SAND2026-15786

SANDIA REPORT

Printed January 9, 2026

Sandia National Laboratories

Sierra/SD – User's Manual – 5.28

Sierra Structural Dynamics Development Team

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

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 user's 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.

This page left blank

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page intentionally left blank.

ACKNOWLEDGMENTS

The **Sierra/SD** software package is the collective effort of many individuals and teams. A core Sandia National Laboratories based **Sierra/SD** development team is responsible for maintenance of documentation, testing, and support of code capabilities. This team includes Dagny Beale, Gregory Bunting, David Day, Clark Dohrmann, Payton Lindsay, Justin Pepe, Julia Plews, Jesse Thomas, and Ben Treweek.

The **Sierra/SD** team also works closely with the Sierra Inverse and Plato teams to jointly enhance and maintain several capabilities. This includes contributions from Wilkins Aquino, Mark Chen, Sean Hardesty, Elizabeth Livingston, Clay Sanders, Chandler Smith, Adam Sokolow, Timothy Walsh, and Ray Wildman.

The **Sierra/SD** team works closely with other Sierra teams on core libraries and shared tools. This includes the DevOps, Sierra Toolkit, Solid Mechanics, Fluid Thermal Teams. Additionally, analysts regularly provide code capabilities as well as help review and verify code capabilities, testing, and documentation. Other individuals not already mentioned directly contributing to the **Sierra/SD** documentation, testing, and code base during the last year include Victor Brunini, Jonathan Clausen, Nathan Crane, Jared Crean, David Glaze, Mark Hamilton, Jacob Healy, Andrew Kimler, Dong Lee, Kevin Manktelow, Scott Miller, Matthew Mosby, Tony Nguyen, Tolu Okusanya, Heather Pacella, Malachi Phillips, Kendall Pierson, Nick Reynolds, Philip Sakievich, Timothy Shelton, John Shimanek, Greg Sjaardema, Clinton Stimpson, Tyler Voskuilen, Ellen Wagman, Alan Williams, and Christopher Wilson.

Historically dozens of other Sandia staff, students, and external collaborators have also contributed to the **Sierra/SD** product and its documentation.

Many other individuals groups have contributed either directly or indirectly to the success of the **Sierra/SD** product. These include but are not limited to;

- Garth Reese implemented the original **Sierra/SD** code base. He served as principal investigator and product owner for **Sierra/SD** for over twenty years. His efforts and contributions led to much of the current success of **Sierra/SD**.
- The ASC program at the DOE which funded the initial **Sierra/SD** (Salinas) development as well as the ASC program which still provides the bulk of ongoing development support.
- Line managers at Sandia Labs who supported this effort. Special recognition is extended to David Martinez who helped establish the effort.
- 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 linear solvers.

- 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.
- Danny Sorensen, Rich Lehoucq and other developers of ARPACK, which is used for eigenvalue problems.
- Esmond Ng who wrote *sparspak* for us. This sparse solver package is responsible for much of the performance in **Sierra/SD** linear solvers.
- 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.
- Padma Raghaven for development of a parallel direct solver that is a part of the linear solvers.
- The developers of the SuperLU Dist parallel sparse direct linear solver. It is used through GDSW for a variety of problems.
- Leszek Demkowicz at the University of Texas at Austin who provided the HP3D[24] library and has worked with the **Sierra/SD** team on several initiatives. The HP3D library is used to calculate shape functions for higher order elements.

This work was supported by the Laboratory Directed Research and Development (LDRD) program.

1. INTRODUCTION

This document covers the use of **Sierra/SD**.

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.

Sierra/SD is similar to commercial codes like NASTRAN or Abaqus. It has some nonlinear capability, but excels in linear computation. It is different than the above commercial codes in that it is designed to operate efficiently in a massively parallel environment.

This page intentionally left blank.

2. RELEASE NOTES

The sections in this chapter mainly describe new features, bug fixes, performance improvements, and features deprecated or removed in each new version of **Sierra/SD**. As a preface to this information, an explanation of the feature deprecation schedule and process in **Sierra/SD** is presented in the following section.

2.1. Feature deprecation procedure

From time to time, the **Sierra/SD** development team may determine that a feature should be deprecated. This might happen if a certain newly developed capability, workflow, or input syntax is preferred over an existing one. It may also occur when a capability is determined by the development team to be unused by the analyst community, and thus it does not need to be maintained any longer.

When a feature is selected for deprecation, **Sierra/SD** will issue a clear warning message in the log file with planned date and code version for full deprecation. These release notes will also be kept up-to-date with newly planned feature deprecations and their respective planned full deprecation dates and **Sierra/SD** versions.

Full feature deprecation is based on the fiscal year schedule. Features marked for deprecation between October and September of a given year will be fully deprecated in the first release of the following fiscal year, i.e., the first release that falls on or after the first day in October. For example,

- A feature marked deprecated in April 2023 is fully deprecated in the first release occurring on or after October 1, 2024.
- Features marked deprecated in September 2023 are fully deprecated in the first release on or after October 1, 2024.
- A feature marked deprecated in October 2023, is fully deprecated in the first release on or after October 1, 2025.

Full deprecation of a given feature is subjective, but is described in detail in Table 2-1.

Table 2-1. – Description of full feature deprecation.

The feature will . . .	**Sierra/SD** will output . . .	Rationale
be disabled completely	fatal error, log file message	if there is reasonable skepticism of the accuracy or reliability of the feature or if there are hazardous side effects from using the feature or if it is unsafe to ignore input syntax, or if the code team wishes to no longer maintain the feature.
do nothing	log file warning	if the input syntax can be safely ignored.
remain functional	log file warning	if no harmful side effects are identified in the code. if little or no effort is required for the code team to maintain the feature.

2.2. Release 5.28

New or Improved Features

- Restart is now enabled for eigen solutions with residual vectors. See Section 5.8.7.1 for details.
- Element `centroid nearest location` and `at element` `HISTORY` outputs now support any element keyword. Previously, only stress, strain, and von Mises stress were supported. See section 9.2.2 for more information.
- Users may now output the sum of nodal and element variables as a user-defined global field using the following syntax: `compute global <string>Name as sum of element|nodal <string>Variable`. See sections 9.5.1 and 9.5.2 for more information.
- Closest distance calculations can now utilize block skinning to improve performance relative to full block comparisons. See section 9.5.3.
- Linear frequency specification using `freq_min/freq_step/NF` is now enabled. See section 9.3 for more information.
- User Outputs now have a "transform" capability that allows a user to rotate variables like stress for output. See section 9.5.5
- **Sierra/SD** now supports SM-style input syntax for viscoelastic swanson material model. See input 6.4 for an example.
- GDSW options to control the tacho solver settings have been updated. Tacho is the default solver on gpu platforms. See table 4-18 for details.
- Tensor and vector expression variables may now be read in as expression variables for analytic functions.
- The use of multiple loads in DirectFRF (section 5.6.1) no longer requires a direct solver (e.g. `useParallelDirectSolver = yes` in the `GDSW` section). However, use of a direct solver will still give the best performance when not memory constrained.

Behavior Changes

- The previously-supported (but undocumented) keyword `eorientation` has been deprecated. Use the keyword `eorient` to request element orientation vectors (section 9.8.4).
- The keyword to request user-defined the material direction has been consolidated to simply `material_direction`. The old keywords (`material_direction_1`, `material_direction_2`, `material_direction_3`) have been deprecated. See section 9.8.11 for more details on this capability.
- Rigidset names (section 7.36) are no longer required.

- Linear frequency specification using `freq_min/freq_step/freq_max` has changed slightly. Previously, the final frequency was the nearest multiple of F_{step} to F_{max} (above or below). Now, the nearest multiple *greater* than F_{max} is used. See section 9.3 for more details.

Production-Ready Features

The following features have been moved from beta to production, and no longer require the `--beta` command-line flag in **Sierra/SD**.

- Rigidset block definitions (section 7.36).

Bug Fixes

- Linear frequency specification using `freq_min/NF/freq_max` was previously incorrect. See section 9.3 for the new behavior.
- A bug in the superposition solution case (section 5.25) was fixed where acceleration/velocity could be 0 if pre-requisite output requests were not present. Now, displacement, velocity, and acceleration can be output independently, and displacement is no longer output by default when no outputs are requested.
- An error in output of stress and strain at specific locations of Tet10 elements has been fixed. This effected history output at location, line sample output, and some transfer operations. Previous computed values were interpolated to an incorrect point well outside the bounds of the element and could be in error by orders of magnitude.

2.3. Release 5.26

New or Improved Features

- **Sierra/SD** 5.26 officially supports ATS-4 platforms El Capitan and El Dorado.
- Elements with local coordinate frames can now use `coordinate from_geometry` in the `block` section to infer the coordinate system from the geometric orientation. See Section 6.6.1.3 for additional details.
- Contact cutoff variables have been improved to support more versatile options for **Sierra/SD** analyses.
  - Cutoff variables may be specified for each contact surface interaction within a `begin contact definition` section.
  - Cutoff variables may also now be specified within the `interaction defaults` section.

  - The `cutoff_variable` option has been removed from the `begin contact definition` (Dash contact) options section.

  Consult section 10.2.1 for more information.

- Craig–Bampton reduction (CBR) now supports the `nmodes = all` and `untilfreq` options to solve for fixed interface modes. See section 5.3 for more information.
- `MFile` output of constraint matrices was added. See section 9.7.34 for details.
- When restarting, **Sierra/SD** now checks for consistency between the restart input file and the input file used to generate the restart database. This behavior may be customized using the new `restart_input_checking` parameter. See section 4.4.1 for more information.
- The `transfer` capability has been significantly refactored and improved for better-quality results. A new option `mismatch mesh behavior` also now exists to help users identify transfer errors due to non-matching or misaligned sending and receiving mesh domains. See section 4.3.2 for more information.
- The `mode` option in the `DAMPING` section can now handle single integers, sequences, and ranges of modes to apply the same damping parameters across multiple modes.
- Spring stiffness parameters may now be defined as the output of a function. See section 7.17 for more details.
- Material direction output (section 9.8.11), previously only available for three-dimensional element types, is now output for one- and two-dimensional elements as well.

