, $[D]$ is the elasticity matrix for the DKT element, and

$$\{\kappa\} = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} \tag{73}$$

where β_x and β_y are rotations of the normal to the undeformed middle surface in the x-z and y-z planes, respectively (assuming the element lies in the x-y plane). $\vec{\sigma}_{dkt}$ does vary with the thickness of the element. Note, the above stress equations are written with respect to a local element coordinate system as shown in Figure 77.

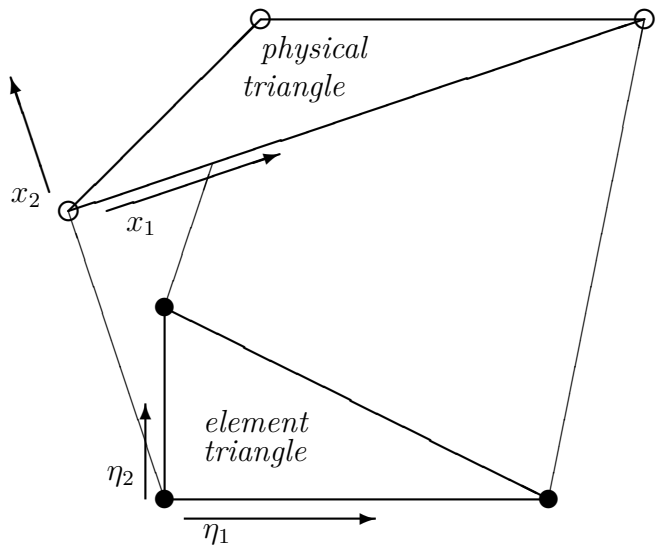
Combining the stress vectors from Allman's triangle and the DKT element above yields the stress vector for the element which is output in the local element frame.

For composite elements (such as QuadT, Quad8T and Tria6), the stresses are computed from the underlying Tria3 element and then transformed to the element orientation of the composite element. For the quad elements, the stress of the two central triangles is averaged. Figures 56, 57 and 61 describe these composite elements.

4.4 Line Element Stress/Strain

Reporting stresses for line type elements (Beams, Rods, Springs, etc) is even more problematic than it is for shells. For many of these elements an axial stress could be reported.

Figure 77: Tria3 Stress Recovery. Stresses are output in the orthogonal x_1, x_2 coordinate frame in physical space, which has been mapped from the η_1, η_2 frame in element space.



But, for beam elements that stress could not include the effects of beam bending unless details of the beam cross section were available. For some elements (such as a spring) no concept of stress is even correct. As a consequence, we do not report stresses for line type elements. Some recovery may be obtained using the element force output (see section 2.9.21).

5 Inverse Problems

Inverse methods include transient and direct frequency response load identification, direct frequency response material identification, and material identification from eigenvalues. The methods are based on solving optimization problems, with the goal to minimize the norm of the difference between measured and predicted data. More detail is provided in the references found in the theory notes (Theory section 1.6). Inverse methods for identifying an unknown material (2.1.12, 2.1.9) or an unknown load (2.1.8, 2.1.35) require solution block input. There may be additional input required, such as the specification of test data results 5.2.2 and the specification of the Optimization 5.1 and Inverse Problem 5.2 sections. Figure 78 illustrates a *partial* input for a "directfrf-materialid" problem. Highlighted portions of the input are outlined below.

Sierra/SD uses the Rapid Optimization Library (ROL) as an optimization engine. Portions of the ROL documentation can be found on the Trilinos website.[5]

[5] https://trilinos.org/packages/rol

```
SOLUTION
    directfrf-materialid
END
OPTIMIZATION
    optimization_package = ROL_lib
    ROLmethod = linesearch
    LSstep = secant
    opt_tolerance = 1e-10
    use_SimOpt = yes
END
INVERSE-PROBLEM
    data_truth_table = ttable.txt
    real_data_file = data.txt
    imaginary_data_file = data_im.txt
END
BLOCK 1
    inverse_material_type=homogeneous
    material 1
END
BLOCK 2
    inverse_material_type=known
    material 2
END
MATERIAL 1
    ISOTROPIC
    density 10
    G 1
    K 1
END
MATERIAL 2
    ISOTROPIC
    density 1
    G 2
    K 2
END
```

Figure 78: Sample "directfrf-materialid" Problem Input. Portions of the input that are specific to inverse methods are highlighted in red.

5.1 Optimization

The **optimization** section provides options to control the optimization strategy as part of an inverse method such as material identification. Parameters for the optimization section are listed in Table 93, and an example is shown in Figure 80.

Sierra/SD uses the Rapid Optimization Library (ROL),[48] which is a Trilinos[49] package for large-scale optimization. ROL is particularly well suited for the solution of optimal design, optimal control and inverse problems in large-scale engineering applications. The currently supported methods are trust region and line search, and the corresponding parameters for these methods are listed in Tables 93, 94 and 95. These are only a subset of the parameters available in ROL, and future **Sierra/SD** releases are expected to support even more of the available methods. We note that in tables 94 and 95, the abbreviations *tr* and *ls* stand for *trust region* and *line search*, respectively. We use these abbreviations to keep the parameter names succinct.

We note that for *source inversion* problems that involve **directfrf-loadid** or **transient-loadid** solution cases, we recommend using Krylov-based methods such as *line search* with the *newton-krylov* option, or *trust region* with *truncatedcg* option. An example for that case is given in Figure 79.

```
OPTIMIZATION
        ROLmethod = linesearch
        LSstep = newton-krylov
        Max_iter_Krylov = 50
        opt_tolerance = 1e-8
        opt_iterations = 5
END
```

Figure 79: Optimization Section Example for Source Inversion

In the case of *material inversion* problems, the best algorithm is problem-dependent, and may require some experimentation to arrive at the optimal parameters.

Sierra/SD currently uses ROL methods for unconstrained and bound-constrained optimization with line searches and trust regions (ROLmethod, boundConstraints, Table 93). To provide the context for the parameter tables, we review some notation and provide references to optimization textbooks.

Suppose $\mathcal{X}$ is a Hilbert space of functions mapping Ξ to $\mathbb{R}$. For example, $\Xi \subset \mathbb{R}^n$ and $\mathcal{X} = L^2(\Xi)$ or $\Xi = \{1, \ldots, n\}$ and $\mathcal{X} = \mathbb{R}^n$. We assume that the objective function $f : \mathcal{X} \to \mathbb{R}$ is twice continuously Fréchet differentiable and that the bound constraints a, $b \in \mathcal{X}$ are given with $a \leq b$ almost everywhere in Ξ. We focus on methods for solving unconstrained and bound-constrained optimization problems,

$$\underset{x}{\text{minimize}} \quad f(x) \qquad \text{and} \qquad \underset{x}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad a \leq x \leq b,$$

respectively. The methods implemented in ROL utilize derivative information and two strategies for guaranteed (global) convergence from remote starting points, line searches and trust regions. We use the notation $\nabla f(x)$ to the note the gradient of f at x and $\nabla^2 f(x)$ to denote the Hessian of f at x.

Line-search methods. Let x_k be the k-th optimization iterate. For unconstrained problems, line search methods compute an update to x_k in the form of

$$x_{k+1} = x_k + \alpha_k s_k,$$

where s_k is a descent vector, and $\alpha_k > 0$ is a scalar. The vector s_k can be computed using a variety of methods, including steepest descent, nonlinear conjugate gradients, quasi-Newton (secant) methods and Newton-Krylov methods, see [50, Ch. 3, 5, 8 and 6, resp.] and [51, Ch. 6.3]. Table 94 lists the parameters corresponding to the choice of the method to compute the descent vector s_k (LSstep, Table 94). To compute the scalar α_k, a line search approximately minimizes the one-dimensional function $\phi_k(\alpha) := f(x_k + \alpha s_k)$, i.e., it approximately solves the optimization problem

$$\underset{\alpha}{\text{minimize}} \quad \phi_k(\alpha) := f(x_k + \alpha s_k).$$

In general, the approximate minimizer α_k must satisfy sufficient decrease and curvature conditions to guarantee global convergence [50, Ch. 3]. **Sierra/SD** uses the *cubic interpolation* line search from ROL, which includes a backtracking procedure that satisfies the Armijo sufficient decrease condition

$$\phi_k(\alpha_k) \le \phi_k(0) + c_1 \alpha_k \phi_k'(0) \qquad \Longleftrightarrow \qquad f(x_k + \alpha_k s_k) \le f(x_k) + c_1 \alpha_k \langle \nabla f(x_k), s_k \rangle_{\mathcal{X}},$$

where $0 < c_1 < 1$, and does not require a curvature condition. An initial guess for the line-search parameter can be specified if steepest descent or nonlinear conjugate gradient methods are used for the computation of the descent vector s_k (initial_LS_Par, Table 94). For bound constrained problems, the line search is a *projected* search. That is, the line search approximately minimizes the one-dimensional objective function

$$\phi_k(\alpha) = f(P_{[a,b]}(x_k + \alpha s_k)),$$

where $P_{[a,b]}$ denotes the projection onto the upper and lower bounds. Such line-search algorithms result in projected gradient, projected quasi-Newton and projected Newton algorithms (see below).

Trust-region methods. For unconstrained problems, given the k-th iterate x_k trust-region methods compute the trial step s_k by approximately solving the trust-region subproblem

$$\underset{s}{\text{minimize}} \quad \frac{1}{2}\langle B_k s, s \rangle_{\mathcal{X}} + \langle g_k, s \rangle_{\mathcal{X}} \qquad \text{subject to} \qquad \|s\|_{\mathcal{X}} \le \Delta_k,$$

where $B_k \in L(\mathcal{X}, \mathcal{X})$ is an approximation of $\nabla^2 f(x_k)$, g_k approximates $\nabla f(x_k)$, and $\Delta_k > 0$ is the trust-region radius. The approximate minimizer s_k must satisfy the fraction of Cauchy decrease condition

$$-\frac{1}{2}\langle B_k s, s \rangle_{\mathcal{X}} - \langle g_k, s \rangle_{\mathcal{X}} \ge \kappa_0 \|g_k\|_{\mathcal{X}} \min\left\{ \Delta_k, \frac{\|g_k\|_{\mathcal{X}}}{1 + \|B_k\|_{L(\mathcal{X},\mathcal{X})}} \right\}$$

for some $\kappa_0 > 0$ independent of k. ROL implements several trust-region methods, including Cauchy point, dogleg, double dogleg, and truncated conjugate gradient methods, see [50, Ch. 4] and [51, Ch. 6.4]. Table 95 lists the parameters corresponding to the choice of the method (TRstep, Table 95). Additionally, the user can specify the initial trust-region radius, Δ_0 (initial_TR_Radius, Table 95). For bound constrained problems, ROL employs projected gradient, projected secant, and projected Newton-type methods. These methods prune variables based on the binding set (see below) and run standard trust-region subproblem solvers on the remaining variables. To ensure sufficient decrease, ROL then performs a modified projected line search.

Bound Constraints. The bound constraint methods in ROL require the active set of an iterate x_k,

$$\mathcal{A}_k = \{\, \xi \in \Xi \,:\, x_k(\xi) = a(\xi) \,\} \cap \{\, \xi \in \Xi \,:\, x_k(\xi) = b(\xi) \,\}.$$

The active set is the subset of Ξ corresponding to points in which x_k is equal to the upper or lower bound. The complement of the active set (called the inactive set), $\mathcal{I}_k = \mathcal{A}_k^c = \Xi \setminus \mathcal{A}_k$, is the subset of Ξ corresponding to points in which x_k is strictly between a and b. Given $\mathcal{A}_k$ and the gradient $g_k = \nabla J(x_k)$, we define the binding set as

$$\mathcal{B}_k = \{\, \xi \in \Xi \,:\, x_k(\xi) = a(\xi),\ -g_k(\xi) < 0 \,\} \cap \{\, \xi \in \Xi \,:\, x_k(\xi) = b(\xi),\ -g_k(\xi) > 0 \,\}.$$

The binding set contains the values of $\xi \in \Xi$ such that if $x_k(\xi)$ is equal to either the upper and lower bound, then $(x_k - g_k)(\xi)$ will violate bound. For both projected line-search and trust-region methods, the step is computed by fixing the variables in the active set and only optimizing over the inactive variables. That is,

$$s_k|_{\mathcal{A}_k} = 0 \quad \text{and} \quad s_k|_{\mathcal{I}_k} \text{ are free.}$$

Simply considering active, inactive and binding variables results in poor overall performance. To circumvent this behavior, ROL employs ϵ variations of this set. Namely, if $\epsilon > 0$, then

$$\begin{aligned}
\mathcal{A}_k^\epsilon &= \{\, \xi \in \Xi \,:\, x_k(\xi) \le a(\xi) + \epsilon \,\} \cap \{\, \xi \in \Xi \,:\, x_k(\xi) \ge b(\xi) - \epsilon \,\} \\
\mathcal{B}_k^\epsilon &= \{\, \xi \in \Xi \,:\, x_k(\xi) \le a(\xi) + \epsilon,\ -g_k(\xi) < 0 \,\} \cap \{\, \xi \in \Xi \,:\, x_k(\xi) \ge b(\xi) - \epsilon,\ -g_k(\xi) > 0 \,\}.
\end{aligned}$$

The ϵ inactive set is similarly defined as the complement of the ϵ active set. ROL dynamically controls ϵ so that as an algorithm approaches the optimal solution, ϵ decreases to zero.[52,53,54,55,56]

Krylov methods. Both line-search and trust-region methods may involve iterative methods of the Krylov type in the step computation. If such methods are requested, the stopping conditions for these sub-solvers can be defined through the parameters Absolute_Krylov_tol, Relative_Krylov_tol and Max_iter_Krylov described in Table 93. Scenarios requiring Krylov methods are triggered, for instance, if TruncatedCG is selected for a trust-region algorithm or if newtonkrylov is selected for a line-search algorithm.