Behavior Changes

- The behavior when specifying a contact `normal tolerance` (section 10.2) has changed. Previously, the larger of the user-defined and default (15% of the characteristic element length) tolerances would be used. This could result in unintended contact with a user-defined tolerance smaller than the default 15%. Now, any user-specified tolerances will always be respected, even if the default tolerance is larger. Note that the new behavior is no longer consistent with **Sierra/SM**. For consistent/legacy behavior, use `normal tolerance behavior = auto`.
- The `residual_vectors` solution has been removed and is now an option in `eigen` and `cbr` solution cases. See section 5.8 and section 5.3 for details and syntax. For eigen, loads can now be used to generate residual vectors in addition to a node list file.
- The **Exodus** output for the `ddam` solution case (section 5.5) now follows the ordering convention of the `eigen` solution case (ascending eigenvalue). Text-file output (`PREDDAM_RESULTS.txt` and `DDAM_RESULTS.txt`) also now follows the same convention.
- The `Orientation` option for one-dimensional elements now defaults to $\{0, 1, 0\}$ and automatically corrects the axis to be orthogonal to the x-axis. When the orientation changes from the input, warnings are output with relevant information about the coordinate frame.

- To alleviate confusing behavior, NBeams can no longer take just three offset arguments in the input deck. Instead, either zero or six offsets are required. To replicate legacy behavior (`offset x y z`), a user may specify `offset x y z x y z`.
- Documentation has been clarified and code warning messages have been added when specifying a contact definition (section 10.2) to reflect that the `normal tolerance` (whether user-defined or automatically calculated) is always used to calculate a critical gap, irrespective of the gap orientation. A user-provided **tangential tolerance** in the `contact definition` is ignored.

Bug Fixes

- An implementation bug where `e_dx`, `e_dy`, and `e_dz` were output in global coordinates, regardless of user-specified coordinate system, has been addressed. This bug did not affect `eforce`, even though it is requested with the same output keyword.
- A sporadic performance bug in CBR generation has been fixed.
- A parallel bug in output transfer matrix (OTM) generation for CBRs when interface nodes are included in the OTM (via history output) has been fixed.
- Code bugs in structural–acoustics eigen (`sa_eigen` solutions affecting mode shapes in some simulations have been addressed.
- Support for `viscofreq` in `sa_eigen` has been added.

Deprecated Features

- Sensitivities are deprecated for CBR superelements.
- The Transhock solution case is deprecated.

2.4. Release 5.24

New or Improved Features

- Analytic functions may now read data from element fields. See section 4.10.9.2 for more details.
- Stress transfer and geometric stiffness for beam and shell elements from SM has been enabled.
- **Sierra/SD** now supports **Exodus** mesh files with 5 or 13-noded pyramid elements. See sections 7.7 and 7.8 for more information.

Behavior Changes

- The Gauss point locations of all element types have been standardized for maintainability and consistency. This resulted in the use of different integration schemes in several instances, which could lead to slight changes in the simulation results. The only differences observed in the **Sierra/SD** test suite involved Wedge15 elements, and the changes were very minor.
- Gauss point “ijk” labels section 9.7.20.6 have also been standardized. This will result in a different naming convention for both Tet10 elements with 16 Gauss points (which previously did not follow the convention) and Wedge15 elements with 9 Gauss points (the locations of which were updated as outlined above). In both cases, reading in data (section 5.29) with the old labels will continue to be supported with an informative warning.
- Block definitions in the input deck that do not appear in the mesh will now trigger a fatal error. This behavior can be modified by the **RequireMatchedBlocks** parameter (section 4.4.1).
- Block definitions for omitted blocks (section 7.34) are no longer required. However, we still recommend keeping omitted block definitions in the **Sierra/SD** input for accurate syntax checking.
- Omitted blocks (section 7.34) are now compatible with contact definitions. Contact with omitted blocks will simply be skipped.
- Omitted blocks (section 7.34) will no longer be included in **Exodus** output.

Bug Fixes

- A bug in sa_eigen where some of the modal data was being excluded in calculations has been corrected. Additional testing comparing modalfrf (using sa_eigen and projection eigen) to directfrf has been added to catch errors.
- Exodus data transfer from a non-matching mesh file was found to be exhibit significant slow-downs, especially for larger models. The issue was related to repeated unnecessary reads of the Exodus database when trying to load the data from specific time steps. The unnecessary reads have been removed and performance has been restored. Additional testing has been added to catch future slow-downs.

This page intentionally left blank.

3. HOW TO RUN SIERRA/SD

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. The **Sierra/SD** software evolved from the "Salinas" package.

Sierra/SD tutorials for new users are available at
`https://snl-wiki.sandia.gov/display/CKB/How-to+articles`.

This section describes the command line arguments and workflows typical of **Sierra/SD** analysis. This section is primarily applicable to analysts at Sandia. Chapters 4 through 10 describe **Sierra/SD** capabilities invoked via the input deck. Section 11 provides example problems.

Many conventions are used in this manual. `Teletype` indicates new terms. Blue boxes are used for input deck listings. Pink and purple boxes or red fonts are used for warnings.

3.1. Accessing Sierra/SD

At Sandia Sierra is installed on many Unix-based High Performance Computing (HPC) systems. This includes dedicated and shared blades, shared computational clusters (e.g. cee-compute) and large queued clusters (Institutional Clusters such as Eclipse). To run Sierra applications you need to have access to one of these machines. You also must request "SIERRA Analysts Code Access" through WebCARS.

3.2. Modules and Executables

Configuration of Sierra applications is controlled via modules. The first step to running a Sierra application is to load the appropriate module for the desired version. For example:

```
$ module load sierra
```

Several modules are commonly used.

- **sierra**: This is the latest released version of Sierra and is generally the recommended version to be used.
- **sierra/X.YY**: This will load a specific Sierra version, for example "sierra/4.58". Sierra releases are done twice a year and generally the previous four releases are available.

- **sierra/sprint**: This is the latest “sprint snapshot” version of Sierra. A new sprint snapshot is installed every three weeks. Sprint snapshots contain the latest developed features but have less quality assurance testing than the main releases. Sprint snapshots should primarily be used to do beta evaluation of newly developed features.
- **sierra/daily**: This is the development version of Sierra built on the previous night. No quality assurance is done on this version. This version should only be used when directly working with the development team to help evaluate a bug fix.

3.3. The Sierra/SD salinas Executable

The primary executable for running structural dynamics analysis is called “salinas”. It uses a text format input deck to configure a simulation. This users manual primarily describes the input deck options and format. An example invoking a basic serial **Sierra/SD** analysis is:

```
$ module load sierra
$ salinas -i deck.inp
```

The commonly used command line arguments to the salinas executable are given in Table 3-1.

Table 3-1. – Command Line Options Part One.

String	Descriptor
`-h`	Prints help message
`-i`	Path and name of the input deck. The input deck is traditionally given the extension `.inp`. However, any name is allowed. If no ’-i’ is given the last argument of the line is assumed to be the input deck name.
`-l` or `-o`	Name for the output log file. By default, output to `<input_deck_name>.rslt` in serial or `<input_deck_name>_0.rslt` in parallel.
`-n`	Do not overwrite any existing output log and diagnostic `.dat` files. Do write a new file appending a 1, 2, 3, etc. to the name.
`-d`	Change working directory for the run. Equivalent to `cd` to that directory and running salinas
`-define`	Define variables for Aprepro [57] pre-processing of the input deck. See Section 4.1 for details

Table 3-2. – Command Line Options Part Two.

String	Descriptor
`--check-syntax`	Do initial syntax checking of the input deck and then stop before reading mesh or doing any large calculations. This option can be used to debug model input in serial prior to submitting to a queued system. In contrast to a normal simulation, **Sierra/SD** will **not** exit when fatal errors are encountered. This enables the discovery of multiple errors at once, but has the potential to result in cascading errors or even crash the code. Setting `exit on fatal = true` in the `parameters` section (beta capability, see table 4-7) will override this behavior and instead exit on the first fatal error.
`-beta`	Enable use of 'beta' capabilities. These are capabilities in active development, are subject to change without notice, and may have less rigorous testing.
`-nt`	Number of threads to use per MPI rank. Using more threads than physical cores (i.e., hyper-threading) may result in decreased performance. This option is only supported on a few platforms. Ask the **Sierra/SD** team for more information.
`-ndevices`	Number of GPUs to use per hardware node. This option is in development and not ready for general use.
`-create-xml`	Create an xml format command specification. Primarily for interacting with graphical input file creation utilities such as SAW. [55]

3.4. MPI Parallel Execution

Sierra/SD may be run either in serial or in parallel with MPI. In parallel **Sierra/SD** has demonstrated efficiency to thousands of processor cores.[13] Some basic work-flow examples for parallel execution are provided below. Information on using on the current platforms is available at High Performance Computing website `https://computing.sandia.gov`

Additional steps are necessary to execute in parallel instead of serial. The following examples use the input deck `deck.inp` and the **Exodus** mesh file `mesh.exo`.

3.4.1. Number of MPI Processes Needed

MPI is an optimization that can decrease run time. The configuration of Sierra depends on the architecture. A central processing unit (CPU) has multiple cores and possibly a graphical processing unit (GPU) accelerator. These notes only discuss the CPU. A user must know one thing about the CPU, the number of cores per node (or processor). One way to determine the cores per node is to use the `lscpu` command and look for `Core(s) per socket`. Another way is `cat /proc/cpuinfo`. Look for `cpu cores`.

Running **Sierra/SD** in parallel requires the user to specify how many MPI ranks will be used. Choosing the number of cores so that the number of elements per subdomain approximately 5000 typically works fine. Generally a computational "node" will run several MPI ranks and many computational nodes may be used simultaneously. Selecting the number of ranks to use is based on three concerns. First, each computational node has finite memory and enough nodes must be used to fit the problem in memory. Second, use of more MPI ranks and more computational nodes can reduce time to solution. Third, efficient use of machines should be considered as no analysis will achieve perfect parallel efficiency and using many computational cores on a small problem may provide sub optimal machine utilization. The appropriate number of MPI ranks to use is primarily determined by how many degrees of freedom (DOF) are in the model. Though machines are variable Table 3-3 can be used as a rough guide help guide selection of the number of MPI ranks and thus computational cores to use.

memory per 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 3-3. – Determining Number Of Processors Needed.

Memory use depends on a variety of factors including the element type used, the solution strategy and the output processing. The numbers in the table are generally conservative. Fortunately, in most cases it is not critical and compute node memory suffices.