Parameter	type	default	Description
ROLmethod	select	required	LineSearch, TrustRegion
boundConstraints	Yes/No	Yes	Bound constraints on design variables
scaleDesignVars	Yes/No	no	Controls scaling of design variables by initial guess to make them nondimensional
opt_tolerance	real	1e-10	Gradient tolerance
opt_iterations	int	20	Maximum iterations; an iteration may include one solve of the Newton system
objective_tolerance	real	1e-20	Objective function tolerance
Absolute_Krylov_tol	real	1e-6	Krylov absolute tolerance
Relative_Krylov_tol	real	1e-3	Krylov relative tolerance
Max_iter_Krylov	int	20	Krylov iteration limit

Table 93: Optimization Section Parameters

Parameter	type	default	Description
LSstep	select	secant	nonlinearcg, steepest, secant, newtonkrylov
initial_LS_Par	real	1.0	only applicable for nonlinearcg and steepest.

Table 94: Optimization Section Parameters for Line Search

Parameter	type	default	Description
TRstep	select	Cauchypoint	TruncatedCG, CauchyPoint, Dogleg, DoubleDogleg
initial_TR_Radius	real	-1	initial radius; if -1, ROL computes an initial value

Table 95: Optimization Section Parameters for Trust Region

```
OPTIMIZATION
        ROLmethod = linesearch
        LSstep = nonlinearcg
        Max_iter_Krylov = 50
        opt_tolerance = 1e-8
        opt_iterations = 5
END
```

Figure 80: Optimization Section Example

5.2 Inverse-Problem

Inverse problems optimize parameters to reproduce experimental results. The **inverse-problem** section of the input connects externally defined data describing the test. The format for the files is described in the Inverse Data Files section (5.2.2). This section also controls parameters of the optimization (such as regularization). Options for the **inverse-problem** section are included in Table 96.

Inverse Problem Parameters	**Load Id**		**Material Id**	
	Trans	**FRF**	**FRF**	**Eigen**
data_file	R	NA	NA	NA
data_truth_table	R	R	R	NA
real_data_file	NA	R	R	NA
imaginary_data_file	NA	R	R	NA
modal_data_file	NA	NA	NA	R
modal_weight_table	NA	NA	NA	R
Regularization Parameters				
gradientSurfaceRegParameter	NA	-	NA	NA
tikhonovParameter	-	-	-	-
MECE_penalty	NA	NA	-	NA

Table 96: Inverse-Problem parameters. Items marked with "-" are optional. Items marked 'R' are required, and items marked NA are not applicable.

5.2.1 Regularization Parameters

gradientSurfaceRegParameter a penalty term, penalizing jumps in load input parameters between neighboring patches. Increasing the regularization parameter decreases checkerboarding on multi-patch outputs.

tikhonovParameter a penalty term penalizing large values of the design variables. A larger term forces smaller design variables.

MECE_penalty a penalty term, penalizes the size of the misfit error between the measured and predicted data. Unlike least squares methods, MECE methods to not strive to exactly reproduce the measured data. Increasing the penalty term decreases the misfit error.

5.2.2 Inverse Data Files

The interface for measured data involves several data files. The specific files needed depends on the solution method, as summarized in Table 96. This section describes the data files and data files formats. Figure 81 provides an example of a frequency domain inverse problem.

```
inverse-problem
        data_truth_table = truthTable.txt
        real_data_file = dataReal.txt
        imaginary_data_file = dataImag.txt
end
```

Figure 81: The inverse problem section for direct frequency response sets the names of the files that contain user specified data.

Data Truth Table

The **data_truth_table** file contains the global node numbers (a.k.a. target nodes) where the experimental data measurements are given. The first line in the file contains the number of points where measurements are given, and the remaining lines contain the global node numbers where the experimental data is specified. Figure 82 provides a simple example.

The data truth table identifies the location of measurement data. For example, if the experimental data is collected at three microphones, which correspond to nodes 10, 120, and 3004, then the **data_truth_table** file is as follows.

```
3
10
120
3004
```

Thus there are a total of 4 lines in the file, even though the first line specifies three nodes for the measurement data. The global node numbers correspond to the global ID in the exodus input to **Sierra/SD**. The order of the nodes in this list corresponds to the order of the data in the corresponding real or imaginary data files. The first node in this list corresponds to the first list in the real (or imaginary) data file. The second node in this list corresponds to the second list in the real (or imaginary) data file, and so on. Other than that, there is no restriction on the ordering of the node numbers (i.e. they do not need to be in ascending or descending order).

Figure 82: Sample Data Truth Table Input for Acoustic Problem

Real Data File

For inverse problems in the frequency domain, the **real_data_file** file contains the real component of the measurement data at each frequency, corresponding to the nodes that are specified in the **data_truth_table** file. The first line of the file contains the number

```
4
5  0  1  0  0
6  0  1  0  0
7  0  0  0  1
8  0  0  0  1
```

Figure 83: Example data truth table for Structural Acoustics. There are 4 nodes, with numbers 5 to 8. Nodes 5 and 6 are structural where only the y component of displacement is measured. Nodes 7 and 8 are acoustic, where only the pressure is measured.

For acoustics, you need only specify the list of nodes. For structural and structural/acoustics problems the additional columns communicate which dofs are active variables in the optimization. Each of the four subsequent columns are the x, y, z and acoustic pressure, in that order.

```
4
25  1  1  1
96  1  1  1
13  1  1  1
17  1  1  1
```

Figure 84: Example Data Truth Table for Structures. There are 4 nodes, with numbers 25, 96, 13, 17. The second through fourth columns of 1, 1, 1 means that all structural degrees of freedom are active, the fifth column is acoustic and implied to be zero.

of nodes where measurement data is provided, followed by the number of frequencies of data. Starting on the second line, the real part of the data at the first node is given for all frequencies. In particularly the first node is the node first in the truth table list, not the node with the lowest ID number. Similarly, each subsequent line contains the real part of the data, at all frequencies, for the second node. An example is shown in Figure 85.

We build on the small example given in Figure 82 that has measurements at nodes 10, 120, and 3004, and consider the case where there are 2 frequencies in the data set. The **real_data_file** file could look as follows

```
3       2
1.1     2.4
0.7     3.3
2.1     1.4
```

The actual values in the above table were chosen arbitrarily, but observe that there is one "header row" followed by 3 data rows. There are 2 columns, corresponding to the two frequencies of the measured data. The units of the measurements must correspond to appropriate units in the analysis. In the frequency domain the units for the real part of the pressure of the liner spectrum (scaled Fourier transform of the pressure) are Pascal-seconds. For structures these are units of displacement.

Figure 85: Sample Real Data File Input

The frequencies of the measured data are specified in the **FREQUENCY** section, see section 2.11. The frequencies given by **FREQUENCY** section must correspond to the frequencies where the experimental data was measured. These frequencies can be either uniformly or non-uniformly spaced, as specified in the **FREQUENCY** section.

Imaginary Data File

The **imaginary_data_file** file has the exact same format as the **real_data_file** file, except that it contains the imaginary part of the data rather than the real part.

Data File

Transient time history data is stored in the **data_file**. The format is identical to the **real_data_file**. However, each column of data now corresponds to a time step of analysis. No interpolation is performed, and the measured data must exactly match the time steps of the analysis. As in the **real_data_file** each data row corresponds to a different measurement location.

We note that for time-domain inverse problems, the data in the **data_file** must be padded with zeros at the beginning, since the sensor data is typically started at time $t = 0$. The maximum time-of-flight from the input loads to the sensors can be estimated easily (an upper bound is fine). Then, dividing the maximum time-of-flight by the time step gives the number of zeros to be added to the beginning of each time history. Without these zeros, the forward problem would not be able to come up with loads that match the sensor data in the early time response, due to the finite wave propagation speed.

Modal Data File

For inversion with the `eigen` solution, measured modal results are stored in the **modal_data_file**. The first line of the file is the number of eigenvalues. Each subsequent line contains only the eigen frequency of the measured mode. It is important that the simulation modes are in the same order as the test modes.

Modal Weight Table

Not all computed eigenvalues may correspond to a test (or measured) mode. Further, even those modes identified may be more or less important. The **modal_weight_table** contains the weights applied to each computed mode. Weights must be non-negative.

Computed modes with no corresponding measured data should have zero weight. The first line contains the number of weights, which must match the number of modes computed in the analysis. Each subsequent line contains the weight for the corresponding computed mode.

5.3 BLOCK section for Material Identification

For material inverse problems, the **BLOCK** section provides an additional option to control the optimization strategy. In particular, blocks can be specified as **known**, **homogeneous**, or **heterogeneous**, as shown in Table 97, and an example is shown in Figure 86.

The default behavior for the **inverse_material_type** keyword is **known**, which implies that the material parameters given in the corresponding material are fixed, and are not modified in the optimization process. In the case where all blocks are known, there is no need to solve the material inverse problem at all.

The remaining two options for the **inverse_material_type** keyword are:

- **Homogeneous**. In this case, the material parameters are unknown and will be optimized during the inversion process, but are treated as constant over the entire

block. This option is best used in the case when material properties are unknown in a block, but not expected to vary much over the block.

- **Heterogeneous**. In this case, the material parameters are also unknown, will be optimized in the inversion process, but each element in the block will be given its own material properties. This is typically referred to as *spatially varying material properties*.

Figure 86 shows an example wherein Block 1 is **heterogeneous**, Block 2 is **homogeneous**, and Block 3 is **known**.

Parameter	type	Description
inverse_material_type	string	block inverse type

Table 97: Block Section Parameters for Material Inversion

```
BLOCK 1
        material 1
        inverse_material_type heterogeneous
END
BLOCK 2
        material 2
        inverse_material_type homogeneous
END
BLOCK 3
        material 1
        inverse_material_type known
END
```

Figure 86: Block Section Example for Material Inversion

5.4 MATERIAL section for Material Identification

For material inverse problems, the **MATERIAL** section provides additional options to control the optimization strategy. In particular, bound constraints can be specified as as shown in Table 99, and an example is shown in Figure 88. These parameters constrain the optimizer to work in a restricted space of possible design variable values, and prevent convergence to physically unrealistic values of the parameters.

We note that the parameters **G_bounds**, **K_bounds** only apply to elastic materials, whereas **Greal_bounds**, **Kreal_bounds**, **Gim_bounds**, and **Kim_bounds** only apply for viscoelastic materials.

Additional material parameters that can be optimized include **joint2g** elements and block proportional stiffness damping **blkbeta**. Since these parameters live in the **BLOCK** section rather than in the **MATERIAL** section, their bound constraints are specified in the former. However, we specify their input syntax here since they are material parameters. An example of input syntax for **joint2g** and **blkbeta** inputs for material optimization are given in Table 98 and Figure 87.

The **joint2g** elements involve 6 parameters that can be optimized. The **joint_truth_table** specifies which of these parameters will be included in the optimization solution. For example, in Figure 87, only the first (i.e. *x-component*) of the **joint2g** parameters will be optimized. The remaining parameters will stay fixed in the optimization, including those that are specified as *NULL*. In the case of **blkbeta**, the optimal *blkbeta* for the specified block will be computed and written to the result file.

Parameter	type	Description
joint_elastic_bounds	real real	bounds on joint2g spring stiffnesses
joint_damper_bounds	real real	bounds on joint2g dashpot (damper) parameters
blkbeta_bounds	real real	bounds on block stiffness proportional damping

Table 98: Block Section Parameters for Material Inversion

```
BLOCK 50
        joint2g
        kx = elastic 2.474e5
        ky = elastic 2.e8
        kz = elastic 2.e8
        krx = NULL
        kry = NULL
        krz = NULL
        joint_elastic_bounds = 2.0e5 1e9
        joint_truth_table = yes no no no no no
END

BLOCK 51
        inverse_material_type = homogeneous
        material 1
        blkbeta 2.0e-4
END
```

Figure 87: Block Section Example for Material Inversion

Initial guesses for the material parameters are also given in the **Material** block. In the case of elastic materials, the initial guess is specified as the values for G and K that are input in the **Material** block. In the example shown in Figure 88,the shear and bulk moduli are given initial guesses of 100 and 200, respectively.

For viscoelastic materials, the real and imaginary components of G and K are frequency-dependent. Thus the initial guesses are specified in functions that define the values of these parameters as a function of frequency. As shown in Figure 88, in this example functions $1-4$ specify the initial guesses for the real and imaginary components of G and K.

Parameter	**type**	**Description**
G_bounds	real real	lower and upper bounds on shear modulus
K_bounds	real real	lower and upper bounds on bulk modulus
Greal_bounds	real real	lower and upper bounds on real part of shear modulus
Kreal_bounds	real real	lower and upper bounds on real part of bulk modulus
Gimag_bounds	real real	lower and upper bounds on imag part of shear modulus
Kimag_bounds	real real	lower and upper bounds on imag part of bulk modulus

Table 99: Material Section Parameters for Material Inversion

```
MATERIAL 1
        ISOTROPIC
        G_bounds 0 1e4
        K_bounds 0 1e4
        G 100.0
        K 200.0
        density 10.0
END
MATERIAL 2
        ISOTROPIC_VISCOELASTIC_COMPLEX
        Greal_bounds 0 1e4
        Kreal_bounds 0 1e4
        Gim_bounds 0 1e2
        Kim_bounds 0 1e2
        Greal = function 1
        Gim = function 2
        Kreal = function 3
        Kim = function 4
        density = 10.0
END
```

Figure 88: Material Section Example for Material Inversion

Figure 100 shows which material parameters can be optimized in the different solution procedures. In general, parameters associated with damping can only be optimized in **directfrf** solutions, and those associated with stiffness can be optimized in either **directfrf** of **eigen**.

Parameter	directfrf-materialid	eigen-materialid
K	yes	yes
G	yes	yes
K_real	yes	no
K_imag	yes	no
G_real	yes	no
G_real	yes	no
joint2g elastic	yes	yes
joint2g damper	yes	no
blkbeta	yes	no

Table 100: Table of Supported Material Parameters for Inverse Methods

5.5 LOADS section for Load Identification

For inverse load problems, the **LOADS** section provides additional options to control the optimization strategy. Inverse loads are currently supported for both acoustic and structural sidesets. In particular, loads on sidesets can be specified as **known** or **spatially_constant**, as shown in Table 101, and an example is shown in Figure 89.

The default behavior for the **inverse_load_type** keyword is **known**, which implies that the loads on that sideset are fixed, and are not modified in the optimization process. In the case where all blocks were known, there would be no need to solve the inverse loads problem.

The remaining option for the **inverse_load_type** keyword is:

- **spatially_constant**. In this case, the load amplitude is unknown and will be optimized during the inversion process, but are treated as constant over the entire sideset.

Figure 89 shows an example wherein sideset 1 is **known**, sideset 2 is a real-valued, unknown acoustic load that is **spatially_constant**, and sideset 3 is an imaginary-valued, unknown acoustic load that is **spatially_constant**. We note that in the case of a transient problem, the **LOADS** block in an inverse loads problem would look the same except that there would be no imaginary loads in that case.

For acoustic problems, we note that currently inverse acoustic loads are limited to the **acoustic_accel** option. The **acoustic_vel** keyword is not supported for acoustic loads, but is planned to be in future releases.

Figure 90 shows a similar example for a structural pressure and traction load case. This example contains a known pressure load, and unknown pressure and traction loads. Note that in the case of traction loads, the direction of the traction load (in this case 111) is fixed, and only the function amplitudes are calculated in the inverse problem.

In Figure 89, functions $1-3$ would contain the initial guesses for the load amplitudes for sidesets $1-3$. These load amplitudes would then be refined during the optimization process. The resulting loads will be written to a text file called *force_function_data.txt*, which could then be included in a subsequent forward or inverse loads case in a restart analysis.

Parameter	**type**	**Description**
inverse_load_type	string	load inverse type

Table 101: Loads Section Parameters for Force Inversion

```
LOADS
        sideset 1
        acoustic_accel = 1
        function = 1
        sideset 2
        acoustic_accel = 1
        function = 2
        inverse_load_type = spatially_constant
        sideset 3
        iacoustic_accel = 1
        function = 3
        inverse_load_type = spatially_constant
END
```

Figure 89: Loads Section Example for Acoustic Force Inversion

5.6 Limitations for Inverse Load Problems

Limitations: There are a number of limitations which apply to transient load identification. These include the following.

1. Structural, acoustic and structural-acoustic domains may be addressed.
2. Only the simple Newmark integrator should be used, i.e. do not use generalized alpha integration and do not use the **rho** keyword.
3. For inverse loads each sideset must apply a unique function. You may *not* use the same function for more than one sideset.
4. Pressure is always applied along the surface normal. Inverse methods do not support follower pressures.
5. Load identification applies to **acoustic_accel** loadings on acoustic sidesets, and pressures/tractions on structural sidesets. Moments and point loads are not supported.

```
LOADS
        sideset 1
        pressure = 1
        function = 1
        sideset 2
        pressure = 1
        function = 2
        inverse_load_type = spatially_constant
        sideset 3
        traction = 1 1 1
        function = 3
        inverse_load_type = spatially_constant
END
```

Figure 90: Loads Section Example for Structural Force Inversion

6. In force identification problems, a pressure or traction may be applied on a shell. However the measured displacement fields (i.e. those in the truth table) cannot include a node with rotational DOFs.

5.7 ROL Output for Inverse Problems

Sierra/SD uses the Rapid Optimization Library (ROL), which is part of Trilinos[49] for solving optimization problems. During the optimization process, ROL writes out a text file called *ROL_Messages.txt* that contains information about the convergence of the optimization solution. It is important to examine this file to assure that the solution is adequately converged.

An example of a *ROL_Messages.txt* is given in Figure 91. The first 2 lines show the optimization method that was used by ROL, and the following lines contain convergence information. Each line corresponds to a single optimization iteration. The first column shows the iteration number under the *iter* heading. The second column shows the value of objective function at that iteration, under the *value* heading. The third column shows the absolute norm of the gradient (i.e. the derivative of the objective function with respect to the optimization variables). In Figure 91 we only show the first three columns of output as these will typically be of most interest to the user, but the remaining columns contain information about step size, number of function evaluations, etc... (Denis, Drew, any input here would be great).

For a typical **Sierra/SD** user, the first three columns in the *ROL_Messages.txt* file will typically be of the most important to pay attention to. As the goal is to minimize the objective function, a substantial decrease in the second column should be observed. Also,

the desired minimum of the optimization corresponds to a zero gradient, and thus the third column should be observed to be as close to zero as possible.

```
Newton-Krylov with Cubic Interpolation Linesearch satisfying
Null Curvature Condition Krylov Type  Conjugate Gradients
iter                 value                gnorm
0                    5.171112e-03         7.337197e-03
1                    1.160506e-10         1.245979e-06
2                    1.453148e-17         4.667996e-10
```

Figure 91: Example of *ROL_Messages.txt* file for Inverse Problem Solution

6 Troubleshooting

A variety of issues can cause an analysis to fail. Clearly, there are still bugs in the **Sierra/SD** software, and these will continue to be found. However, most problems are identified with problems in the model or other input to the software. This section may help to identify these issues with the goal of completing the analysis properly. Typically the fastest resolution to a problem is to try to eliminate the modeling issues, and only then treat the problem as a potential bug.

Users can troubleshoot **Sierra/SD** issues through stand-alone tools or using **Sierra/SD** capabilities. The following sections will describe some of the ways this may be done. The first part describes the stand-alone tools . The second part describes the ways of using **Sierra/SD** capabilities to troubleshoot problems or issues.

6.1 Stand-Alone Tools

Currently, two tools exist which can help the user debug their mesh file, i.e., **Exodus** file: **Grope** and **Cubit**.

6.1.1 Grope

Grope is an ACCESS/SEACAS utility that can be used to interrogate the **Exodus** file. One of the commands in **Grope** that can be used is **check**. It is used as follows:

```
prompt> grope cube.exo
.
.
GROPE> check

 Database check is completed

GROPE>
```

If there are any warning or errors, they will appear before the `Database check is completed` message.

6.1.2 Cubit

The Cubit team has developed a GUI-based tool named **Cubit**. **Cubit** can be used to look at various mesh quality parameters of the **Exodus** file. For questions about **Cubit**,

please contact the Cubit team at **cubit-dev@sandia.gov**.

6.2 Using Yada to identify disconnected regions

A common problem with many models is that part of the structure may be improperly connected to other components. Sierra/SD is particularly sensitive to issues of this type because they can cause computational issues for many linear solvers when the model is decomposed. Section 6.3.4 discusses this a little more.

The decomposition tool, **yada**, can be used to identify the problem areas. Please recognize that *yada* has a very specific definition for a "connected" mesh. That definition requires that elements be connected so as to eliminate rigid bodies. Figure 92 provides a simple example. While *yada* is designed to decompose a model, part of that task is identification of potential problem regions.

The steps for identification of these "disconnected" regions follow.

1. Run *yada* and attempt to split the model into only 1 region.

   ```
   mpirun yada example.exo 1
   ```

2. If *yada* finds no disconnected regions, you are done. Typically some regions may be found.

3. Use *nem_spread* to spread the exodus file into pieces. See section 3.5. The result is a number of exodus files, where each contains a fully "connected" region.

4. Alternatively, you can use *color_domains* to generate an exodus file with the domain number as an output variable. Standard visualization tools can be used to examine portions of the model.

   ```
   unix> color_domains example.exo example.nem example-colors.exo
   unix> ensight example-colors.exo
   ```

5. Examine each of the spread to understand the connectivity issue.

6.3 Using Sierra/SD To Troubleshoot

When running **Sierra/SD** on a parallel platform, new users will most likely face issues they are not accustomed to. One of the issues will be in choosing the correct **FETI** section parameters. Section 6.4 can help the user identify and troubleshoot the **FETI** parameters.

The user has to take additional steps before executing the parallel version of **Sierra/SD**. One of the steps is to run **nem_slice** or **yada** to decompose the finite element model.

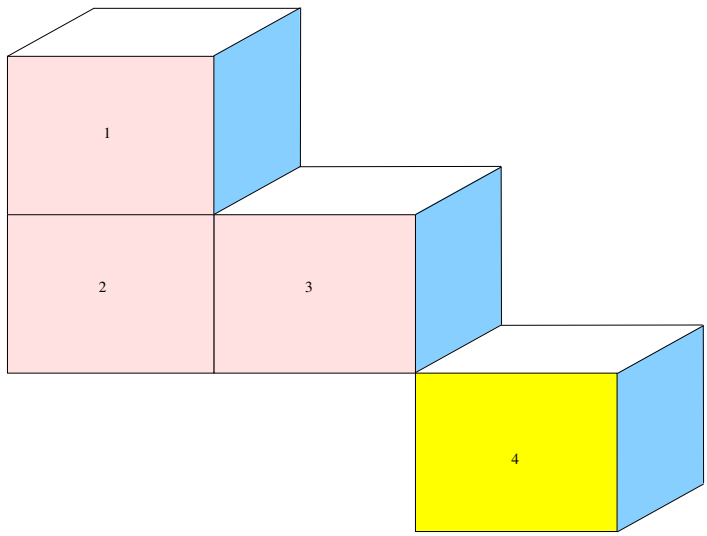


Figure 92: Problem Decomposition. The 3 hex elements on the left are properly connected. However, element 4 is connected only along an element edge. A mechanism is possible.

This will produce a load balancing file with a "`.nem`" extension. Using the **Exodus** file and the load balancing file, the next step is to run **nem_spread** to create the partitioned files on the parallel platform where **Sierra/SD** will be executed. Finally, the commands needed to run **Sierra/SD** on the parallel platform need to be learned so that execution of **Sierra/SD** can begin. Many of these steps can cause frustration to the user, but problems with any of these steps are often easily addressed.

The **FETI** solver is one of the most advanced solvers in the world, but it also is sensitive to the decompositions created. Therefore, a model that might appear to be working in serial can fail in parallel due to decomposition issues. Sometimes, the problem can be the model itself, e.g., a model that has been improperly equivalenced.

Sierra/SD developers have added various capabilities into **Sierra/SD** to help troubleshoot various issues.

6.3.1 Modal Analysis

There are a few things not to forget to try if the `eigen` solution method (discussed in section 2.1.11) diverges.

Section 2.3 mentions the `eig_tol` parameter. The default value is about 10^{-16}. Adequate modes can be determined with much larger values of `eig_tol`. Values such as 10^{-8} are reasonable.

The modal analysis algorithm depends on a linear solver, and assumes that the linear systems are solved accurately. Almost all of the iterative linear solver (GDSW, FETI,...) parameters trade off between speed and accuracy. For example, a very small value of the

`solver_tol`.

At times an analyst may choose to use the `eigen` method to diagnose a hidden problem. This can be done by using as large a value of `eig_tol` as is needed. The number of modes would also be as small as needed. After the hidden issue is resolved, if eigenvectors are still needed, don't forget to reset `eig_tol` to a small value.

6.3.2 Evaluating Memory Use

The **Sierra/SD** software tends to use a lot of memory. Matrices are generated and solved, and while this is often the fastest method of solution, it results in large memory demands. Parallel computing has its own issues for memory use.

Memory use diagnostics can be requested in the "ECHO" section of the input (see section 2.8).

6.3.3 Using The Node_List_File For Debugging Subdomains With RBMs

The *node_list_file* option is very useful in debugging subdomains that have ZEMs (or RBMs)[6]. To use this feature for debugging,

1. Make sure **prt_debug 3** is set in the **FETI** section. This will produce a `corner.data` file.
2. The "corner.data" file has the following format:

   ```
   NumCorners
   global_id local_id subdomain_id x_coord y_coord z_coord
   .
   .
   ```

3. Use *awk* (or similar utility) to obtain the local_ids of the subdomain from corner.data
4. Make sure to add an offset of 1 to the local_ids. Put these ids in a file, e.g., sub.corners.
5. Change the **Boundary** section of the Sierra/SD input file to include *node_list_file*:

   ```
   Boundary
     node_list_file="sub.corners"
       fixed
   End
   ```

6. Change the **geometry_file** to point to the subdomain being investigated.

[6] Zero Energy or Rigid Body modes

7. Run serial Sierra/SD using the parallel input file

This will help in debugging subdomains that are problematic.

6.3.4 Identifying Problematic Subdomains

Sometimes it is very difficult to identify subdomains that might be problematic. When running an eigen solution (in parallel), a shift is usually specified. Though the shift helps obtain solutions when global rigid body modes exist, this shift can also hide problematic subdomains. This issue also arises when running transient analysis. If a problematic subdomain is suspected, try an eigen analysis with a shift of zero. This will help identify subdomains with ZEMs. If ZEMs are discovered, then section 6.3.3 can help evaluate the source of the ZEMs on that subdomain using a serial version of Sierra/SD.

Sometimes bad subdomains can exist if the global model is not well connected. It is possible to use **yada** to try and create a one processor decomposition of the global Exodus file. If **yada** finds what appears to be disconnected pieces, it will add one processor for each disconnect piece. Once execution is complete, the **color_domains** utility can be used to create an Exodus file for visualization that will have the processor id as an element variable. Or, simply run **nem_spread** on the new decomposition and visualize each subdomain individually.

6.3.5 Problematic Elements and Connectivity

Many problems are caused by "bad" elements. Following are a few issues that come up periodically.

Rotational Invariance can be lost for certain elements such as springs if they are not of zero length. The spring shown in Figure 93 is invariant to rotation about the x axis, but not invariant to rotation about y or z. If we consider an undeformed rotation about the center of the beam along the z axis we would find that $u_y(1) < 0$ and $u_y(2) = -u_y(1)$. If the spring has **KY**$\neq 0$, then this undeformed rotation actually results in strain energy, $E = 2\,K_Y\,u_y^2$. Thus, the rigid body rotation is no longer a zero energy mode.

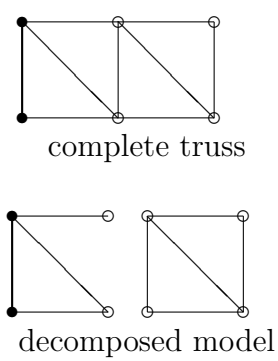
This is important for a variety of line type elements including spring, joint2g and gap elements.

Bad element shape is a major source of problems. For example, we have examined models that have "triangles" where one side is 1/200th the length of the other sides. This produces extremely poor element matrices. In some cases this can destroy the condition of the entire system. Such elements can sometimes be found using the **kdiag** output option described in section 2.9.32.

Figure 93: Single *Spring* element

Decomposition weakness is an issue for trusses (or rods) and some other elements. The truss in the top part of Figure 94 is self sustaining when made of truss elements. However, because truss elements have no rotational stiffness, the decomposed model in the lower part of the figure contains mechanisms. Note that there is no way to decompose the model without introducing such mechanisms.