For high memory use analyses it may be necessary to run **Sierra/SD** with half the maximum number of cores available per computational node. The reason is that using half MPI ranks doubles the memory assigned to each rank. As there is significant memory overhead for an MPI rank, memory limited analyses will benefit more by adding more memory to each rank than spreading the mesh to a larger number of ranks.

3.4.2. Mesh Decomposition

If running in parallel, prior to running the `salinas` executable first the **Exodus** mesh file must be parallel partitioned. Mesh decomposition involves dividing the input mesh file into several pieces, one of which will be read by each MPI rank. Several tools will decomposition the mesh into subdomains. The recommended tool is `stk_balanceparallel computing!stk_balance`. It is documented at the command line,

```
workstation_prompt> stk_balance  --help
```

```
workstation_prompt> launch -n 16 stk_balance --sd mesh.exo -o split
```

The latter command will decompose the mesh file into 16 parts with names beginning with `mesh.exo.16.00`, `mesh.exo.16.01`, etc. and place those spread files into the directory 'split'.

Storing the decomposed mesh files in a separate directory is optional. The location of the input mesh file is specified via commands in the input deck as described in Section 4.2.

3.4.3. *Running the* Sierra/SD *Executable in Parallel*

Here is a way to run **Sierra/SD** on 16 MPI ranks.

```
$ launch -n 16 salinas -i deck.inp
```

The input file contains the location of a partition of the mesh file into 16 subdomains. For queued systems, extra commands are needed to access the queue. See Section 3.6 for more examples on different systems.

3.4.4. *Post Processing in Parallel*

In parallel **Sierra/SD** creates several output files as described in Section 9.

- **log**: In serial the default log file name is `<input_deck_name>.rslt`. In parallel the default name is `<input_deck_name>_0.rslt`. This is a text output file giving vital information about the run. The '_0' indicates the is file written by processor zero. Information relevant to the global model behavior is accumulated to processor zero for output.
- **Mesh based Exodus**: The output name will depend on the structure of the input deck, but usually follows a pattern like `<mesh_name_base>-out.<mesh_name_extension>`. This includes mesh based information such as nodal displacement. Similar to the decomposed input mesh this output **Exodus** file will be written as a decomposed file with one portion per MPI rank. Some post-processing tools such as Ensight or Paraview can read the decomposed **Exodus** file directly. For other tools the file must first be combined to a single file with the SEACAS tool 'epu'.[25]
- **Diagnostics**: This could include detailed information from the solver (`dd_solver.dat`) or other text and MATLAB format output files.

3.5. File system concerns

Sierra/SD can output large files and works best when writing to fast output systems. Running analyses on scratch or Lustre drives on HPC system or local '/scratch' on blades are ideal. Large shared disks such as '/gpfs1' are viable though may incur some runtime overhead. Drives that are not mounted by computational nodes of HPC systems such as `/tmp` should be avoided.

3.6. Workflow Examples

There are many ways to run **Sierra/SD** depending on the analysis size and machine used. Several common examples are presented.

The next commands run a serial job on a blade and examine the log file.

```
$ module load sierra
$ salinas -i deck.inp
$ cat input.rslt
```

Run a parallel job using four processors of a blade. Look at exodus output results via the textual output query tool explore.[58]

```
$ module load sierra
$ launch -n 4 stk_balance mesh.g
$ launch -n 4 salinas -i deck.inp
$ module load seacas
$ epu --auto mesh-out.g.4.0
$ explore mesh-out.g
```

Run a parallel job on a queued system using 72 MPI ranks and 2 computational nodes manually calling `sbatch`, then combine to a single file, all with a single queue submission. In this case assuming a machine like Eclipse where each computational node can run up to 36 MPI ranks. The `SEACAS` tool 'epu'[25] can run in parallel, but is usually run on only a few MPI ranks.

```
#!/bin/bash
launch -n 72 stk_balance -i mesh.g
launch -n 72 salinas -i deck.inp
launch -n 8 epu --auto mesh-out.g.72.00
```

Code 3.1. submit.sh

```
$ module load sierra
$ module load seacas
$ /usr/bin/sbatch --account=<acct_id> --nodes=2
↪ --time=4:00:00 ./submit.sh
```

Run a parallel job on a queued system using 8 MPI ranks spread to 4 computational nodes get access to more memory per computational process. In this case assuming a machine like Eclipse where each computational node defaults to 36 ranks.

```
#!/bin/bash
launch -n 8 stk_balance -i mesh.g
launch -n 8 salinas -i deck.inp
launch -n 8 epu --auto mesh-out.g.8.0
```

Code 3.2. submit.sh

```
$ module load sierra
$ module load seacas
$ /usr/bin/sbatch --account=<acct_id> --nodes=4
↪ --time=4:00:00 ./submit.sh
```

Run a parallel job on a queued system using the "sierra" script. If the mesh file was not yet decomposed for parallel execution the sierra script will automatically decompose it prior to running the `salinas` executable. By default, the sierra script will use `stk_balance` for mesh decomposition with the `-sd` option. Epu would run in serial on the HPC head node. Note the sierra script can be used to submit **Sierra/SD** analyses. However, the sierra script '–post' option is currently not functional with **Sierra/SD** (for other Sierra tools the post option automatically combines output mesh files, similar to the below "epu" command.) The sierra script can potentially be helpful to translate the number of MPI ranks to the correct number of nodes to use on a given machine. Additionally, the same sierra script submission command can be used on queued HPC systems and non queued blades and clusters.

```
$ module load sierra
$ sierra -j 72 -T 4:00:00 salinas -i deck.inp
$ module load seacas
$ epu --auto mesh-out.g.72.00
```

Run coupled **Sierra/SD** Fuego analysis on a non queued system for fluid structure coupling. In this case salinas will use 8 MPI ranks and fuego will use 16 MPI ranks for a total use of 24 MPI ranks. Note, this is an advanced use case.

```
$ module load sierra
$ launch -n 8 stk_balance salinas_mesh.g
$ launch -n 16 stk_balance fuego_mesh.g
$ launch -n 8 salinas -i salinas_deck.inp
↪ --fuego-coupling : -np 16 fuego -i fuego_deck.inp
```

3.7. Thread Parallelism

In addition to decomposition based MPI parallelism, **Sierra/SD** also supports thread parallelism on some platforms (currently Trinity and GCC development platforms). Threads are activated by the command line option "-nt <numThreads>". The 'numThreads' given will be the number of OpenMP threads to use on each MPI rank. Threaded execution is most valuable on large models. Using a mixture of thread parallelism and MPI parallelism can give optimal performance when the number of MPI processes required would otherwise be very large. As a rule of thumb thread parallelism will provide benefit when exceeding about 200 MPI processes or when more cores are required than MPI ranks to obtain more memory. When using thread parallelism, the number of threads used times the number of MPI ranks used should be setup to be equal the total number of processor cores available on compute nodes.

Note: while the number of threads used in **Sierra/SD** is controlled by the command line option "-nt", it is recommended that the user also set the environment variable 'OMP_NUM_THREADS' to be the same value. While **Sierra/SD** does not depend on 'OMP_NUM_THREADS', there might be other aspects of your workflow that would, and so we recommend setting both to be consistent. In fact, **Sierra/SD** will output a warning if 'OMP_NUM_THREADS' does not match the value set by "-nt".

3.8. Troubleshooting

A variety of issues can cause an analysis to fail. 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. The best troubleshooting strategy 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 ways to do this. First stand-alone tools are described. Second ways of using **Sierra/SD** capabilities to troubleshoot are described.

Queuing systems are inevitable in high performance computing. Any extra step lowering the risk of there being a mistake in the input deck is worthwhile.

3.8.1. Stand-Alone Tools

Currently, two tools exist which can help the user debug their mesh file, i.e., **Exodus** file: `Explore` and `cubit`.

3.8.1.1. Explore

`Explore` is an ACCESS/SEACAS utility that can be used to interrogate the **Exodus** file. One of the commands in `Explore` that can be used is **check**. It is used as follows:

```
$ explore cube.exo
.
.
EXPLORE> check

 Database check is completed
```

```
EXPLORE>
```

If there are any warning or errors, they will appear before the `Database check is completed` message.

3.8.1.2. Cubit

The Cubit team has developed a GUI-based tool named `cubit`. It can be used to visualize mesh quality parameters of the **Exodus** file. For questions about `Cubit`, please contact the Cubit team at **cubit-dev@sandia.gov**.

3.8.2. *Using* Sierra/SD *To Troubleshoot*

A 'model_check' analysis 5.7 report diagnostic information about model setup. It is optimized to avoid computationally expensive operations, and can be run in serial. For example, the information can help pinpoint areas of the model causing linear solver issues. Users are advised to do some sort of model checking before submitting jobs in long queues. Available model check information includes kdiag, mlumped, constraint_info, and ElemEigChecks.

Syntax checking 4.4 is helpful.

The user has to take additional steps before executing the parallel version of **Sierra/SD**. One of the steps is to run `stk_balance` to decompose the finite element model. 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.

These steps are due diligence before running a large model on a queued system. Running the large model on a smaller computer (without a queue) to confirm that the simulation fails in the expected way (by running out of virtual memory) is also worthwhile.

The output includes the Approximate Matrix infinity norm ratios table. This table shows information about the stiffness, mass and dynamic matrices. The value shown is the ratio of the largest diagonal entry to the smallest diagonal entry. If the matrix has a 0 on the diagonal, then 0 is shown.

3.8.3. *Modal Analysis*

It is possible for the `eigen` solution method (discussed in Section 5.8) to diverge.

Section 4.4 mentions the **eig_tol** parameter. The default value is about 10^{-16} (for the modal solution case). Adequate modes can be determined with much larger values of `eig_tol`. Values such as 10^{-8} are reasonable.

At times an analyst may choose to use the `eigen` method to diagnose a 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 issue is resolved, if eigenvectors are still needed, don't forget to reset `eig_tol` to a small value.

The modal analysis algorithm depends on a linear solver, and assumes that the linear systems are solved accurately. The shift is almost always negative. Increasing the magnitude of a negative shift forms linear systems that are easier to solve, and makes the eigenvalue problem harder to solve. The iterative linear solver (GDSW) parameters trade off between speed and accuracy. For example, a small value of the `solver_tol` table 4-10 increases accuracy and computational expense. Solving the linear system more accurately makes the eigenvalue problem easier to solve.