Figure 94: Truss Decomposition Issues



This does not mean that truss elements must not be used in **Sierra/SD** . There are times when they are the correct element to use. However, extreme care must be taken in their decomposition, and occasionally extra "corner nodes" may be needed to avoid mechanisms (see section 2.4.1).

Poor Connectivity A structure that has poorly connected regions can be very difficult to analyze. If elements have not been properly equivalenced, there can be thousands of zero energy modes in the model. **Sierra/SD** is fairly good at identifying up to a few dozen redundant modes in the best of cases.

6.4 Troubleshooting FETI Issues

6.4.1 Introduction

The Finite Element Tearing and Interconnecting (FETI) solver achieves unprecedented speed and scalability on massively parallel computers. However, it is significantly more complex than a standard direct solver. We discuss a number of the options associated with the solver in the following sections. These options are required to achieve three sometimes-competing goals.

1. Insuring that there is sufficient memory to run on the MP platform.
2. Obtaining the current solution through correct rigid body (or zero energy) identification on the subdomain and on the coarse grid.
3. Tuning the solver to maximize performance.

6.4.2 Standard FETI Block

The default entries for the FETI block are shown below.

```
FETI
    rbm                   geometric
    preconditioner        dirichlet
    corner_algorithm      1
    corner_dimensionality 6
    corner_augmentation   none
    max_iter              200
    orthog                1000
    solver_tol            1e-6
    grbm_tol              1e-6
    coarse_solver         sparse
    local_solver          sparse
    precondition_solver   sparse
    prt_summary           yes
    prt_rbm               yes
    prt_debug             2
END
```

6.4.3 Memory

The FETI options that directly affect memory usage are listed in the following table. Memory is directly related to the "size" of a subdomain. The number of elements associated

with a subdomain can approximately measure the "size". The topology or connectivity of those elements also directly affects the memory since this determines the local sparse matrix structure.

Large memory allocations occur in the following order with the relative importance listed in parentheses. These operations are only done once for linear static/dynamic and eigen analysis in Sierra/SD.

1. Preconditioner (3)
2. Local Solver (2)
3. Coarse Grid (1)
4. Orthog vectors (4)

6.4.3.1 Preconditioner The lumped preconditioner requires less memory but generally does more iterations than the dirichlet preconditioner which requires more memory. The `precondition_solver` option only affects the memory if the Dirichlet preconditioner is selected. Then the comments in the Local Solver section also apply.

6.4.3.2 Local Solver The skyline solver typically takes more memory than the sparse solver. For small problems (less than 1000 equations), the skyline solver may require less memory than the sparse solver. Generally the skyline solver is the more robust option particularly when the solution may be singular (i.e. eigenvalue analysis on a floating structure).

6.4.3.3 Coarse Solver The corner algorithm, corner dimensionality, corner augmentation, and coarse solver options affect the coarse grid memory requirements. The number of equations in the coarse grid can be found in the solution.data file. Reducing the number of equations in the coarse grid reduces the memory required by the coarse grid.

If your model has shell elements, then corner dimensionality 6 results in more memory than corner dimensionality 3. If your model does not have shells, then this option will not affect memory. Corner dimensionality 6 is generally required for good performance on shell models.

Corner algorithm memory requirements are model dependent and are directly related to the interface topology of the decomposed global model. Typically, algorithm 0 results in the smallest coarse grids. This is also the least robust corner algorithm. Corner algorithm 3 is the most conservative corner algorithm and typically generates larger coarse grids. It is recommended to start with `corner_algorithm=1`. If problems arise, change to corner algorithm 3.

Both the skyline and sparse coarse grid solvers are redundantly stored on every processor. The same comments about the skyline and sparse solvers found in the Local Solver section apply here too. The parallel sparse (psparse) solver distributes the coarse grid memory among

Ns coarse solver processors. Very large coarse grids can be used with this option. If there are any problems found with the parallel sparse solver, please contact me at khpiers@sandia.gov.

6.4.3.4 Orthogonalization (Ortho) Vectors The number of ortho vectors directly affects the memory requirements of FETI-DP. Generally, you want to select as many ortho vectors as possible given the memory limitations. Ortho vectors decrease the number of iterations required for successive right hand side vectors (eigen/dynamic analysis).

Memory Diagnostics

On most platforms, diagnostics on the memory usage can be obtained using the **prt_debug** keyword. If `prt_debug` is greater than 2, then a memory diagnostics file, `memory.data` is written from which one may determine the memory statistics on multiple processors. Unfortunately, not all platforms currently support memory diagnostics. On those platforms, memory.data will have no meaningful information.

Options that Affect Memory

```
FETI
     preconditioner [lumped/dirichlet]
     precondition_solver [skyline/sparse]
     orthog 200
     local_solver [skyline/sparse]
     coarse_solver [skyline/sparse/psparse]
     corner_dimensionality [3/6]
     corner_algorithm [0,1,2,3,4]
     corner_augmentation [none/subdomain/edge]
     prt_debug=3 // set 2 or higher for diagnostics
END
```

6.4.4 Local Rigid Body Modes

Local rigid body modes (RBMs) refer to the local subdomain stiffness matrix having singularities found during the LDLT factorization and in general the solution will be corrupted if local RBMs are found. The command "prt_rbm yes" in the FETI block will print the number of local RBMs found for each subdomain in your model. Each subdomain is expected to have zero local RBMs. The following steps can be taken if you find a subdomain with a non-zero number of local RBMs.

1. Reduce the tolerance used in the LDLT factorization, For example, the default value for "rbm_tol_mech" is 1.0E-08, then try "rbm_tol_mech 1.0E-12"

2. If this does not remove the local RBMs, then try changing the corner algorithm while holding the previously set tolerance constant. The recommended and default algorithm is 1. If corner algorithm 1 fails to remove the local RBMs, then try corner algorithm 3.

3. If you have shell elements in the model (and more specifically in the subdomain you have found local RBMs), then "corner dimensionality 6" may be required.

4. For more detailed debugging of RBMs (or ZEMs) for specific subdomains, see section 6.3.3.

5. If you still have local RBMs, contact me at khpiers@sandia.gov and I'll be happy to look at your specific problem.

6.4.5 Global Rigid Body Modes

Global rigid body modes (RBMs) refer to the global stiffness matrix having singularities present. Finding 6 RBMs for a 3D model is expected when performing an eigen analysis with Sierra/SD and the global model does not have any prescribed displacement boundary conditions. FETI-DP can handle this case, but in many cases tolerances have to be adjusted for a particular model.

Finding the incorrect number of RBMs can lead to either stagnation in the FETI solution or the dreaded "relative residual greater than 1" error in Sierra/SD. Troubleshooting this problem can be done in the following fashion.

1. First, determine the expected number of RBMs in your model. Typically in eigen analysis, this is zero (fully constrained), three (2D-floating), or six (3D-floating). The number of RBMs is expected to be zero for transient dynamics.

2. Next, determine how many you are finding with the FETI parameters you have selected. The number of global RBMs are printed to the screen during a Sierra/SD run and printed to the solution.data file. Executing the following UNIX command will find the number of global RBMs found during the last Sierra/SD run. grep "Global RBM" solution.data

3. The parameter "grbm_tol 1.0E-06" will have to be adjusted to find the expected number of RBMs in your model.

4. Decrease grbm_tol if you want to find less global RBMs.

5. Increase grbm_tol if you want to find more global RBMs.

6. For eigen analysis, you may want to use a negative shift (in the Sierra/SD SOLUTION block). Use a shift value equal to the negative of the first anticipated flexible eigenvalue, i.e. $(2\pi f)^2$. This should eliminate all global RBMs, but may slow the solution.

7. If you still have problems with global RBMs, please contact Kendall Pierson at `khpiers@sandia.gov`, and I will be happy to help resolve the problem.

7 Acknowledgments

Sierra/SD is a success based on work by many individuals and teams. These include the following.

1. The ASCI program at the DOE which funded its development.
2. Line managers at Sandia Labs who supported this effort. Special recognition is extended to David Martinez who helped establish the effort.
3. Charbel Farhat and the University of Colorado at Boulder. They have provided incredible support in the area of finite elements, and especially in development of FETI.
4. Carlos Felippa of U. Colorado at Boulder. His consultation has been invaluable, and includes the summer of 2001 where he visited at Sandia and developed the HexShell element for us.
5. Danny Sorensen, Rich Lehoucq and other developers of **ARPACK**, which is used extensively for eigen analysis.
6. Esmond Ng who wrote *sparspak* for us. This sparse solver package is responsible for much of the performance in Sierra/SD and in FETI.
7. The *metis* team at the university of Minnesota. *Metis* is an important part of the graph partitioning schemes used by several of our linear solvers. These are copyright 1997 from the University of Minnesota. Documentation is available at, http://www-users.cs.umn.edu/ karypis/metis/metis/index.html.
8. Padma Raghaven for development of a parallel direct solver that is a part of the FETI solver.
9. The developers of the *SuperLU* package. This is used in a variety of areas, including a sparse direct complex solver. More information can be obtained at, http://www.nersc.gov/ xiaoye/SuperLU.

References

[1] Sjaardema, G. D., "APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses," Tech. Rep. SAND92-2291, Sandia National Laboratories, 1992.

[2] Schoof, L. A. and Yarberry, V. R., "EXODUS II: A Finite Element Data Model," Tech. Rep. SAND92-2137, Sandia National Laboratories, 1994.

[3] Johnson, C. D., Kienholz, D. A., and Rogers, L. C., "Finite element prediction of damping in beams with constrained viscoelastic layers," *AIAA Journal*, **vol. 20**, no. 9, 1982, pp. 1284–1290.

[4] Avery, P., Farhat, C., and Reese, G., "Fast Frequency Sweep Computations Using a Multi-point Pade-Based Reconstruction Method and an Efficient Krylov Solver," *Int. J. Numer. Meth. Engng.*, **vol. 69**, no. 13, Sept 2006, pp. 2848–2875.

[5] Lehoucq, R. B., Sorensen, D., and Yang, C., *ARPACK Users' Guide*, SIAM, Philadelphia, PA, USA, 1998.

[6] Baker, C. G., Hetmaniuk, U. L., Lehoucq, R. B., and Thornquist, H. K., "Anasazi Software for the Numerical Solution of Large-Scale Eigenvalue Problems," *ACM Transactions on Mathematical Software*, **vol. 36**, no. 3, 2009, pp. 13:1–13:23.

[7] Wirsching, P. H. and Light, M. C., "Fatigue under wide band random stresses," *Journal of the Structural Division, ASCE*, **vol. 106**, no. 7, 1980, pp. 1593–1607.

[8] Wirsching, P. H., Paez, T. L., and Ortiz, K., *Random Vibrations, theory and practice*, John Wiley and Sons, Inc, 1995.

[9] Reese, G., Field, R., and Segalman, D. J., "A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments," *Shock and Vibration. Digest*, **vol. 32**, no. 6, 2000.

[10] Segalman, D., Reese, G., Field, R., and Fulcher, C., "Estimating the Probability Distribution of von Mises Stress for Structures Undergoing Random Excitation," *Transactions of the ASME*, **vol. 122**, January 2000.

[11] Kinsler, Frey, Coppens, and Sanders, *Fundamentals of Acoustics*, John Wiley & Sons, 1982.

[12] Farhat, C., Crivelli, and Géradin, M., "Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory," *Int. J. Numer. Meth. Engng.*, **vol. 41**, 1998, pp. 675–696.

[13] Chung, J. and Hulbert, G., "A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized alpha method," *Journal of Applied Mechanics*, **vol. 60**, 1993, pp. 371–375.

[14] S D Team, "Sierra Structural Dynamics Verification," Tech. Rep. SAND2011-7898P, Sandia National Laboratories, 2011.

[15] Dohrmann, C. R., "A study of two domain decomposition preconditioners," Tech. rep., Sandia National Laboratories, SAND2003-4391, Albuquerque, New Mexico, 2003.

[16] Dohrmann, C. R. and Widlund, O. B., "Hybrid domain decomposition algorithms for compressible and almost incompressible elasticity," *Int. J. Numer. Meth. Engng.*, **vol. 82**, 2010, pp. 157–183.

[17] Farhat, C. and Roux, F.-X., "A Method of Finite Element Tearing and Interconnecting and Its Parallel Solution Algorithm," *Int. J. Numer. Meth. Engng.*, **vol. 32**, 1991, pp. 1205–1227.

[18] Cook, R. D. and D. S. Malkaus, M. E. P., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, third edn., 1989.

[19] Puso, M. A., "A 3D Mortar Method for Solid Mechanics," *Int. J. Numer. Meth. Engng.*, **vol. 59**, 2004, pp. 315–336.

[20] Knupp, P. M., "Achieving Finite Element Mesh Quality Via Optimization of the Jacobian Matrix Norm and Associated Quantities : Part II - A Framework for Volume Mesh Optimization and the Condition Number of the Jacobian Matrix," Tech. Rep. SAND99-0709J, Sandia National Laboratories, 1998.

[21] Thompson, D., Pébay, P. P., and Jortner, J. N., "An Exodus II Specification for Handling Gauss Points," Tech. Rep. SAND2007-7169, Sandia National Laboratories, 2007.

[22] S D Team, "Sierra Structural Dynamics How To," 2000.

[23] S D Team, "SALINAS Program Notes," Tech. Rep. SAND2008-2559P, Sandia National Laboratories, 2008.

[24] S D Team, "Sierra Structural Dynamics - Theory Manual," Technical Report SAND2015-9131, Sandia National Laboratory, PO Box 5800, Albuquerque, NM 87185-5800, October 2015.

[25] Walsh, T. F., Reese, G. M., Dohrmann, C., and Rouse, J., "Finite element methods for structural acoustics on mismatched meshes," *Journal of Computational Acoustics*, **vol. 17**, no. 3, 2009, pp. 247–275.

[26] Brown, K. and Voth, T., "ACME: Algorithms for Contact in a Multiphysics Environment, API Version 1.3," *SAND Report 2003-1470, Sandia National Laboratories*, 2003.

[27] Aklonis, J. L. and MacKnight, W. L., *Introduction to Polymer Viscoelasticity*, Wiley, 1983.

[28] Ferry, J. D., *Viscoelastic Properties of Polymers*, Wiley, 1980.

[29] Hamilton, M. F. and D. T. Blackstock, E., *Nonlinear Acoustics*, Academic Press, 1998.

[30] Carroll, S. K., Drake, R. R., Hensinger, D. H., Luchini, C. B., Petney, S. J. V., Robbins, J. H., Robinson, A. C., Summers, R. M., Voth, T. E., Wong, M. K. W., Brunner, T. A., Garasi, C. J., Haill, T. A., and Mehlhorn, T. A., "ALEGRA: Version 4.6," Tech. Rep. SAND2004-6541, Sandia National Laboratories, 2004.

[31] Engineering, A., "Attune User's Guide," .

[32] Alvin, K. F., Reese, G. M., Day, D. M., and Bhardwaj, M. K., "Incorporation of Sensitivity Analysis into a Scalable, Massively Paralle, Structural Dynamics FEM Code," Boulder, CO, August 1999.

[33] Alvin, K. F., "Implementation of Modal Damping in a Direct Implicit Transient Algorithm," April 2001.

[34] Taylor, R. L., Beresford, P. J., and Wilson, E. L., "A Non-conforming Element for Stress Analysis," *Int. J. Numer. Meth. Engng.*, **vol. 10**, 1976, pp. 1211–1219.

[35] Ibrahimbegovic, A. and Wilson, E. L., "A Modified Method of Incompatible Modes," *Communications in Applied Numerical Methods*, **vol. 7**, 1991, pp. 187–194.

[36] MacNeal, R. H., *Finite Elements: Their Design and Performance*, Marcel Dekker, 1994.

[37] Reddy, J. N., *An Introduction to the Finite Element Method*, McGraw Hill, 1984.

[38] Ochoa, O. O. and Reddy, J. N., *Finite Element Analysis of Composite Laminates*, Kluwer Academic Publishers, 1992.

[39] Belytschko, T., Tsay, C., and Liu, W., "A stabilization matrix for the bilinear Mindlin plate element," *Computer methods in applied mechanics and engineering*, **vol. 29**, no. 3, 1981, pp. 313–327.

[40] Allman, D. J., "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems," *Computers and Structures*, **vol. 19**, no. 1-2, 1996, pp. 1–8.

[41] Batoz, J.-L., Bathe, K.-J., and Ho, L.-W., "A Study of Three-Node Triangular Plate Bending Elements," *Int. J. Numer. Meth. Engng.*, **vol. 15**, 1980, pp. 1771–1812.

[42] Felippa, C. A., "The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation," Tech. Rep. CU-CAS-02-03, Univ. Colo. at Boulder, 2002.

[43] Oden, J. T., *Mechanics of Elastic Structures*, McGraw-Hill, Inc., first edn., 1967.

[44] Smallwood, D. O., Gregory, D. L., and Coleman, R. G., "A three parameter constitutive model for a joint which exhibits a power law relationship between energy loss and relative displacement," in *Shock and Vibration Symposium*, Destin, FL, 2001.

[45] Segalman, D. J., "An Initial Overview of Iwan Modeling for Mechanical Joints," Tech. Rep. SAND2001-0811, Sandia National Laboratories, 2001.

[46] Segalman, D. J. and Starr, M. J., "Relationships Among Certain Joint Constitutuve Models," Tech. Rep. SAND2004-4321, Sandia National Laboratories, 2004.

[47] Smallwood, D. O., Gregory, D. L., and Coleman, R. G., "A three parameter constitutive model for a joint which exhibits a power law relationship between energy loss and relative displacement," Tech. Rep. SAND2001-1758C, Sandia National Laboratories, November 2001.

[48] Kouri, D. P., Ridzal, D., van Bloemen Waanders, B. G., and von Winckel, G., "Rapid Optimization Library, SAND2014-19572 C, Sandia National Laboratories, https://trilinos.org/packages/rol," 2014.

[49] Heroux, M., Bartlett, R., Howle, V., Hoekstra, R., J., H., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., Williams, A., and Stanley, K., "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, **vol. 31**, no. 3, 2005, pp. 397–423.

[50] Nocedal, J. and Wright, S., *Numerical Optimization*, Springer, 2006.

[51] Dennis, Jr. , J. E. and Schnabel, R. B., *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*, SIAM, Philadelphia, 1996.

[52] Kelley, C. T. and Sachs, E. W., "A Trust Region Method for Parabolic Boundary Control Problems," *SIAM J. Optimization*, **vol. 9**, 1999, pp. 1082–1099.

[53] Bertsekas, D. P., "Projected Newton Methods for Optimization Problems with Simple Constraints," *SIAM J. Control and Optimization*, **vol. 20**, 1982, pp. 221–246.

[54] Lin, C.-J. and Moré, J. J., "Newton's method for large bound-constrained optimization problems," *SIAM J. Optim.*, **vol. 9**, no. 4, 1999, pp. 1100–1127 (electronic).

[55] Burke, J. V., Moré, J. J., and Toraldo, G., "Convergence properties of trust region methods for linear and convex constraints," *Math. Programming*, **vol. 47**, 1990, pp. 305–336.

[56] Calamai, P. H. and Moré, J. J., "Projected gradient methods for linearly constrained problems," *Math. Programming*, **vol. 39**, 1987, pp. 93–116.

[57] Hendrickson, B. and Leland, R., "The Chaco Users Guide," Tech. Rep. SAND1995-2344, Sandia National Laboratories, 1995.

1 Sierra/SD Example Input Files

The following sections give examples of **Sierra/SD** input files. Note, case sensitivity of the keywords is ignored unless in quotes. The exception is the **#include** command, where the filename following the command must not be in quotes, but case sensitivity is preserved.

1.1 An Eigenanalysis Input File

The following input file will output the first four mode shapes to an **Exodus** output file name *hexplate-out.exo*. A results file, *hexplate.rslt*, will not be created since no results have been selected for output in the **ECHO** section.

```
SOLUTION
         eigen
         nmodes 4
         title 'Obtain First Four Mode Shapes For Hexplate'
END

// The f.e.m. is in hexplate.exo
FILE
         geometry_file         'hexplate.exo'
END

BOUNDARY
         nodeset 77
                  fixed
END

LOADS  // loads are unnecessary for eigenanalysis
END

// Only deformations will be output
OUTPUTS
//         maa
//         kaa
//         faa
         deform
//         stress
//         strain
END

// No results are output to the text log file, *.rslt
```

```
ECHO
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
 none
END


// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default hex is an underintegraged hex.
BLOCK 44
         material 3
         hex8
END

MATERIAL 3
         name "steel"
         E 30e6 +/- 1 %
         nu .3
         density 0.288
END

SENSITIVITY
         values all
END
```

1.2 An Anisotropic Material Input File

The following input file is an example of a hexahedral mesh with anisotropic properties.

```
SOLUTION
         eigen
         title 'Example of anisotropic format'
END

FILE
         geometry_file         'anisogump.exo'
END

boundary
        nodeset 4 y = 0
        nodeset 5 x = 0
        nodeset 6 z = 0
end

loads
         // sum of forces on surface should be equal to area
         // imposed forces are additive
         nodeset 1 force = 0.0  0.083333  0.0
         nodeset 2 force = 0.0 -0.041666  0.0
         nodeset 3 force = 0.0 -0.020833  0.0
end


OUTPUTS
//         maa
//         kaa
//         faa
           deform
//         stress
//         strain
END

ECHO
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
```

```
//  STRAIN
//  STRESS
//  MFILE
none
END


// the following element block  is all hex
BLOCK 1
         hex8
         material 1
END

MATERIAL 1
         name "anisotropic gump"
         anisotropic
         Cij
         1.346     0.5769    0.5769  0       0        0
                   1.346     0.5769  0       0        0
                             1.346   0       0        0
                                     0.3846  0        0
                                             0.3846   0
                                                      0.3846
         density 1
END
```

1.3 A Multi-material Input File

The next example shows the input for an **Exodus** model with many element blocks and materials. Keyword **lumped** in the **SOLUTION** section causes **Sierra/SD** to use a lumped mass matrix instead of a consistent mass matrix.

```
SOLUTION
         eigen
         nmodes 1
         title 'Multiple block, multiple material example'
         lumped
END

FILE
         geometry_file         'multi.exo'
END
```

```
BOUNDARY
    nodeset 1
    fixed
    nodeset 3
    x = 0
    y = 0
    z = 0
    RotY = 0
    RotZ = 0
END

OUTPUTS    // output only displacements to exodus file
     deform
END

ECHO
    none
END

// element block specifications. One such definition per element
// block in the exodus (genesis) database.
BLOCK 1
         material 2
         Beam2
END

BLOCK 101
         integration full
         wedge6
         MATERIAL 1
END

BLOCK 2
         material 2
END

BLOCK 102
         integration full
         wedge6
         MATERIAL 2
END

BLOCK 3
         material 3
```

```
END

BLOCK 103
         integration full
         wedge6
         MATERIAL 3
END

BLOCK 4
         material 4
END

BLOCK 104
         integration full
         wedge6
         MATERIAL 4
END

BLOCK 5
         material 5
END

BLOCK 105
         wedge6
         integration full
         MATERIAL 5
END

BLOCK 6
         material 6
END

BLOCK 106
         wedge6
         integration full
         MATERIAL 6
END

// material specifications. Extra materials are acceptable, but
// every material referenced in a necessary "Block" definition,
// must be included here.
MATERIAL 1
         name "Phenolic"
         E 10.5E5
         nu .3
```

```
         density 129.5e-6
END

Material 2
         name 'Aluminum'
         E 10.0E6
         nu 0.33
         density 253.82e-6
END


Material 3
         name 'foam'
         E 100.
         nu 0.3
         density 18.13e-6
END

Material 4
         name 'HE'
         E 5E5
         nu 0.45
         density 129.5e-6
END

material 5
         name 'Uranium'
         E 30e6
         nu 0.3
         density 1768.97e-6
end

material 6
         name 'wood'
         E 200.e3
         nu .3
         density 77.7e-6
end
```

1.4 A Modaltransient Input File

The next example shows the input for a **modaltransient** analysis. Accelerations are output to an **Exodus** file *bar-out.exo*. This example has damping, polynomial and linear functions. Also, sensitivities are calculated.

```
SOLUTION
  modaltransient
    nmodes 10
    time_step .000005
    nsteps 100
    nskip 1
    title 'Test modal transient on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
END

MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
```

```
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 0.0001 1.0
  data 0.0001 0.0
  data 1.0 0.0
END
```

1.5 A Modalfrf Input File

The next example shows the input for a **modalfrf** analysis. Accelerations are output to an **Exodus** file *bar-out.frq*.

```
SOLUTION
  modalfrf
    nmodes 10
    title 'Test modalfrf on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

frequency
   freq_min 0
   freq_step=10
   freq_max=3000
   nodeset 3
   disp
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
END
```

```
MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 2
// this is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945
  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 10000. 1.0
END
```

1.6 A Directfrf Input File

The next example shows the input for a **directfrf** analysis. Displacements are output to an **Exodus** file *bar-out.frq*.

```
SOLUTION
directfrf
END

Frequency
  freq_min = 1000.0
  freq_step = 7000
  freq_max = 5.0e4
  disp
  block 1
End

FILE
  geometry_file 'bar.exo'
END

OUTPUTS
disp
END

ECHO
//
none
END

BOUNDARY
  nodeset 1
    fixed
END

BLOCK 1
  material 1
END

MATERIAL 1
  name "aluminum"
  G 0.8E+9
  K 4.8E+9
```

```
  density 2.59e-4
END

LOADS
  sideset 1
  pressure = -1.0
  function=3
END

FUNCTION 3
   type LINEAR
   name "white noise"
   data 0.0 1.0
   data 10000. 1.0
END
```

1.7 A Statics Input File

The following example is a **statics** analysis which will output stresses to the **Exodus** output file *quadt-out.exo*.

```
SOLUTION
        statics
        title '10x1 beam of quadt'
END

FILE
         geometry_file         'quadt.exo'
END

BOUNDARY
           nodeset 1
            fixed
END

LOADS
           nodeset 2
            force = 1000.0 1000.0 0.0
END

OUTPUTS
      stress
END

ECHO
  none
END

// the following element block is quadt
BLOCK 1
         material 1
         QuadT
END

MATERIAL 1
         name "steel"
         E 30.0e6
         nu 0.25e0
         density 0.7324e-3
END
```

2 Running Sierra/SD on serial UNIX platforms

On serial Unix platforms, **Sierra/SD** is run with a single argument, the ASCII input file.

```
salinas example.inp
```

The log file will be written to *example.rslt* if outputs have been specified in the ECHO section. If outputs have been specified in the OUTPUTS section, a new exodus file will be generated. The file name is derived from the `geometry_file` specified in the ASCII input (see section 2.12).

This page intentionally left blank.

3 Running Sierra/SD in Parallel

This section provides an example of how to perform a parallel analysis on a multiprocessor machine. There are a variety of architectures available, and unfortunately the commands differ from platform to platform. Please refer to the high performance computing website https://computing.sandia.gov for information regarding running on various platforms. There is some overhead to running in parallel versus serial. Assuming a **Sierra/SD** text input file exists and an **Exodus** file exists which contains the finite element model, the following steps are needed.

1. Decide on how many processors, *nproc*, are needed.
2. Generate a load balance file by partitioning the exodus geometry using **yada**.[7] The partitioning software can be executed on a workstation to create a load balance file. This file usually has a *.nem* extension.
3. Create your work space on the parallel machine. It is important that this work space be mounted by all the processors in the run, and that it be a fast disk system. Note that `/tmp` is usually local to each processor on distributed machines, and is therefore not a viable location for your data.
4. Move the **Sierra/SD** input file, **Exodus** file, and load balance file to your work space on the parallel machine.
5. Create an input file for **nem_spread**. Execution of **nem_spread** (on the parallel machine) with this input will create *nproc* **Exodus** files from the master **Exodus** file and move them to the locations specified in the **nem_spread** input file. On many platforms you may use the **fastspread** script to do this.
6. Use the appropriate command to run **Sierra/SD** in parallel. You may need to also run in a queue. See https://computing.sandia.gov for information on platform specific commands.
7. Use **epu** or create an input file for **nem_join** to join your results back into one **Exodus** output file.

Each step is detailed in the following paragraphs.

3.1 Number of Processors Needed

Running **Sierra/SD** in parallel requires the user to specify how many processors at a minimum are needed in order to "fit" the problem into available memory on your platform.