3.8.4. Evaluating Memory Use and Fixing Issues

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 9.9).

It is often helpful to identify problematic subdomains. The partition of a finite element mesh into subdomains may be visualized by rejoining the partitioned files into a new file using `epu` with the `-add_processor_id` option.

3.8.5. Problematic Elements and Connectivity

Many problems are caused by "bad" elements. Element quality is a function of both the element geometry and the type of element. The Hex8, Wedge6, Tet4, Tria3, and QuadT elements implement a condition number. Note that the QuadT consists of two triangular elements. See section 9.8.3 for more details.

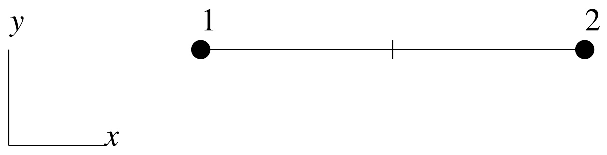
Overall second and higher order finite element methods are much less sensitive to element quality than low order methods. However, there is a tipping point. A high order element will invert before[63] the corresponding low order element. With that in mind, it is noteworthy that **Sierra/SD** ignores this issue.

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 3-1 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 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.

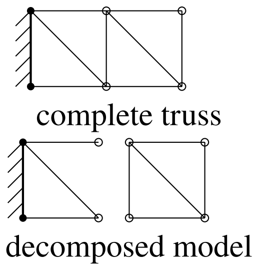
Figure 3-1. – Single *Spring* element.



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 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 9.8.7.

Decomposition weakness The broken figure is an issue for trusses (or rods) and some other elements. The truss in the top part of Figure 3-2 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 3-2. – Truss Decomposition Issues.



Truss elements can be used in **Sierra/SD**. Simulation issues have been traced to poor decompositions in the past. In practice these issues have not come up with stk_balance (see 3.1).

Poor Connectivity A structure that has poorly connected regions can be difficult to analyse. If elements have not been properly equivalenced, there can be thousands of zero energy modes in the model. **Sierra/SD** can identify up to a few dozen redundant modes in the best of cases.

Poor Units In models with shell elements, rotational degrees of freedom are active. The rotations scale like the displacements divided by the length. An unfortunate choice of length unit will cause ill conditioning. In theory a unit of length that is much too large could lead to inaccurate rotations. And a unit of length that is much too small could lead to inaccurate displacements. This issue has not been investigated. A smaller linear solver tolerance table 4-10 would generally mitigate the issue.

3.8.6. Over-determined Constraints and Loss of Rigid Body Modes

Building models with `tied surfaces` or other multi-point constraints (MPCs) may involve more than one iteration. Computing eigenvalues is a common way to check a model. Gap Removal is a light-weight solution method 5.32 enabling users to obtain information about their tied surfaces earlier in the process, before having to do all the work necessary to successfully set up, submit, and run the massively parallel simulation of the modes.

Solving eigenvalue problems has issues of its own. The eigenvalue decomposition of a model with no prescribed boundary conditions would generally be expected to produce six rigid body modes. A variety of internal constraint issues can cause loss of these rigid body modes. Such artificial constraint of a model can substantially degrade the quality of the overall solution. **Sierra/SD** has several internal correction algorithms and warnings to detect and repair issues, such as GDSW **con_tolerance**, described in Section 4.7. However, care should be taken to minimize the complexity of constraint equations. Generally, having different types of constraints using the same nodes can cause problems. The following types of constraint combinations are known to be particularly problematic:

- Intersection of tied data or contact constraints where one node is tied to two or more faces.
- Chains or cycles of tied/contact constraints. These could involve a node on side ‘A’ of an interface being tied to faces on side ‘B,’ while, simultaneously, nodes on side ‘B’ are tied to faces of side ‘A’. This case is especially common in “self contact” situations, for instance, when a tied surface folds over onto itself.
- Intersection of tied/contact constraints and rigid set constraints where a tied node of the node-face constraint is also part of the rigid set.
- Manually specified MPCs that form long chains, cycles, or extensive intersections.
- Intersection of complex manually specified MPCs with any other type of constraint such as rigid set or tied data.

3.9. Input Deck Modification Scripts

Many scripts are available for manipulating the input deck.

3.9.1. Extracting Sections

`extractSection` is a command-line tool to extract sections of a given type from a list of **Sierra/SD** input files.

- Access to the tool is obtained by loading the sierra module.
- All matching sections will be extracted from the files passed to the script.
- All extracted sections are printed to screen.

- Comments are left in the input deck documenting each extraction. To remove them after running the script and inspecting the results, another tool, `cleanupSyntaxConversionNotes` is provided. It operates recursively from the current directory.
- Prior to extraction, an unmodified copy of each input file will be saved with a ".bcp" extension.

For example, to extract load sections from a single (or multiple) files, use

```
$ extractSection --section=load file1 (file2 file3 ...)
```

To operate on all .inp files in the current directory, use

```
$ extractSection --section=... *.inp
```

(replace `*.inp` with `**/*.inp` to run recursively, i.e., in the current directory and all below).

To operate on list of files generated from a previous command such as find:

```
$ previous command | xargs extractSection ...
```

To write the extracted sections to a file, use shell redirection (>):

```
$ extractSection ... > all_materials.inp
```

To further modify the extracted sections (sortSections, uniqSections, ...):

```
$ extractSection ... | subsequent command
```

3.9.2. Sorting Sections

`sortSections` is a command-line tool to sort sections of a **Sierra/SD** input deck.

- Access to the tool is obtained by loading the sierra module.
- An input file may be specified as a command-line argument.
- If no file is given, the command-line standard input will be used.
- Sorted sections are printed to screen.

For example, to operate a specified input file:

```
$ sortSections sd.inp
```

To operate on the output from a previous command such as extractSection:

```
$ previous command | sortSections
```

To write the sorted sections to a file, use shell redirection (>):

```
$ sortSections ... > all_materials.inp
```

To further modify the sorted sections (e.g. with uniqSections):

```
$ sortSections ... | subsequent command
```

3.9.3. *Removing Duplicate Sections*

`uniqSections` is a command-line tool to remove duplicate sections of a **Sierra/SD** input deck.

- Access to the tool is obtained by loading the sierra module.
- An input file may be specified as a command-line argument.
- If no file is given, the command-line standard input will be used.
- Unique sections are printed to screen.

For example, to operate a specified input file:

```
$ uniqSections sd.inp
```

To operate on the output from a previous command such as extractSection:

```
$ previous command | uniqSections
```

To write the unique sections to a file, use shell redirection (>):

```
$ uniqSections ... > all_materials.inp
```

To further modify the unique sections (e.g. with sortSections):

```
$ uniqSections ... | subsequent command
```

3.9.4. *Converting Includes*

`convertIncludes` is a command-line tool to convert legacy includes to APREPRO syntax.

- Access to the tool is obtained by loading the sierra module.
- Files are overwritten by this script.
- All other areas of the input other than `include` definitions will remain unchanged.

- Comments are left in the input deck documenting each change. To remove them after running the script and inspecting the results, another tool, `cleanupSyntaxConversionNotes` is provided. It operates recursively from the current directory.
- SD-style includes supported an implied ".inp" (e.g., `include file` would include the file `file.inp`). This is not supported with Aprepro includes, and the missing ".inp" will need to be manually added.

To convert a single (or multiple) files, use

```
$ convertIncludes file1 (file2 file3 ...)
```

To convert all .inp files in the current directory, use

```
$ convertIncludes *.inp
```

(replace `*.inp` with `**/*.inp` to run recursively, i.e. in the current directory and all below).

To convert a list of files generated from a previous command (grep, find, etc.), use

```
$ previous command | xargs convertIncludes
```

3.9.5. Converting Coordinate Systems

`convertCoordinateSystems` is a command-line tool to convert legacy coordinate system sections to the new syntax.

- Access to the tool is obtained by loading the sierra module.
- Files are overwritten by this script, and message is printed for each file touched.
- Any comments in the coordinate section will be removed.
- All other areas of the input other than `coordinate` definitions will remain unchanged.

To convert a single (or multiple) files, use

```
$ convertCoordinateSystems file1 (file2 file3 ...)
```

Convert all .inp files in the current directory using

```
$ convertCoordinateSystems *.inp
```

(replace `*.inp` with `**/*.inp` to run recursively, i.e., in the current directory and all below).

And too convert a list of files generated from a previous command (grep, find, etc.), use

```
$ previous command | xargs convertCoordinateSystems
```

3.9.6. Converting Named Materials

A command-line tool, `commentDeprecatedMaterialNames` is provided to comment out any usage of the `name` keyword in material sections.

- Access to the tool is obtained by loading the sierra module.
- The tool takes no arguments, and will operate on all files in or below the current directory.

If desired, another command-line tool, `renameMaterials` is provided to rename materials to the name given in the `name` keyword (if applicable).

- Access to the tool is obtained by loading the sierra module.
- The tool takes no arguments, and will convert all files in or below the current directory.
- Files are overwritten by this script, and a message is printed for each change.
- It modifies both the name in the `MATERIAL` section, and material call in the calling (e.g., `Block`) section, even if they are not in the same file (it traverses included files).
- Only materials with a material *number* are replaced. Named materials (e.g., `MATERIAL foo`) are unchanged.
- If the same name is given for multiple materials (or an identically-named material is already defined), it will append the old material number to the name. For example, if material 11 and 12 both have `name = one`, material 11 will be renamed `one`, and 12 will be renamed `one_12`)
- Comments are left in the input deck documenting each change. To remove them after running the script and inspecting the results, another tool, `cleanupSyntaxConversionNotes` is provided. It also operates recursively from the current directory.

For example, after running `renameMaterials` + `cleanupSyntaxConversionNotes`, you would see the following input deck transformation:

```
BLOCK 1            -->  BLOCK 1
  material 11      -->    material one
END                -->  END
                   -->
MATERIAL 11        -->  MATERIAL one
  name one         -->
  ...              -->  ...
END                -->  END
                   -->
BLOCK 2            -->  BLOCK 2
  material 12      -->    material one_12
END                -->  END
                   -->
```

```
MATERIAL 12          -->  MATERIAL one_12
  name one           -->
  ...                -->  ...
END                  -->  END
                     -->
BLOCK 3              -->  BLOCK 3
  material foo       -->    material foo
END                  -->  END
                     -->
MATERIAL foo         -->  MATERIAL foo
  name bar           -->
  ...                -->  ...
END                  -->  END
```

3.9.7. *Removing Syntax Conversion Notes*

`cleanupSyntaxConversionNotes` cleans up comments added by other syntax conversion and input deck modification tools. It should be called as a last step, after reviewing all changes.

It takes no arguments, and will modify all files in or below the current directory.

This page intentionally left blank.

4. GENERAL COMMANDS

4.1. Input deck format

The input is a finite element mesh (in **Exodus** format) paired with an input deck setting both the materials used in the model and what to simulate. Once mesh and materials are set, any type of analysis can be performed. An input deck (or text input deck) consists of sections. The order of the sections does not matter, nor does the order of the directives within a section. Commands are case-insensitive.

The Sierra tools related to **Sierra/SD** include the Cubit geometry and mesh generation toolkit, the ACCESS/`SEACAS` tool suite for manipulating exodus files and the Nasgen program for convert NASTRAN files to both the exodus file and the text input deck.

Traditionally the sections of an input deck are ordered so that directives that are most likely to change are near the top. The solution section selects an analysis type. The file section sets the **Exodus** file name. An **Exodus** file for a structural model typically contains many element blocks. The input deck must have a block section for each element block. And a material section specifies each of the materials used in one or more element blocks.

Typically, the input deck 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 deck 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:" tabs, spaces, and line feeds.

Comments

Several options are available for a comment specifier. These are listed in Table 4-1. For any string used to specify a comment, all characters following the comment string are skipped. [1]

[1] To be safe, define comments as one of the allowable comment characters (i.e. "//") followed by a space.

Table 4-1. – Comment String Options.

String	Descriptor
//	C++ style line comments.
/*...*/	C++ style block comments.
#	Line comment. Used in scripts and Sierra input.
$	Line comment. Aprepro [57]/`SEACAS` default.

Meta-characters

Sierra/SD supports the use of several meta-characters in the input deck. They are listed below, along with what they represent.

%P The number of processors for the run.

%B The base file name. For example, if the input deck is `example.inp`, %B will be replaced with **example**.

%T The **Sierra/SD** start time. This is the same time that is echoed to the screen at the beginning of the run, but in a shorter and more file-friendly format. For example, if the **Sierra/SD** start time was **Tue Sep 24, 2019, 16:05:41**, %T will be replaced with **2019_09_24_16_05_41**.

Skipping Sections

Occasionally an entire section may need to be commented out. This may be done using "//" on each line of the section, or surrounding the section with the block comment characters "/*" and "*/". A third way to comment out an entire section is to begin it with double "$$" characters. In the following, block 19 is commented out, and block 43 is active.

```
$$ BLOCK 19            // this section skipped
  material 1
END

BLOCK 43                 // this section valid
  material 1
END
```

Except for special cases such as file names, the input deck is case-insensitive. Either the single quote ' or the double quote " may be used throughout the input to keep multiple words together (e.g. a name or title). 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

The Algebraic PREPROcessor, `aprepro`, in the ACCESS/SEACAS tool suite can be run either standalone, or as part of the analysis. The use of Aprepro is enabled by default, but can be disabled with the `-noaprepro` command line flag. Aprepro can be used for a variety of purposes.

1. Define variables on the command line. This is especially useful for automated runs such as optimization and uncertainty quantification.
2. Simplify input by allowing algebraic expressions, e.g. Y={ 4 * 3}.
3. Automatically include text of other files.
4. Manage various systems of units, e.g. Y={ 10 * psi }.

For details on Aprepro in general, and for standalone documentation, please refer to the SEACAS documentation.[57] All the rules for command line substitution apply to the built-in capability. Definition of command line variables during analysis requires specification of a command line argument, `-define`, as used in the example.

```
sierra salinas --define "E_val=10E6 nu_val=0.30" -i example.inp
```

In this example, the text "E_val" in the input deck, `example.inp` will be replaced with "10E6". Likewise, "nu_val" will be replaced by 0.30.

Including Files

The input parser supports nested includes using Aprepro. Files may be included to any depth.

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

The `include` command may occur anywhere on the line (though for readability and consistency we recommend that it be the start of the line). Case sensitivity will be preserved.

Input Summary

Summarizing, a minimum of two files are needed to run **Sierra/SD**, namely, a text input deck, e.g. `example.inp`, and an **Exodus** input deck,[48] 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 4.2).

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

Integer List An integer list may be required as a parameter for some 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 if such lists have spaces they be placed in (single or double) quotes, i.e. ’1 2’ is preferred, but 1 2 without quotations is also acceptable.

Lists are also enabled in all areas of the code that support the input of multiple entities (blocks, nodesets, or sidesets) on a single line. Integer ranges may also be used as shown above to select all entities within in a range of ids. When using an integer range, there must be at least one valid entity contained in the range. Suppose that a mesh contains element blocks 1, 5, and 15. The range “1:10” selects blocks 1 and 5. The range “1:20” selects all blocks. However, the range “20:30” would return an error.

The keyword `all` is a special keyword that will select all entities of a given type. Most contexts that accept multiple entities will also recognize an optional “remove” keyword. Specifically, it is not currently supported for block definitions (section 6.6), spot welds (section 10.4), and omitted blocks (section 7.34). The effect of this keyword is that any further entities listed on the same line will be removed instead of added to the overall list. For example, the following example selects all nodesets on the mesh *except* nodesets in the range 100:200 for output on the history mesh.

```
HISTORY
  nodeset all remove 100:200
END
```

4.2. Input Mesh Geometry 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
user subroutine file	File containing user subroutines, currently relevant only to user mount element

4.2.1. *Geometry_file*

The geometry file is used for input of the mesh geometry including the nodes, elements, connectivity. Element specific data can be set in the geometry file by setting the element attributes. Element parameters can also be used to configure the elements from the input deck, as summarized in table 6-4

4.2.1.1. Multiple processors

In a multiprocessor environment, the file name is determined by appending the "dot qualified" processor number and processor id to the geometry file specification. [2] For example, if the user specifies,

```
geometry_file='temp1/example.par'
```

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
```

In this example the input file is not in the current working directory.

Specifying the **Exodus** mesh input file in a way that does not depend on the number of input files (or MPI ranks) saves time in the long run. If the input file is `example.exo` and the partitioner is configured to generate the files 'example.exo.4.0',. . . then the same **Sierra/SD** input deck works in serial and in parallel.

4.2.1.2. Single processor

On a single processor, the file is not "spread", and the full file path is provided. A representative serial `file` section follows.

```
FILE
  geometry_file example.exo
END
```

Note:

- A single processor run, even using MPI, will not append the number of processors and processor ID to the file name.
- Section 3.1 shows the steps involved in the parallel execution of **Sierra/SD**.

[2] 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.

4.2.2. **Exodus** *Database Naming Conventions*

There are three basic conventions that apply to user input for various command lines. The conventions concern side sets (surfaces), node sets and blocks.

First, the **Exodus** side set is referenced as a surface. In Sierra, a surface consists of element faces plus all the nodes and edges associated with these faces. A surface definition can be used not only to select a group of faces but also to select a group of edges or a group of nodes that are associated with those faces. For nodal boundary conditions that use the surface specification, all the nodes associated with the faces on a specific surface will have this boundary condition applied to them.

A group of elements can also be used to select other mesh entities. In Sierra, a block consists of elements plus the associated faces, edges, and nodes. The block and surface concepts are similar in that both have associated derived quantities.

Blocks, sidesets, or nodesets can be grouped together into `assemblies`.[59] However, an assembly must contain entities of the same "type", where "type" is either a block, sideset, nodeset, or another assembly. An assembly cannot contain itself, either directly or indirectly. Currently, not all tools support assemblies. One way to create them is with `io_modify`. In addition, assemblies can be created directly in the **Sierra/SD** input deck using the `ASSEMBLY` section (section 4.2.3).

The specification for a surface identifier can have the form `surface_id`, where `id` is the integer identifier for the surface. Surfaces can also be identified by the `id` alone. For example, if the side set identifier is 125, the surface could be called `surface_125` or simply `125`. It is also possible to name a surface in some mesh generation programs, and that name can be used in the input file. The specification for element blocks and nodesets are defined similarly (`“block”_id/id/name` and `“nodelist”_id/id/name`). Assemblies can also be specified similarly (`“assembly”_id/name`), but specification of assemblies by `id` alone is not allowed. Assemblies can be substituted anywhere in the input deck where a list of blocks, sidesets, or nodesets is expected. If the assembly itself contains sub-assemblies, the set of unique leaf blocks, sidesets, or nodesets will be used, depending on context.

4.2.3. *ASSEMBLY section*

Assemblies can be created using the `ASSEMBLY` section of the **Sierra/SD** input deck as shown below (syntax 4.1). The assemblies created in this way can then be used elsewhere in the **Sierra/SD** input deck. If an integer is given as the assembly name, then the integer will be used as the assembly identifier and the assembly name will be `assembly_id`, where `id` is the assembly identifier. Note that a `ASSEMBLY` section can only refer to assemblies that either already exist in the input mesh or were defined in a prior `ASSEMBLY` section. If an assembly already exists on the input mesh, defining it again using a new `ASSEMBLY` section will result in modifying the assembly so that it contains the union of the entities in the original and the new instance. Additionally, a warning will be printed in the **Sierra/SD** results file about modifying an existing assembly.