[7] In the past, **nem_slice** was used for this partitioning, but it is no longer recommended.

First, determine approximately how many degrees of freedom (d.o.f.) are in the model. Then, Table 1 can be used to determine the number of processors needed.

mem/core	dofs per core	**Num Cores Needed**
1GB	15,000	$dofs/15,000$
2GB	30,000	$dofs/30,000$
4GB	50,000	$dofs/50,000$
8GB	70,000	$dofs/70,000$
16GB	90,000	$dofs/90,000$

Table 1: Determining Number Of Processors Needed

The selection of the number of processors is only a guess. Optimal solutions would use nearly all the available memory on a compute node. However, it is annoyingly hard to get that right. The best measure found to date is the total number of degrees of freedom, but that measure can be significantly in error. Memory use depends on a variety of factors including the element type used, the solution strategy and even the output processing. The numbers in the table are generally conservative. Fortunately, in most cases it is not that critical unless the available memory on a compute node is exceeded.[8]

Please note that on machines with multiple cores per node, it is often better to run **Sierra/SD** with less than the maximum number of cores available per node. The reason is that when you request fewer cores, each core essentially gets more memory assigned to it.

3.2 Use "yada" to load balance the model

Sierra/SD must have data partitioned so an element-by-element computation may be performed on each processor. This approach results in scalable parallel algorithms. The"load balance" file contains the requisite information about which elements belong on which processors (or subdomains). The load balance file is generated by **yada**. For example,

workstation_prompt> `mpirun yada example.exo 500`

results in a load balance file,`example.nem`, containing information on spreading the exodus file, `example.exo`, into 500 subdomains.

The load balancing software, **yada**, is typically executed on a serial machine such as a workstation. More detailed information on **yada** is available in the following sections. Some platforms have both a graphical user interface, `yada_gui`, and a command line version of yada, (`yada_direct`). Each provides identical capability – only the user interface is different.

[8] These numbers are guidelines only. No optimization has been run on these platforms. Such would also be very problem dependent. Note that the amount of memory required is not at all linear with problem size.

3.3 Running Yada on serial UNIX platforms

On Unix and Linux platforms, **yada** is run to create a nemesis file, the decomposition, of the finite element mesh which is stored in an Exodus/Genesis file format.

The input to **yada** is an Exodus file and the number of processors for which the decomposition is needed.

```
mpirun yada example.exo 20
```

The nemesis file will be written with a .nem extension and the same base file name as the Exodus file. In the above example, a file named example.nem will be created using this command line instruction.

The preferred (default) method of running **yada** uses the decomposition tool named **chaco**. Chaco has numerous parameters that can be adjusted to improve the decomposition, either via the User_Params file or the interface in the source code for **yada** . The current defaults should produce a reasonable decomposition, and therefore, for most cases there will not be a need to change any **chaco** parameters. However, there is one parameter in the User_Params file, Check_Input, that is currently set to 'false' to improve decomposition speed. This can be changed to 'true' to help diagnose any problems that might be encountered while running **yada** . By default, the User_Params files is created in the subdirectory where **yada** is invoked. **Yada** will not create the file if one already exists. Therefore, the parameters to **yada** can be changed via this file, and the next execution of **yada** will use these parameters and not overwrite them. For further information on passing parameters to **chaco** via the User_Params file see.[57]

3.4 Parallel Machine Work Space

To run **Sierra/SD** in parallel, work space on the parallel machine is needed. Well designed parallel machines have specific disks established for fast parallel I/O. Simply choose one, and make a directory using your username, as follows.

```
cd /scratch
mkdir $USER
```

After the work space on the parallel machine is set up, move the **Sierra/SD** input file, **Exodus** file, and load balance file (e.g. *example.nem*) there.

3.5 Using Nem_spread

Each processor will reads from an individual input file. The purpose of creating a parition-ing file, say generating `example.nem` from `example.exo`, is to generate these mini-databases. The tool that creates the mini-databases is called nem_spread. Here's an example of an input file for the nem_spread.

```
Input FEM file                    = example.exo
LB file                           = example.nem
Debug                             = 4
-------------------------------------------------
                Parallel I/O section
-------------------------------------------------
Parallel Disk Info      = number=1
Parallel file location  = root=./, subdir=.
```

In this exapmle, the miniature input files are stored in the subdirectory "1" of the current working directory. The subdirectory must already exist.

To facilitate this operation a small script `fastspread` was created to perform these operations. It requires that the load balance and input FEM file have the same root name, and that the data is put into a subdirectory named "1". It will create that directory if it does not exist. For the example above, fastspread is run by typing,

```
fastspread example
```

This execution of **nem_spread** will spread *nproc* **Exodus** files onto the RAID disks specified in the input file for **nem_spread**. This location must also be specified in the **FILE** section of the **Sierra/SD** input file as follows, assuming your load balance file is *example.nem* created for 500 processors,

```
FILE
   geometry_file './1/example.par'
END
```

On some platforms `nem_spread` itself may be a parallel application and may require `mpirun` or equivalent to run it. It does not scale well, and should normally not require more than a few processors.

3.6 Sierra/SD FILE Section

As discussed above, if a load balance file *example.nem* is created for execution of **Sierra/SD** for 500 processors, and the number of raids is 1, then the **FILE** section of the **Sierra/SD**

input file must look something like the following.

```
FILE
   geometry_file "./1/example.par"
END
```

3.7 Running Sierra/SD

Once the necessary setup has been done, and a parallel **Sierra/SD** code exists in your work space, then the following can be done for example:

```
cd /scratch/$USER
mpirun -np 500 salinas example.inp
```

This will run **Sierra/SD** in parallel on 500 processors using the input file *example.inp*.[9]

In practice, only a small number of processors are available interactively on many parallel platforms. To use a larger number of processors, the queuing system must be used. Help is available on https://computing.sandia.gov. Because this is quite machine dependent, we present only a small example (using qsub), and refer the analyst to system information.

To submit a queue submission, create a small shell script, such as the following.

```
#!/bin/sh
date
cd /scratch/$USER
mpirun -np 500 salinas example.inp
date
```

The job is submitted using `qsub` with a command such as the following.

```
qsub -lT 90:00 -lP 500 -q snl.day -me run_it
```

3.8 Joining Result Files

Once the analysis run has been completed, the output exodus files will need to be recombined into a single file for visualization and processing. **epu** or **Nem_join** can accomplish this process.

[9] See Table https://computing.sandia.gov for commands appropriate to other platforms.

The easiest way to use **epu** is with the -auto flag and specifying the name of the first output results file.

```
epu -auto 1/example-out.par.500.0
```

As an alternative, the **Nem_join** input file is very similar to the **nem_spread** input file. An example input file is, e.g. *example_join.inp*.

```
Input FEM file                    = example.exo
Scalar Results FEM file           = example-out.exo
Use Scalar Mesh File              = yes
Parallel Results file base name   = example-out.par
Number of processors              = 500
Debug                             = 4
------------------------------------------------------------
                      Parallel I/O section
------------------------------------------------------------
Parallel Disk Info    =number=18
Parallel file location=root=/pfs_grande/tmp_,subdir=username
```

To run **nem_join**, do the following.

```
cd /scratch/$USER
nem_join example_join.inp
```

This will create a file *example-out.exo* in your current directory by combining all the **Exodus** output files located on the RAID disks. This is a standard **Exodus** file which may be visualized and processed using serial tools.

4 CF FETI

We have found it advantageous to maintain a stable, fixed linear solver, while continuing solver development with Charbel Farhat originally at the University of Colorado at Boulder (or CU), and now at Stanford University. However, in many respects the stable version is some sort of copy of the development code, called "CF" code.

4.1 Features of CF solver

The current CF solver has a number of features that we have not yet merged into the stable version. This may warrant use of this code for some analysis. The primary features are listed below.

Complex Solver The templated nature of this code permits solution of both real and complex systems of equations. This means that the solver may be used for direct frequency response calculations in parallel.

Contact The solver is designed to understand contact. While we have limited experience to report at this time, the solver takes the contact information and computes the response directly. In this sense, it is a nonlinear solver. It is anticipated that this methodology could permit much more efficient calculation of contact response.

Mortars The solver also directly accepts Tied Surface information (see section 2.20). Again, internal to the solver, appropriate mortar elements are constructed, and solution is performed. Use of mortar elements provides a means of consistently computing the response at an interface. Thus, mismatched meshes should still pass the patch test. We also hope to be able to better handle the large numbers of constraints that can be introduced at these surfaces.

4.2 Limitations of the Solver

There are some limitations and restrictions that should be understood in using this solver.

Parameters Some parameters are invalid, and others are added to provide the additional functionality. See the table below.

Robustness

Constraints Constraints are currently not supported in the complex portion of the solver. This will soon change. 1-27-04.

Testing While we have tested the solver on our test suite, there is much less available history at Sandia for the solver. Some testing also occurs at Stanford of course.

The following table lists parameters that are added, deleted or modified with respect to the Sandia FETI-DP solver. For the standard parameters see section 2.4 and table 28.

Table 2: CF FETI Parameter Modifications

Parameter	Change	Description
rbm_method	modify	only `geometry` accepted
outerloop_solver	add	`cg, gmres`
corner_aug_rbm_type	add	`translation, all, none`
corner_algorithm	modify	*integer*
numWaveDirections	add	*integer*
crbm_tol	add	*Real number*
weighting	add	`Topological, Stiffness`
mpc_method	add	`Dual, Primal`
mpc_submethod	add	`None, Full, PerFace, PerSub, Diag, BlockDiag`
mpc_tol	add	*Real number*
pivoting	add	`on, off`
mpc_solver	add	`skyline, sparse, spooles`
mpc_weighting	add	`topological, stiffness`
multibody	add	`0, 1 or 2`

Some of these parameters are described below.

corner_algorithm These are different from the FETI-DP parameters.

1 standard old

3 Three per neighbor

5,6,7,8 use nQ= 1, 2, 3, and 4 respectively, where nQ+2 is # of touching subdomains

numWaveDirections number of wave directions used to construct Q matrix in FETI-DPH. range 0-13, default = 3.

multibody If equal to "0" use externally defined body parameter passed in CF_Feti constructor. For multibody = "1" we force all subdomains to be treated as a single body. For multibody = "2" then use the FETI_DPC cornerMaker to find bodies. The default is "0".

Regarding multibody problems such as contact and tied surfaces, there are three alternatives for determining which body each subdomain belongs to. The default "multibody 0" is to do the body decomposition on the Sierra/SD side and pass the body id to the CF_FETI constructor. This is currently not implemented, you are just setting body = 1 in CF_FetiSolver.C for all cases which obviously won't work for multibody. However, by selecting "multibody 2" you can choose to ignore the Sierra/SD body decomposition and let the our CornerMaker algorithm do it for you. Unfortunately this algorithm doesn't always get it right for some odd cases, and although we are working

to improve it you will eventually need to implement the body decomposition on your side because our CornerMaker is CMSoft and will have to be replaced with Kendall's CornerMaker which doesn't do this. The third option is "multibody 1" which forces all subdomains to be treated as a single body.

As currently implemented, it is necessary for every body to be totally independent (i.e. not share any nodes or RBMs) with the exception that they can be connected by MPCs. So to solve a problem with a mechanism you need to split the body containing the mechanism into 2 or more bodies (creating duplicate nodes as required) and then re-tie them with MPCs. This could possibly be done as a pre-processing step on the Sierra/SD side.

This page intentionally left blank.

5 GDSW Solver Parameters for Older Version

For completeness and to help avoid potential confusion, Tables 3-4 describe the solver parameters specific to the older version of the GDSW solver. See §2.6 for remaining parameters and defaults that are common to both versions of the solver.

krylov_method Two generalized minimum residual (GMRES) methods and one preconditioned conjugate gradient (PCG) method are available. Each method is available with or without the use of store search directions (see parameters *orthog* and *orthog_option* below). The PCG method is typically faster for well conditioned problems. The right-preconditioned GMRES method is somewhat more robust with a small additional cost. We note PCG may not be numerically stable for nonsymmetric or indefinite systems of equations.

orthog_option Provides additional control over the use of stored search directions. Setting this to zero eliminates all acceleration using stored search directions.

- 0 no use of stored search directions
- 1 not used
- 2 used for acceleration of PCG
- 3 used for acceleration of GMRES

One should set *orthog_option* to 0 if convergence problems are encountered.

scale_option Is used to scale matrix entries prior to factorization. For a diagonal matrix, the scaled matrix is the identity. This option is only available for the default direct solver. Setting its value to 1 can also be used as a diagnostic tool to identify the presence of one or more matrix rows containing all zeros.

coarse_solver Is used to specify the type of direct solver used for the coarse problem. Similar direct solver options are available for subdomain interiors (*I_solver*) and overlapping subdomains (*O_solver*). The option *direct* selects the workhorse sparse Cholesky solver used in the serial version of Sierra-SD, whereas option *LDM* selects an in-house sparse solver which can be used for either symmetric or nonsymmetric matrices.

precision_option_coarse Is used to specify the numerical precision used for the coarse problem direct solver if *coarse_solver* is set to *LDM*. Similar options are available for *I_solver* and *O_solver* via the parameters *precision_option_I* and *precision_option_O*. Options available for this parameter are *double* and *single*. Use of single rather than double precision requires less memory, and can lead to improved overall performance. One should not use single precision for *precision_option_I* unless *SC_option* is set to 0. Use of single precision direct solvers in GDSW has not yet been tested thoroughly, and users are advised to contact `sierra-help@sandia.gov` with any questions.