```
BEGIN ASSEMBLY <string>
  // exactly one of the following:
  block = <list(block)>
  block = all
  sideset = <list(sideset)>
  sideset = all
  nodeset = <list(nodeset)>
  nodeset = all
END
```

Syntax 4.1. ASSEMBLY example

4.2.4. *Exodus Naming Limitations*

Sierra/SD supports the use of mesh names in the input deck and lists of mesh objects. As a result, it becomes ambiguous which keywords are mesh names and which are intended as input syntax. For example, a nodeset named "fixed" creates ambiguity when the user specifies "`nodeset 1 fixed`". This could mean that nodeset 1 is fixed in space, or it could be a list of nodesets "1" and "fixed".

To resolve this ambiguity, **Sierra/SD** maintains a list of reserved keywords which may not appear in the mesh. This list has over 400 entries, and changes with each release. The rules of this list are simple; any valid syntax which appears adjacent to a nodeset list may not appear in the mesh's nodesets. The same rule applies to sidesets and blocks. Since assemblies are valid in all three contexts, they inherit the restrictions of all three.

There are no keywords in the **Sierra/SD** input deck which begin or end with an underscore. Therefore, "_fixed_" is a valid name for any mesh object in any context. Similarly, "assembly_5_front" is obscure enough to be safe from this restriction.

Pure integer values are not allowed as mesh names for the same reasons. This is not a restriction on names like "Block_5", but blocks named "5" are not allowed.

A warning is issued if reserved characters are found in the mesh names. These warnings are not fatal, but having reserved characters in the mesh names severely limits our ability to parse them in the input deck. The reserved character list includes comment characters (//,/*,*/,#, and $), and special-use characters (=,',",and :), which have extra meaning in **Sierra/SD** input decks.

Since the list of reserved keywords is based on the allowable syntax in **Sierra/SD**, it will change based on the current version of the code. The current list of reserved keywords for the version of the code you're using can be printed to screen at any time using the "–keywords" command-line option. Additionally, the ambiguous naming checks can be changed to a warning or ignored entirely using the **reserved_keywords** parameter (section 4.4.1).

4.2.5. *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 9.9). 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 9), an output **Exodus** file will be created. An **Exodus** file is a binary file (NetCDF). It contains the original geometry and the 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 a hyphen followed by the case name if defined, (or `-out` otherwise) before the file name extension. The output **Exodus** file will be written to the same directory where the geometry file is stored.

4.3. Transferring Exodus data from other applications

Sometimes a user might wish to use quantities calculated from another simulation (in **Exodus** format) to drive a subsequent **Sierra/SD** analysis. **Sierra/SD** currently supports

(i) reading data computed in other applications from the **Sierra/SD** input mesh file, and

(ii) reading data computed in other applications from an auxiliary mesh, or source mesh file, using non-matching mesh transfers.

4.3.1. *Reading* Exodus *data from the input mesh file*

User-defined setup of *some* input variables using `initialize variable name = <key>` in the `FILE` section, where <key> is a **Sierra/SD** keyword such as **displacement(x)**. The `step = first|last|<int>` or `time = first|last|<real>` options can be used to control the time step on the mesh database from which field data are read. In accordance with **Exodus** format, steps are one-based: `step = 2` refers to the second step on the mesh. The `time = <real>` option will select the nearest step with a time *greater than or equal to* the requested value.

An example which initializes displacements on the mesh is shown below, where nodal displacements are stored as "dx", "dy", and "dz" on the input geometry file `input_mesh.g`.

```
FILE
  geometry_file = input_mesh.g

  # nodal displacement components stored in input_mesh.g...
  initialize variable name = displacement(x)  # x-component
    variable type = node                      # nodal displacement
    read variable = dx                        # from input "dx"
    time = 2.5                                # at the nearest step
                                              # with time >= 2.5
  initialize variable name = displacement(y)  # y-component
    variable type = node                      # nodal displacement
    read variable = dy                        # from input "dy"
    step = FIRST                              # at the first step

  initialize variable name = displacement(z)  # z-component
    variable type = node                      # nodal displacement
    read variable = dz                        # from input "dz"
    step = LAST                               # at the last step
END
```

Options `function` and `variable type` may also be used to initialize nodal variables with user-defined functions as shown in input 4.1. The first component of displacement is calculated using an analytic expression that scales *dx* from the mesh.

```
FILE
  geometry_file = input_mesh.g

  initialize variable name = displacement(x) # x-component of
    variable type = node                     # displacement
    function = my_disp_x
END

FUNCTION my_disp_x
  type analytic
  expression variable dx_Input = nodal dx
  evaluate expression ="0.5 * dx_Input"
END
```

Input 4.1. Input displacement based on analytic function

Shorthand notation of the user-defined label mapping is also supported and demonstrated below for the same displacement example as above. For vector fields, ‘x’, ‘y’ and ‘z’ are appended to the base variable name, while for tensors, ‘xx’, ‘yy’, ‘zz’, ‘xy’, ‘yz’ and ‘zx’ are appended. For symmetric tensor fields, the order of off-diagonal indices is irrelevant (‘xy’ is equivalent to ‘yx’).

```
FILE
  geometry_file = input_mesh.g

  initialize variable name = displacement # displacement is stored
    variable type = node                  # in nodal fields
    read variable = d                     # named "dx", "dy" and "dz"
    step = 23                             # at the 23-rd step
END
```

4.3.2. *Transferring* Exodus *data from a non-matching mesh file*

When the geometry of the source mesh differs from the **Sierra/SD** input mesh, a limited capability is provided to transfer *some* fields from the auxiliary mesh using a **transfer** section. The syntax for the **transfer** section is shown below:

```
TRANSFER <exo_file> // source file
  source blocks = <list(int)>
  source sidesets = <list(int)>
  destination blocks = <list(blocks)>
  destination sidesets = <list(sidesets)>

  mismatch mesh behavior = IGNORE|EXTRAPOLATE|TRUNCATE|PROJECT|ABORT

  initialize variable name = <string> // SD internal naming convention
    read variable = <string>          // the name on the mesh
      variable type = node|element|sideset
      step = first|last|<int>  (optional, option 1)
      time = first|last|<real> (optional, option 2)

  copy variable <string>
    variable type = node|element|sideset
    step = first|last|<int>  (optional, option 1)
    time = first|last|<real> (optional, option 2)
END
```

Syntax 4.2. TRANSFER section syntax

The `initialize variable name` option is intended to initialize a variable on the **Sierra/SD** mesh once at the beginning of analysis, akin to the same option in the FILE section (section 4.3.1).

Currently, the only supported fields for the **initialize variable** option are nodal displacements, which are transferred by default from the final source file timestep and used to update the initial geometry of the destination mesh, and in the definition of element-local coordinate systems (section 6.6.1.3).

On the other hand, the `copy variable` option is used to transfer potentially time-varying fields from the source mesh to the **Sierra/SD** mesh over the course of an analysis. Element or face variables are *copied, not interpolated* from the source file to the corresponding nearest element or face of the destination mesh. As with `initialize variable name`, the `copy variable` command also supports an optional `time` or `step` specification to load data from a step other than the default (final) step.

For example, to use displacements labeled as (`dx`, `dy`, `dz`) on "source.exo" to update the initial geometry of "destination.exo" (**Sierra/SD** input mesh),

```
TRANSFER source.exo
  initialize variable name displacement
    read variable = d
    variable type = node
END

FILE
  geometry_file = destination.exo
END
```

Input 4.2. TRANSFER geometry update

To use the displacements from only blocks 1 and 2 on the source mesh to update only block 10 on the destination,

```
TRANSFER source.exo
  source blocks = 1 2
  destination blocks = 10

  initialize variable name displacement
    read variable = d
    variable type = node
END
```

Input 4.3. TRANSFER subregion example

The following example demonstrates how to apply the source element field `press` (e.g., from a surface mesh) as a pressure load on the destination sideset `skin`.

```
FILE
  geometry_file destination.exo
END

TRANSFER source.exo
  copy variable = press # name of variable in source.exo
    variable type = element
    destination sidesets = skin
END

LOADS
  sideset skin
    pressure 1
    function = from_transfer
END

FUNCTION from_transfer
  type=ExodusRead
  interp=linear
  exo_var scalar press
END
```

Input 4.4. TRANSFER pressure from element surface to sideset

To enable pressure transfer from a sideset with id '1' on the source mesh to a sideset "skin" on the destination mesh, the `TRANSFER` section in the above example could be modified as follows:

```
TRANSFER source.exo
  source sidesets = 1
  destination sidesets = skin
  copy variable = press # name of variable in source.exo
    variable type = sideset
END
```

Input 4.5. TRANSFER pressure from sideset to sideset

There are several other known current limitations and guidelines for use of non-matching mesh transfers, as follows.

Unlike the `destination blocks|sidesets`, the `source blocks|sidesets` list does *not* currently support specification of blocks by name or by range.

While multiple transfer blocks and, therefore, multiple transfer *files* may be used, each transfer destination *field* must be unique. If a duplicate destination field is found, a fatal error will occur.

By default, mismatched domains are not checked when transferring. This behavior can be changed with the `mismatch mesh behavior` option. See options in figure 4-1.

Figure 4-1. – Mismatch mesh behavior options

IGNORE The receiving mesh object can be ignored and will receive no value. This is almost never a good idea as it can cause mesh objects just outside to have a default zero value when the nodes just inside the mesh might have very large values. This can result in a discontinuous receiving field.

EXTRAPOLATE This is the default behavior. The sending field is extrapolated beyond the bounds of the sending mesh. This can lead to extrapolation error, such as when a large gradient at the surface causes a negative values when only positive values are acceptable.

TRUNCATE The receiving coordinate is truncated back to the surface of the sending mesh to determine a value. This ensures that the receiving value is not outside of the field values in the sending mesh.

PROJECT This option is similar to TRUNCATE in which the receiving coordinate is projected back to the surface of the sending mesh to determine a value. In this case more effort is made to make sure that the projection is normal to the surface in the sending mesh. Sometimes gives a better result than Truncate but is a little more expensive to compute. If the PROJECT option is used in transferring of surface values, the sending mesh should envelope the receiving mesh. Failure to satisfy this condition will generally result in failure of the transfer.

ABORT If any receiving point is outside the sending mesh by more than the geometric tolerance, abort the simulation. Do not attempt to project, extrapolate, or otherwise handle the point.

4.4. Parameters

In the input deck, an *optional* Parameters section configures features that apply to all solution cases. A family of options have been added to allow advanced users to suppress specific warnings and errors. In this section all of the optional Parameters are described. Section 4.4.1 discusses suppressions. This section concludes with examples of warning suppression.

Keyword	**Type**	**Default**	**Description**
AllowExodusAttributes	*bool*	true	Seek undefined attributes on the mesh
AllowExodusDistFacts	*bool*	true	Apply distribution factors from the mesh
ComplexStress	*yes/no*	no	output complex stress in FRF solutions
condition_limit	*Real*	1e6	element quality output control
constraint_correction	*yes/no*	no	orthogonalize constraints to RBMS
defaultSpecificHeat	*Real*	none	material specific heat
DoInitialMassSolve	*bool*	true	transients: perform initial mass solve
eig_tol	*Real*	auto	Eigenvalue tolerance
eigen_norm	*string*	mass	"visualization" or "unit"
energy_time_step	*int*	1	input of energy data
energy_exo_var	*string*	TEMP	**Exodus** energy variable name
Info	*int*	1	screen output control
MatrixFloor	*Real*	0	control of matrix fill
MaxmpcEntries	*int*	$10,000$	maximum # entries in any mpc
MaxResidual	*Real*	$1.e-6$	maximum residual for eigen
MortarMethod	*string*	dual	dual or standard mortar method
MFile_format	*string*	sparse_function	control output format for MATLAB
NegEigen	*none*		negative eigenvalue flag
nonlinear_default	*yes/no*	yes	nonlinear element blocks
num_rigid_mode	*int*	0	number of system rigid body modes
output_sideset_data	*bool*	false	Output element values on all sidesets
RandomNumberGenerator	*string*	"rand"	or "test"
RbmTolerance	*Real*	1e-10	tolerance for rigid body zero
RemoveRedundancy	*yes/no*	yes	filter constraints
reorder_Rbar	*none*		constraint reordering flag
SkipmpcTouch	*none*		control of MPCS
TangentMethod	*string*	element	method of computing tangent
WtMass	*Real*	1	Mass multiplier

Table 4-2. – Available keywords in the Parameters section.

Only one **parameters** section is recognized in each file. The parameters and their meanings are listed below and in Tables 4-2 and 4-3. For reference, Table 4-4 are parameters occasionally used by developers, but not recommended for common use.

wtMass For the convenience of users of the English system of units, natural units of mass are units of force, **Sierra/SD** provides the wtmass parameter. It multiplies all input masses and densities. Each input deck block section defines the density of the material in the corresponding element block. If wtmass is specified, then the simulation proceeds exactly as if each element block density had been scaled accordingly. For example, the density of steel is $0.283\, lbs/in^3$, but "lbs" includes the units of

$$g = 386.4\text{in}/s^2.$$

Using a value of WtMass of 0.00259 (1/386.4), density can be entered as 0.283, the outputs

Keyword	Type	Default	Description
nodesets_with_disp	*list*		nodesets with prescribed displacements
thermal_time_step	*int*	1	input of thermal data
thermal_exo_var	*string*	TEMP	**Exodus** temperature variable name
mpmd_transfer_type	*Fuego/copy*		Which code transfer is
	SPARC/interpolation		occurring from
mpmd_transfer_sidesets	*string_list*		Surfaces on which
	string_list		transfer is occurring

Table 4-3. – Parameters section keywords for code coupling and hand-off.

Keyword	Arg	Default	Description
OutputInitialTime	*boolean*	false	write output at t=0
SharedAcousticStiffness	yes/no	no	share stiffness across processors
linear_solver_warn_factor	*Real*	10	GDSW warnings
linear_solver_bailout_factor	*Real*	100	GDSW fatal errors

Table 4-4. – 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.

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 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 4-5.

Table 4-5. – Some useful combinations of units.

length	mass	time	WtMass	density	force	modulus	internal 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/386.4
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

NegEigen Unconstrained structures have zero energy modes which may evaluate to small negative numbers due to machine round off. The eigenvalues and associated frequencies 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 [1]. To retain the negative eigenvalues in the output file, select parameter `NegEigen`.

output_sideset_data This option turns on sideset output for any requested element output. Currently, sideset output is only enabled for (centroid) volumetric stress and strain (sections 9.7.19 and 9.7.20).

eig_tol This is the tolerance used in the **eigen**5.8 solution method for eigenvalues. The default is machine precision. This parameter can sometimes be loosened to aid convergence. See Section 3.8.3 for details.

nonlinear_default In nonlinear transient dynamics or nonlinear statics, computing the fully nonlinear response of all the elements in the mesh can be computationally expensive. In some cases it is unnecessary. 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 block-by-block. And there is a block-level parameter described in Section 6.6.1 that selects the optional nonlinearities for specific blocks. Instead of entering this parameter for each block, `nonlinear_default` sets the default for all blocks. If `no`, then all blocks default to linear behavior, unless specified otherwise in the Block section. If `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 6.6.1, 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 5.20 or transient dynamics 5.21). There are cases when it is better to use a tangent matrix computed from finite difference methods. There are 3 options for the TangentMethod.

element exact (assuming linear materials)

difference finite difference

compare Use the standard method, but also compute the matrix by the difference method. Output of the difference of every element matrix in the model will be sent to the results file [2]

[1]Because many post-processing 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 eigenvalues are also stored as global variables in the file

[2]In parallel solutions the results file is written only for the first processor unless the “subdomains” option is specified in the echo section (9.9).

info Option `info` selects diagnostic information for standard out. There are four different levels. Each level increasingly allows more output to standard out. The GDSW option "prt_debug" overrides "info" for GDSW output. The four levels of control are:

0. Silent – output warnings and std error to the screen (untested)

1. Normal – output the data most analysts would use (untested)

2. Detailed – Convergence, solution addressing issues (unimplemented, tested).

3. Debug – Silent, normal, detailed and diagnostic information (tested).

Info generates diagnostic information for eigenvalue problems. Setting it to 0 suppresses this output.

```
PARAMETERS
  info=0
END
```

SkipmpcTouch **Sierra/SD** uses a unique method of determining an active degree of freedom set. Unlike codes like NASTRAN which use an automatic single point constraint 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 auto-spc, 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 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 disabled (see the `ElemQualChecks` parameter in the **outputs** Section 9.8.3). The default value is 1e6.

The `condition_limit` parameter is ignored if the `ElemQualChecks` option has been disabled in the `output` section (9.8.3).

reorder_Rbar An `Rbar` is a type of rigid element. This option reorders all `Rbar` elements to minimize the number of them connected to a single node. Having may `Rbar` elements connected to the same node results in a problematic matrix sparsity pattern which can lengthen run time.

If redundant `Rbar`s are present (i.e. connections forming a cycle), they are removed by default. To not reorder the `Rbar` pseudo-elements, specify `reorder_Rbar no`.

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 10^{th} 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 temperature. Either energy density or temperature can be input, but not both.

thermal_exo_var If a material specifies a coefficient of thermal expansion and a reference temperature, then the corresponding thermal strain is computed, and the corresponding thermal load 8.3.7 is applied. Otherwise, a material property may be an arbitrary function of temperature. Temperature dependent materials 6.5.6 are supported. Temperature dependent viscoelastic materials 6.3 are also supported, using the corresponding input syntax. In either case to read the temperature from the input **Exodus** file, the name of the temperature field has to be specified. And that is what `thermal_exo_var` is for. The default variable name is 'TEMP'. In the following example the name is changed to temperature.

```
PARAMETERS
  thermal_exo_var "temperature"
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 temperature. 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
```

mpmd_transfer_type Defines which code **Sierra/SD** is coupling to during the run. To use Fuego output, options copy and Fuego are equivalent and one of them must be used. And similarly to use output from SPARC, the options interpolation and SPARC are equivalent and one of them must be used. See Section 8.2.4.

mpmd_transfer_sidesets Defines the sidesets on which coupling are occurring. When data is mapped from one code to the other, the transfer will only occur on these sidesets. See Section 8.2.4.

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 default is $1e-10$. RbmTolerance provides user control of the threshold for that error. It depends on the stiffness matrix, K, and a rigid body vector, ϕ_r,

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

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.

defaultSpecificHeat The specific heat is used to convert energy to temperature with the following equation, $\tilde{E} = C_v \Delta T$, see Section 6.5.8 for details. The specific heat is set on a material by material basis. A default value for the specific heat can be set using the keyword `defaultSpecificHeat` which can then be overridden by defining the specific heat for a material. A fatal error is given if the specific heat is not defined for a block of material containing energy input and the `defaultSpecificHeat` keyword can be used to avoid this.

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.

eigen_norm Eigenvectors may be arbitrarily normalized. Three common approaches are listed in Table 4-6. 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

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

Table 4-6. – Eigenvector Normalization Methods.

that the largest value of the eigenvector is one. [1] A global variable, *EigenVectScale,* provides the scale factor by which the mode was scaled.

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 eliminate some or all the rigid body modes as worked out in section "Orthogonality of MPC to Rigid Body Vectors" in the chapter on Linear Algebra Issues of the /sd 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 4-2 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]

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

Figure 4-2. – Example MFile Format Results.

[1]The "unit" method of normalization computes max(ϕ), 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 eigendecomposition, 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.

RemoveRedundancy RemoveRedundancy affects node-face constraints created by 'Tied Data' or 'Contact Definition'. Redundant constraints cause most solvers to fail. Redundant constraints are often introduced when two surface pairs are tied next to each other, but 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 tied to more than one face or if the node shows up both as a node of a node-face constraint and attached to a face of a different node-face constraint. 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.

MortarMethod Two mortar methods are available in **Sierra/SD**: standard and dual (see[39]). 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 modal 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.

DoInitialMassSolve Determining a nonzero the initial acceleration, for example to preserve quadratic convergence,[29] requires an initial mass solve section 5.28. If the linear system is ill-posed, the option to skip it is,

```
DoInitialMassSolve=false
```

OutputInitialTime In transient simulations, the output can be written at time t=0, prior to the first time step by using the command

```
OutputInitialTime=true
```

By default, it is false, and the initial time step is not output if reading from a restart file because the time is not 0.

linear_solver_warn_factor Although GDSW table 4-10 approximates the solution u of each linear system, $Au = f$ u_k such that,

$$\|f - Au_k\| < \text{solver\_tol}\ \|f - Au_o\|,$$

a predicted value of the residual is used. The true residual may be larger and a careful analyst must check the true residuals. To automate this checking task, a warning message is written if the true residual is more than *linear_solver_warn_factor* (default 10) times the predicted residual.

```
linear_solver_warn_factor=10.0
```

linear_solver_bailout_factor Like the softer GDSW warning factor 4.4, the bailout factor (default 100) is used to ensure that the true linear solver residuals are below the relative residual norm threshold. Once a residual norm exceeds this larger bailout threshold, the whole simulation ends with a fatal error. In an ill-conditioned problem as the `solver_tol` is decreased to improve resolution, increasing the bailout factor to mitigate the risk of a fatal error is recommended.