Table 3: Solver options and defaults specific to older GDSW solver

Variable	**Values**	**Dflt**	**Description**
krylov_method	*integer*	pcg	0 - PCG,
	or		1 - right preconditioned GMRES, or *"gmres"*
	string		2 - left preconditioned GMRES, or *"lgmres"*
scale_option		0	0 - no scaling in factorizations
			1 - use scaling in factorizations
orthog_option	*integer*	2 (PCG)	0 - do not use PCG acceleration,
			2 - use PCG acceleration,
		3 (GMRES)	0 - do not use GMRES acceleration,
			3 - use GMRES acceleration
preconditioner_type	*integer*	2	1 - BDDC
			2 - GDSW
			3 - DIAG
			4 - NODAL
reduced_option	*string*	all	coarse space reduction option
			all - no reduction (corners + edges + faces)
			corners_and_edges - corners & edges only
			corners_and_edges_and_bubble - face bubble
RAS_option	*integer*	0	1 - use Restrictive Additive Schwarz
overlap_method	*integer*	0	0 - standard graph-based
			1 - element-based, e.g. for nie
LO_option	*integer*	1	0 do not use Local Overlap for subdomains
DS_block_size	*integer*	50	block size used for direct solver
coarse_solver	*string*	direct	coarse solver: either direct or LDM
I_solver	*string*	direct	Internal solver: direct or LDM
O_solver	*string*	direct	Overlap solver direct or *LDM*
local_solver	*integer*	1	sets both I_solver and O_solver at once
precision_option_coarse	*integer*	0	0-double: double precision for LDM solver
precision_option_I		0	1 - single precision for LDM solver
precision_option_O		0	
default_precision_opt	*integer*	0	0 - double
			1 - single
			(only available for LDM or Pardiso direct solvers)
ML_max_level	*integer*	7	maximum number of levels for multilevel local solver
ML_max_coarse	*integer*	1000	maximum number of unknowns for coarsest level
ML_print_coarse	*integer*	0	0-no/1-yes: print coarse stiffness matrix
ML_print_Phi	*integer*	0	0-no/1-yes: print interpolation matrix
pardiso_message_level	*integer*	0	0-no/1-yes: print messages
prt_matrix	*integer*	0	0 - no output
			1 - print out matrix in 3-column format
			2 - print out matrix in CSR format
reduced_option_coarse	*integer*	3	same as reduced_option but for subregions
prt_debug	*integer*	0	0-no/1-yes: print debug output

Table 4: Solver options and defaults specific to older GDSW Solver

Variable	**Values**	**Dflt**	**Description**
enable_multilevel_coarse_solver	N/A	N/A	Use GDSW to solve coarse problem.
parmetis_option	*integer*	0	parmetis option for coarse problem partitioning: 0 - PartKway 1 - PartGeomKway
coarse_overlap	*integer*	1	overlap parameter for coarse problem
coarsening_ratio	*integer*	0	coarsening ration for coarse problem
num_sub_per_proc	*integer*	1	number of subdomains per processor
num_iter_improve_I	*integer*	0	number of iterative improvement steps for I_solver = LDM and precision_option_I = single
num_iter_improve_O	*integer*	0	number of iterative improvement steps for O_solver = LDM and precision_option_I = single
num_iter_improve_coarse	*integer*	0	number of iterative improvement steps for coarse_solver = LDM and precision_option_coarse = single
enable_recycle	*integer*	0	0 - do not use alternative recycling algorithm 1 - use alternative recycling algorithm
enable_belos	*integer*	0	0 - do not use Belos 1 - use Belos
belos_num_blocks	*integer*	-1	number of search vectors for a Belos cycle
belos_num_recycle	*integer*	-1	number of search vectors for recycling (belos_num_recycle should be less than belos_num_blocks)
num_recycle	*integer*	-1	number of vectors when restarting recycling (less than max_N_orthog_vecs, set to -1 to not restart recycling)
max_recycle_update	*integer*	-1	maximum number of times recycle space is updated

reduced_option Is used to specify a reduction strategy for the coarse problem size. There is no need to consider this parameter for problems run on less than a few hundred processors. However, as the number of processors (subdomains) becomes large, solving the coarse problem can become a bottleneck. The default (*all*) is to do no coarse problem reduction. Option *corners_and_edges* eliminates all coarse face degrees of freedom. The option *corners_and_edges_and_bubble* is appropriate for problems with nearly incompressible materials. Reduced coarse problem options have not yet been tested thoroughly, and users are advised to contact `sierra-help@sandia.gov`.

enable_multilevel_coarse_solver Tells GDSW to recursively use the same algorithm for the coarse problem as is used for the original problem. Without this option, the coarse problem is solved with a direct solver on one processor and can run into memory limitations for large problems with a lot of subdomains. Use of this option may adversely affect run time performance, so it is not generally recommended for small to medium size problems where the direct solver is sufficient.

enable_recycle Is used to activate a new method for reusing (recycling) previously stored search directions. By default, this new method uses two orthogonalization steps to help reduce the effects of round off errors.

Index

+/-, 240
--aprepro, 4
--define, 4
#include, 351

acceleration, 116, 173
accelX, 141
accelY, 141
accelZ, 141
Acknowledgments, 346
acoustic, 200
    point source, 152
acoustic_accel, 149, 152–154, 331, 332
acoustic_vel, 149, 152–154, 331
AcousticFraction, 52
adagio, 56
adiag, 129
AEigen, 27
AllStructural, 94
alpha, 244
ANASAZI, 46
Anasazi, 27
anasazi, 36
anblocksize, 27
aneigen_tol, 27
angular_acceleration, 165
angular_velocity, 165
anisotropic, 194, 195
Anisotropy, 352
anverbosity, 27, 28
apartvel, 126
aprepro, 4, 5
apressure, 126
ARPACK, 23, 87, 124, 346
Attune, 240
autolayer, 265
autospc, 89

badqual_limit, 86, 90
BAR, 275
BEAM, 275
Beam2, 114, **266**, 266–270, 273, 274
bending_factor, 252, 260
beta, 244, 246
blk_eigen, 28, 247
blkalpha, 189, 191
blkbeta, 189, 191, 329
BLOCK, 3, 187, 191, 193, 327, 329
Block, 21, **187**, 253, 302
    General Parameters, 189
    Material Id, 327
block, 109, 111, 133, 311
body, 148
BOUNDARY, 145
Boundary, 40, **138**, 138, 338
boundary, 151
buckling, 30

case, 8
CBModel, 10, 12, 229, 231, 233
CBR, 10
    null space correction, 11
cbr, 310
ceig, 50
Ceigen, 46
ceigen, 36
center node, 178
Centrifugal, 164
Centripetal, 164
CF, 82
CF_FETI, 81, 373, 374
checkout, 9
CheckSMatrix, 38, 39
Cij, 195
Citations, 347
CJdamp, 9, 203
CJetaFunction, 10, 203
CLOP, 100
Clop, 100
    Parameters, 100
CMS, 10
color_domains, 339
Command Line
    parallel execution, 367

serial unix, 365
comment
entire section, 4
comments, 3
complex, 35, 36
complex load, 164
Component Mode Synthesis, 10
condition_limit, 86, 89, 90
ConMass, **275**, 275–277
ConMassA, 276
consistent loads, 159
consistent mass, 83
constant_vector, 235
constrain_rbms, 168, 169
constraint_correction, 93
constraintmethod, 83
constraints
orthogonalization, 93
Contact, **173**
contact
normal, 176
coordinate, 132–134, 138, 148, 165, 189, 191, **203**, 203, 257
Coordinate Frame
Tied Joint, 182
Coriolis, 164
corner
algorithm - table, 98
algorithms, 97
augmentation - table, 98
parameter - table, 98
selection, 97
corner nodes, 97, 99
corner.data, 338
correction, 11
Craig-Bampton Reduction, 10
Cubit, 335

Damper, 280
viscous, 280
damper, 247, 292
cubic, 292
Damping, 39, **244**, 244
Block, 190
Dashpot, **280**
dashpot, 292
Data Truth Table, 324
data_file, 19, 64, 326, 327
data_truth_table, 19, 21, 64, 324
datafile, 228, 229
dataline, 228
ddamout, 130
Dead, **312**, 312
dead, 312
Decomposition, 367
delta, 229
density, 201
diagnostics, 335
adiag, 129
beams, 128
cubit, 335
grope, 335
kdiag, 127
yada, 336
zero energy modes, 338
dimension, 172, 228
Direct FRF Example, 362
directfrf, 16, 34, 146, 330, 362
directfrf-loadid, 19, 319
directfrf-materialid, 21
disp, 115
disp0, 141
Displacement, 146
displacement, prescribed, 140
Displacement1, 146
distribution
Tied Joint, 184
dmax, 283
dmax,kmax, 283
dump, 22

E, 194
Eagle, 56
ECHO, 108, 109, 111, 118, 127, 137, 351
Echo, **108**, 260
echo, 40, 132
eforce, 109, 121
eig_tol, 48, 87
Eigen
direct solution, 25

eigen, 22, 27–29, 34, 43, 330
eigen tolerance, 87
eigen-materialid, 25
eigen_norm, 92
Eigenanalysis
  element checks, 114
  example, 351
  quadratic, 49, 52
  structural acoustics (modal basis), 49, 52
eigenk, 29
eigenvector
  normalization, 92
elastic-plastic, 292
electrostatic, 275
ElemEigChecks, 114
Element
  Beam2, 266
  ConMass, 275
  Dashpot, 280
  Dead, 312
  eigenvalue checks, 114
  Force output, 121
  Ftruss, 274
  Gap, 293
  Gap2D, 297
  GasDmp, 297
  Hex20, 250
  Hex8, 249
  HexShell, 262
  Hys, 281
  InterfaceElement, 312
  Joint2G, 285
    Property, 286, 290–293
  Mass, 275
  mortar, 373
  Nbeam, 270
  Nmount, 299
  Nquad, 255
  Ntria, 255
  OBeam, 274
  Offset Shells, 261
  orientation, 124
  Quad8T, 254
  QuadM, 252
  QuadT, 251
  Rigid
    RBar, 303
    RBE2, 304
    RBE3, 304
    RRod, 302
  RSpring, 278
  Shys, 283
  Spring, 277
  Spring3, 279
  SpringDashpot, 281
  Stress/Strain, 314
  Superelement, 306
  Tet10, 251
  Tet4, 251
  Tria3, 260
  Tria6, 261
  TriaShell, 256
  Truss, 274
  Truth Table, 314
  Wedge15, 250
  Wedge6, 250
Element Truth Table, 314, 315
Elemqualchecks, 114
end, 3
energy, 118
energy_exo_var, 91, 158
energy_load, 150, 158
energy_time_step, 91, 158
enforced acceleration, 141
  random vib, 173
enforced displacement, 140
engineering units, 87
EOrient, 117
eorient, 124, 262, 316
eplas, 292
equilibrium, 57
error metrics, 120
ErrorNorm
  SA_eigen, 52
Euler Force, 164
Example
  Anisotropy, 352
  Direct FRF, 362
  Eigen, 351
  Modal FRF, 360

Modal Transient, 358
multiple materials, 354
Statics, 364
exo_var, 215–217
exodus, 314
input, 367
results, 367, 371, 372
exodus precision, 129
Exodus Read Functions, 160
explicit, 67
extraNodes.dat, 97

faa, 114
Farhat, Charbel, 346
fastspread, 367, 370
FEI
memory usage, 341
Felippa, Carlos, 346
FETI, 81, **95**, 338
CF specific parameters, 374
CF Version, 373
coarse solver, 342
corner nodes, 97, 99
diagnostics, 99
global rigid body modes, 344
local rigid body modes, 343
local solver, 342
multiple right-hand-sides, 343
options affecting memory, 343
orthogonalization vectors, 343
parameters example, 341
preconditioner, 342
Tutorial, 341
FETI-DP, 97
FieldTime, 146
FILE, 66, 113, 135, 136, 370
File, **135**
file, 230
FilterRbm, 245
FilterRbmLoad, 70, 91, 94, 168
finite_difference, 235
fixed, 138, 140
fluidloading, 22, 23
flush, 42, 55, 61, 133
fmax, 283
follower, 149, 150
follower stiffness, 151
force, 121
qmodal, 45
forces, 124
format, 230, 235
freq_max, 18, 36, 39, 42, 59
freq_min, 18, 36, 39, 42, 59
freq_step, 18, 36, 39, 42, 59, 60
FREQUENCY, 326
Frequency, **134**
frequency, 18, 36, 39, 40, 42, 59, 135
FSI function, 224
Ftruss, 274, 275
FUNCTION, 216
function, 39, 42, 140, 153, 160, 172, **205**, 205, 229
FSI, 224
linear, 207
loglog, 210
plane wave, 219
polynomial, 209
random, 210
random library, 212
rtc, 217
shock wave, 222
spherical_wave, 220
stepwave, 220
table, 208
user defined, 217

G, 194
G_bounds, 328
Gap, **293**, 293, 297
ellipsoidal, 297
Joint2G, 291
gap, 291
gap removal, 174
Gap2D, **297**, 297
gap_removal, 75, 76
GasDmp, **297**
GasDmp , 297, 298
GDSW, 102, 168
Older Version, 377
Parameters, 102

Gemini, 63, 69
Generalized Alpha integrator, 61
GEnergies, 118
geometry_file, 57, 66, 135, **136**, 338
Gim_bounds, 328
global variables, 123
GlobalSolution, 230
Greal_bounds, 328
Grope, 335

harwellboeing, 120
Heterogeneous, 328
heterogeneous, 327, 328
Hex20, **250**
Hex8, **249**, 249, 250
Hex8b, 249
Hex8F, 250
Hex8u, 249
HexShell, **262**, 263
    Mass, 266
History, 130, 231
History Files, **130**
Homogeneous, 327
homogeneous, 327, 328
hydrostatic balance, 71
Hys, **281**, 281, 283, 284
Hysteresis element
    cubic, 281

I1, 267
I2, 267
iforce, 164
ignore_gap_inversion, 95
igravity, 164
imaginary_data_file, 326
imoment, 164
impedance_pressure, 144
impedance_shear, 144
implicit_restarts, 27
include, 5
Inertia Tensor, 10
inertia_matrix, 230
infinite element, 144
Info, 88
INITIAL-CONDITIONS, 170
initial-conditions, **170**
initial_time_step, 68
Integrator, 61
InterfaceElement, **312**, 312
interp, 215, 217
Inverse Data File, 323
Inverse Problem, 323
inverse-problem, 19, 21, 64, 323
inverse_load_type, 331
inverse_material_type, 327
Invoking Sierra/SD, 365, 367
ipressure, 164
isotropic, 194, 256
isotropic_viscoelastic, 194
iterations, 239
itraction, 164
Iwan, 285, 289
iwan, 247

Johnson, Conor, 9
joining files, 371
Joint2G, 182, 186, **285**, 285, 286, 291, 293
joint2g, 329
joint_truth_table, 329

K, 194
K_bounds, 328
kaa, 114
kdiag, 127, 339
keepmodes, 38
Kim_bounds, 328
kmax, 283
kmin, 283
known, 327, 328, 331
Kreal_bounds, 328

Lagrange, 83
Largest_Ev, 29
layer, 257
layered material, 257
lfcutoff, 35, 36, 38, 42
linedata_only, 66
LINESAMPLE, 147
linesample, 66, 137, 146
LinkStiffness, 87
load, 8, 39, 42, 43, **169**, 169, 170, 172

complex, 164
consistent, 159
randompressure, 160
statics, 159
transient, 160
load balance, 368
Load Identification
DirectFrf, 19
Limitations, 332
Load entry, 331
Transient, 64
LOADS, 19, 64, 166, 331
loads, 8, 30, 40, 43, 140, 141, **147**, 147, 151, 160, 169, 170, 172
loglog, 210
lumped, 83, 354
lumped mass, 83
lumped_consistent, 83

maa, 113
Macroblock, **191**, 193, 285
Martinez, David, 346
mass, 109, 111
blockwise properties, 109
consistent, 83
lumped, 83
non-structural, 191
properties, 109
mass=block, 111
MATERIAL, 328, 329
Material, 21, **193**, 329
acoustic, 200
anisotropic, 195
Anisotropic example, 352
density, 201
isotropic, 194
layers, 257
orthotropic, 195
stochastic, 196
temperature dependent, 200
temperature function, 200
viscoelastic, 197
Material Identification
directfrf, 21
Eigen, 25
Block entry, 327
Material entry, 328
Matlab, 146
Matrix
file names, 122
output in mfile format, 122
RanLoads parameter, 172
matrix, 172
Matrix-Function, **224**
matrix-function, 39, 172
MatrixFloor, 92
max_newton_iterations, 54, 55
MaxMpcEntries, 92
maxRatioFlexibleRbm, 246
MaxResidual, 87
meff, 31
membrane_factor, 260
memory, 97, 341–343
diagnostics, 343
memory.data, 343
memory diagnostics, 338
memusage, 112
mesh discretization error, 120
mesh_error, 120
method, 174
Metis, 346
mfile, 121
MFile_Format, 93
MinimumNodalSpacing, 163
mksuper, 307
modal acceleration, 34
modal amplitude, 111
Modal Effective Mass, 31
Modal FRF example, 360
Modal Participation Factor, 31
Modal Transient Example, 358
modal_amp, 122
modal_data_file, 25, 327
modal_weight_table, 25, 327
ModalFilter, 11, 12, 22, 23, 236
ModalFraction, 52
modalfrf, 18, 34, 35, 244, 360
ModalFv, 122
modalranvib, 37, 39, 244
modalshock, 41

modaltrans, 191
modaltransient, 42–45, 69, 79, 244, 358
ModalVars, 111
Model Reduction, 10
mortar, 175
Mortar method, 373
mortar methods, 174
MortarMethod=dual, 175
MortarMethod=standard, 175
MPC, **299**, 299, 301
mpc, 111
Mpc_Scale_Factor, 92
MPF, 32

name, 194, 196
nastran
    output4 in CBR, 230
Nbeam, 267, **270**, 270–273
ncdfout, 230
negative
    element matrices, 92
NegEigen, 85
neglect_mass, 146
nem_join, 367, **371**
nem_slice, 336, 367, 368
nem_spread, 339, **367**, 370
NERSC, 82
netcdf, 230, 309
Newmark-Beta, 61
newmark_beta, 62
Ng, Esmond, 82, 346
NLresiduals, 110
NLstatics, 53
NLtransient, 55
nmodes, 11, 22, 23, 27, 28, 30, 35, 38, 41, 43, 48, 58, 236, 244
Nmount, **299**, 299
    stability, 299
no_geom_stiff, 57
no_symmetrize_struc_acous, 84
node_list_file, 58
NodeListFile, 132, 142, 338
nodes, 109, 132
nodes none mesh, 109
nodeset, 132, 148, 178, 216, 230
nominalt, 226
non-structural mass, 191
none, 109
none nodes, 109
nonlinear, 189, 191
nonlinear_default, 87, 88, 189
Normal
    Tied Joint, 183
normal
    shells, 176
normalization
    eigenvector, 92
normalstiffness, 312
noSVD, 38
NPressure, 125
Nquad, **255**, 255
Nquad/Ntria, 255, 257
nquad_eps_max, 255
nrbms, 35, 36
nskip, 41, 42, 55, 59, 61, 68, 133
nsm, 189, 191
nsteps, 41, 42, 55, 59, 61
Ntria, **255**, 255
nu, 194
null space correction, 11
num_newton_load_steps, 54–56
num_rigid_mode, 94, 168, 169

OBeam, 114, **274**
off, 114
Offset Elements, 313
Offset Shells, **261**
old_transient, 60, 239, 242
OldBeam, 87
on, 114, 115
Optimization, 319
optimization, 319
origin, 229
orthogonalization
    constraints, 93
orthotropic, 194, 195
orthotropic_layer, 196
orthotropic_prop, 194, 195
OTM, 232
OTME, 232

OutElemMap, 232
OutMap, 232
OUTPUT, 113
output, 18, 36, 90, 132
OUTPUTS, 113–118, 120, 121, 124–127, 129
Outputs, **113**, 231
    Slave_Constraint_Info, 126
outputs, 40, 113

P, 138
p0, 141
Padé, 18
parallel, 367
parameter, 85
PARAMETERS, 157, 303
    Info, 88
    syntax_checking, 89
Parameters, **85**
parameters, 103, 104, 168
patch
    negative element matrices, 92
Pdot, 141
plane_wave function, 219
plastic, 292
point_volume_accel, 152–154
point_volume_vel, 152–154
power spectral density, 40
precision, 129
preddam, 12, 23
prescribed acceleration, 141
prescribed displacement, 140
pressure, 125, 149, 152
    depth dependent, 159
    nodal, 125
pressure_z, 159
Presto, 137
presto, 56
Problematic Elements, 339
Problematic Subdomains, 339
Processor Count, 367
Projection_eigen, 52
projection_eigen, 46, 47
Property, 286, 290–293
prt_debug, 58, 99, 338, 343
prt_rbm, 100
prt_summary, 100
PSD, 40, 173

QEVP, 45
qevp, 34, 36
qmodalfrf, 34, 53
qmodaltransient, 44, 45, 53, 69, 79
Quad8T, **254**, 254, 261
QuadM, **252**, 252–254
quadratic eigenvalue
    comparison, 45
QuadT, **251**, 251–253

Raghaven, Padma, 346
RAID disks, 370
Random Number Generator, 94
randomlib, 215–217
RandomLib functions, 212
RandomPressure, 161
RanLoads, 39, **171**, 171, 172
ratiofun, 245
rational function, 18
RBAR, 90, 180, 303, 304
RBar, **303**
RBE2, 304
RBE3, **304**, 304–306
RBM, 338, **343**, 343
RbmDof, 11
RbmTolerance, 91
read_from_file, 57
ReadNodal, 155
readnodal, 215, 216
ReadNodal functions, 215
ReadNodalSet, 216
ReadNodalSet functions, 216
ReadSurface, 155, 216, 217
ReadSurface functions, 216
real_data_file, 324, 326
Receive_Sierra_Data, 56
receive_time_step, 57
REFC, 305
References, 347
regularization, 323
reorder_rbar, 90
reorthogonalization, 37

residual, **123**
    global var, 123
    non-linear norm, 110
    vector, 110, 123
residual work, 123
Restart
    solution support, 79
restart, 77
Rho, 62
rho, 55, 61–63, 332
rhs, 121
Rigid Body Filter, 168
RigidSet, 181
rigidset, **178**, 178
    limitations, 180
rigidsets, 178
RMS, 37, 118
Rod *see Truss*, 274
ROL Output, 333
RotaccelX, 141
RotaccelY, 141
RotaccelZ, 141
rotate, 257
Rotational Frames, 164
rotational_type, 190
RotX, 138
RotY, 138
RotZ, 138
rowfirst, 229
RROD, **302**, 302
RrodSet, 181
RSpring, 114, 277, **278**, 278
RTC, 274
rtcfile, 218
Run time compiler, 217, 274
Running
    parallel, 367
    serial, 365

S_isotropic, 194, 196
SA_eigen, 46, 49, 50
    ErrorNorm, 52
    limitations, 50
sa_eigen, 36, 47
Salinas, 1
SamplingRandom functions, 211
scalar, 217
scale, 153, 160
scaling
    loads, 160
    PSD, 173
scattering, 83
sd_factor, 249, 250, 252, 253
Section Commands
    Loads Rigid Body Filter, 168
    Block, 187
    Block Parameters, 187
    Boundary, 138
    Contact Data, 173
    Coordinate, 203
    Damping, 244
    Echo, 108
    FETI, 95
    File, 135
    Frequency, 134
    Function, 205
    History, 130
    Initial-Conditions, 170
    Load, 169
    Loads, 147
    Macroblock, 191
    Material, 193
    Matrix-Function, 224
    ModalFilter, 236
    Outputs, 113
    Parameters, 85
    RanLoads, 171
    RrodSet, 181
    Sensitivity, 238
    Solution, 6
    solver_options, 108
    Table, 228
    Tied Surfaces, 173
SENSITIVITY, 233, 238, 242
Sensitivity Analysis, **238**, 351
sensitivity_method, 233
sensitivity_param, 310, 311
set
    rigid, 178
Shear

Tied Joint, 184
shear_axis, 182, 183, 287
Shells
Offset, 261
shift, 11, 22, 25, 27, 28, 30, 31, 43, 58
shock_wave function, 222
Shys, **283**, 283, 285
shys, 290
sideset, 133, 148, 149, 178, 217, 230, 262
sierra, 137
Sierra data, 56
Sierra Transfer, 63
sierra_input_file, 135, 137
size, 228
SkipMpcTouch, 89
Slave_Constraint_Info, 126
slosh, 144
Smallwood, D. O., 290
smoothing parameters, 176
SOLUTION, 25, 31, 54, 58, 83, 354
Solution, 3, **6**, 76
Buckling, 30
CBR, 10
ceigen, 48
checkout, 9
CJdamp, 9
complex eigen, 48
direct frequency response, 16
Eigen, 22
Eigen of stiffness, 29
Eigen of subdomain, 58
linear transient dynamics, 60
matrix output, 22
modal frequency response, 34
modal random vibration, 37
modal transient response, 42
Multicase, 6
Parameter Table, 8
Parameters, 6
Multicase Time Stepping, 9
nonlinear statics, 53
nonlinear transient dynamics, 55
Options, **76**
constraint method, 83
lumped mass, 83
no_symmetrize_struc_acous, 84
restart, 77
scattering, 83
solver, 80
qmodal frequency response, 53
qmodal transient response, 44
Receive_Sierra_Data, 56
shock response spectra from modes, 41
shock response spectra from transients, 59
statics, 58
Table of Arguments, 7
tangent stiffness matrix update, 59
thermal structural response preload, 65
solution, 6, 37, 39
solver, 80
parameters, 100, 102
solver_options, 102, 108, 168
Sparsepak, 82
sparspak, 82
SpatialBC functions, 214
spatially_constant, 331
Specific Heat, 201
SPHERE, 276
spherical_wave, 220
spherical_wave function, 220
spreading files, 367, 370
Spring, 114, **277**, 277, 278
cubic, 279
Linear, 277
Parameter Values, **278**
Rotational, 278
Spring3, 114, **279**, 279
SpringDashpot, **281**, 281
srs_damp, 41, 42, 59, 60
Stanford, 373
start_time, 42, 55, 61
static, 155
statics, 58, 364
Statics Example, 364
step size, 61
step_wave function, 220
strain, 116
Stress
Gauss Point, 118

stress, 116, 117
Stress = GP, 118
Stress/Strain Recovery, **314**
Structural Acoustics
    eigen, 49, 52
StructuralFraction, 52
subdomain
    output, 112
subdomain_eigen, 58
subdomains, 137
subspace_size, 27, 28
sum, 114, 115
Superelement, **306**, 306
    parameters, 310
superelement, 233, 307, 310
SuperLU, 82, 346
surface
    Tied Joint, 184
syntax_checking, 89

Table, 226, **228**, 228
table, 208
tablename, 209
tangent, 6, 59, 166
tangentialstiffness, 312
TangentMethod, 88
tcoord, 262, 263
termination_time, 67, 68
Tet10, **251**, 251
Tet4, **251**, 251
thermal_exo_var, 91, 157, 158
thermal_load, 90, 150, 155, 157, 158
thermal_time_step, 90, 157, 158
TIED DATA, 75, 76, 174, 184
Tied Joint
    Coordinate Frame, 182
tied node, 178
Tied Surfaces, **173**
Time Integrator, 61
time step, 61
time_step, 41, 42, 55, 59, 61
time_step_estimation, 68
time_step_increase_factor, 68
time_step_scale_factor, 68
timing, 112
TIndex, 123
tolerance, 54, 55, 239
traction, 149
transfer, 61, 63, 84
Transform, 83
transhock, 59
transient, 45, 55, 60, 64, 79, 155
transient-loadid, 64, 319
TRANSVERSE, 174, 175
Tria3, 251, 254, 256, **260**, 260, 261
Tria6, 254, **261**, 261
TriaShell, 251, 254, **256**, 256, 257, 260, 316
Troubleshooting, 335
troubleshooting
    cubit, 335
    grope, 335
    yada, 336
    zero energy modes, 338
TruncationMethod, 38
TRUSS, 276
Truss, 114, **274**, 274
tsr_preload, 65, 66

units of measure, 85, 87
Univ. Colo, 373
Univ. Colo, 346
Univ. Minn, 346
untilfreq, 22, 23
update_step_interval, 68
update_tangent, 54–56, 294
UseAnalystNodeMap, 95
usemodalaccel, 35, 53
User functions, 217

values, 239
vector, 217
vectors, 239
vectors none, 239
vel, 217
vel0, 141
velocity, 116
velx, 217
vely, 217
velz, 217
viscofreq, 48, 49

viscous damper, 280
volume_acceleration, 215
VonMises, 117
vrms, 40, 118

warninglevel, 129
waterline, 71
Wedge15, **250**
Wedge6, **250**, 250
wet modes, 23
work, 118
WtMass, 85, 109
wtmass, 173

X, 138

Y, 138
yada, 336, 339, 367–369
yes, 146

Z, 138
ZEM, 98, 338

Sandia National Laboratories