```
linear_solver_bailout_factor=100.
```

nodesets_with_disp This parameter, when used in conjunction with the **nUpdateConstraints** transient option (see section 5.28), enables specifying prescribed displacements over a subset of all nodes via nodeset output.

nodesets_with_disp is currently BETA release.
Enable with the "- -`beta`" command-line option.

4.4.1. *Suppression of Warnings*

There are several parameters that control the suppression or modification of errors and warnings. They are summarized below.

ignore_gap_inversion Initial overlap removal is another name for gap removal. Gap removal may change element quality. A fatal error occurs if element quality gets much worse. To ignore this poor element quality set the **ignore_gap_inversion** parameter to true. The gap removal solution case 5.32 making debugging easier.

syntax_checking **Sierra/SD** has the ability to check an input deck 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 disabled completely.

The three levels of control are:

ignore silently ignores those entries.

warn provides a warning for those entries.

Keyword	**Type**	**Default**	**Description**
AllowInvalidExodusParts	*bool*	false	Ignore error due to requesting an invalid **Exodus** part (e.g. sideset/block)
ignore_gap_inversion	*bool*	false	Ignore element quality changes due to gap and overlap removal
syntax_checking	*string*	fatal	“ignore”, “warn”, or “fatal”
RequireMatchedBlocks	*string*	fatal	“ignore”, “warn”, or “fatal”
reserved_keywords	*string*	fatal	“ignore”, “warn”, or “fatal”
restart_consistency_checking	*string*	fatal	“ignore”, “warn”, or “fatal”
restart_input_checking	*string*	fatal	“ignore”, “warn”, or “fatal”
restart_file_checking	*string*	fatal	“ignore”, “warn”, or “fatal”
WarningLevel (beta capability) ⚠	*string*	all	“all” $\longrightarrow$ show <u>all</u> errors “warning” $\longrightarrow$ <u>warnings</u> & above (fatal) only “fatal” $\longrightarrow$ <u>fatal</u> errors only “none” $\longrightarrow$ show <u>none</u> / disable all
exit on fatal (beta capability) ⚠	*bool*	true	false $\longrightarrow$ treat fatal errors like warnings
ignore error (beta capability) ⚠	*regex*	N/A	regular expression or string matching messages are ignored
error detail	*string*	normal	“brief” $\longrightarrow$ only include essential error information “normal” $\longrightarrow$ the default “verbose” $\longrightarrow$ include more detailed error information

Table 4-7. – Parameters for controlling error messages.

fatal stops the analysis. This is the default.

RequireMatchedBlocks **Sierra/SD** always requires that each **Exodus** block have a corresponding entry in the `.inp` file. This parameter controls behavior if there are blocks defined in the `.inp` file that do not appear in the mesh.

ignore silently ignores those entries.

warn provides a warning for those entries.

fatal stops the analysis. This is the default.

reserved_keywords **Sierra/SD** has the ability to check exodus entities for potentially ambiguous or confusing names. 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 disabled completely. Note that this option will only control the behavior when checking ambiguous names: we will always issue a warning if certain reserved characters are found within a name. For more information, see section 4.2.4.

The three levels of control are:

ignore silently ignores those entries.

warn provides a warning for those entries.

fatal stops the analysis. This is the default.

restart_consistency_checking by default **Sierra/SD** will check that the restart data file is consistent with the analysis mesh. These checks include confirming identical node ids, element id, node coordinates, and element connectivity. By default, if the restart file is inconsistent **Sierra/SD** will output a fatal error and terminate execution. Generally such restart inconsistencies indicate a serious problem. For example trying to map the restart data from the wrong analysis onto a new mesh will produce nonsensical behavior. Use `restart_consistency_checking` to convert these checks to a warning or skip them.

The three levels of control are:

ignore silently ignores restart consistency issues.

warn provides a warning for restart consistency issues.

fatal(default) stops the analysis for restart consistency issues.

restart_input_checking when restarting, **Sierra/SD** will verify that the input deck is consistent with the input deck that generated a restart data file (as recorded in the *info records* of the restart **Exodus** file). If it is not, **Sierra/SD** will output a fatal error and terminate execution. Whitespace and comment characters are not considered in this comparison.

The following sections are also ignored from this comparison:

- `SOLUTION`
- Solver options (`GDSW`, `SOLVER_OPTIONS`, `CAMP`, `PARDISO`)

- Output options (ECHO, HISTORY, OUTPUTS, STATISTICS, USER OUTPUT)

New sections of the following type are also ignored. However, modification or deletion of sections from the original restart input is not allowed.

- Coordinate systems
- DAMPING and FREQUENCY BAND DAMPING
- FUNCTION
- FREQUENCY
- LOAD/LOADS
- MATRIX-FUNCTION
- RANLOADS
- TABLE

For all other sections, no modification, addition, or deletion is allowed.

Use `restart_input_checking = warn|ignore` to convert this check to a warning or skip it entirely. As with `restart_consistency_checking`, restart inconsistencies generally indicate a serious problem, so great care should be taken if this error is ignored.

restart_file_checking When attempting to restart a solution with `restart = read` (section 4.6.2), **Sierra/SD** will first verify that the restart data file exists. Use `restart_file_checking` to control the behavior when the file is not found:

ignore silently restart from scratch.

warn warn and restart from scratch.

fatal (default) warn and stop the analysis.

4.4.1.1. Warning Suppression Examples

The below examples will go over examples of some common errors in the input deck, their typical error messages, and the effect of various control parameters listed above.

In the following example, a force load was only partially defined.

```
LOADS
  body
    force 1e7 0 0
  block 1
    force 1e7 0 0
  sideset sideset_1
    traction 1e6 0 0
  sideset 2
```

```
    force 1e5
END
```

This yields the following default error message, and the simulation is abandoned.

```
*************************************************************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**
**  ==== Parsed input lines: =======================================================
**  LOADS
**  body
**  force 1e7 0 0
**  block 1
**  force 1e7 0 0
**  sideset sideset_1
**  traction 1e6 0 0
**  sideset 2
**  force 1e5    <-- FAILED HERE
**
**  ==== Valid lines in this context are: ==========================================
**          force = <real> <real> <real>
**                      --- reached end of block, expecting to find a <real>
**
*************************************************************************************
```

To avoid printing the parsed input lines, use `error detail = brief`.

```
*************************************************************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**  force 1e5    <-- FAILED HERE
**
**  ==== Valid lines in this context are: ==========================================
**          force = <real> <real> <real>
**                      --- reached end of block, expecting to find a <real>
**
*************************************************************************************
```

There are several ways to continue with the simulation after encountering fatal errors. However, use extreme caution when using them, as fatal errors are typically fatal for a good reason! These options are roughly ordered from least to most dangerous due to their potential to suppress other unrelated errors and warnings.

- To ignore this **specific** error, use `ignore error = "force 10"`, `ignore error = "force = <real> <real> <real>"`, or indeed any string or regular expression from the message.
- To convert **all syntax** errors to warnings (continue simulation, but keep message), use `syntax_checking = warn`.

- To continue after **all** errors (keeping the message), use `exit on fatal = false` (this is the default in "–check-syntax" mode).
- To ignore **all syntax** errors and warnings, use `syntax_checking = ignore`.
- To ignore **all** errors and warnings, use `WarningLevel = none`.

In the next example, an incorrect sideset was used to define a loading region. As before, using `error detail = brief` will suppress printing the parsed input lines. Additionally, in this case it will also suppress printing of the valid sidesets on the mesh.

```
LOADS
  body
    force 1e7 0 0
  block 1
    force 1e7 0 0
  sideset sideset_3
    traction 1e6 0 0
  sideset 2
    pressure 1e5
END
```

Input 4.6. Input with error

```
**********************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**  sideset sideset_3   <-- FAILED HERE
**
**  ==== Valid lines in this context are: ===
**  sideset = <sideset>
**  -- EXPECTED "sideset_3" to be a <sideset>
**
**********************************************
```

Output 4.1. `error detail = brief`

```
**********************************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**
**  ==== Parsed input lines: ============================
**  LOADS
**  body
**  force 1e7 0 0
**  block 1
**  force 1e7 0 0
**  sideset sideset_3   <-- FAILED HERE
**
**  ==== Valid lines in this context are: ===============
**  sideset = <sideset>
**  -- EXPECTED "sideset_3" to be a <sideset>
**
**  The following sidesets have been defined in the mesh:
**    sideset name   sideset id
**    ------------   ----------
**    surface_1      1
**    surface_2      2
**
**********************************************************
```

Output 4.2. `error detail = normal` (default)

All of the previously-outlined ways to continue with the simulation after encountering fatal errors are valid in this context as well. However, you can also use `AllowInvalidExodusParts = true` to specifically ignore errors related to invalid exodus parts while keeping others.

In the following example, the 1st keyword of a line is misspelled, so a fuzzy match is used to determine the most likely intent. If the complete list of syntax is desired, use `error detail = verbose`. Examples of the resulting error message with all 3 options of `error detail` are shown below.

```
LOADS
  body
    force 1e7 0 0
  block 1
    force 1e7 0 0
  sideset sideset_1
    traction 1e6 0 0
  sdeset 2
    pressure 1e5
END
```

Input 4.7. Input with error

```
************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**  sdeset      <-- FAILED HERE
**
**  ==== Closest matches are: ======
**  sideset = <sideset>
**
************************************
```

Output 4.3. `error detail = brief`

```
************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**
**  ==== Parsed input lines: =======
**  LOADS
**  body
**  force 1e7 0 0
**  block 1
**  force 1e7 0 0
**  sideset sideset_1
**  traction 1e6 0 0
**  sdeset      <-- FAILED HERE
**
**  ==== Closest matches are: ======
**  sideset = <sideset>
**
************************************
```

Output 4.4. `error detail = normal` (default)

```
********************************************************
**  FATAL ERROR:
**  LOADS... Invalid syntax:
**
**  ==== Parsed input lines: ===========================
**  LOADS
**  body
**  force 1e7 0 0
**  block 1
**  force 1e7 0 0
**  sideset sideset_1
**  traction 1e6 0 0
**  sdeset <-- FAILED HERE
**
**  ==== All valid lines for this section are: =========
**  loads
**  block = <block>
**      angular_velocity = <real> <real> <real>
**      angular_acceleration = <real> <real> <real>
**      force = <real> <real> <real>
**      iforce = <real> <real> <real>
**      moment = <real> <real> <real>
**      imoment = <real> <real> <real>
**      traction = <real> <real> <real>
**      itraction = <real> <real> <real>
**      acoustic_accel = <real>
**      acoustic_vel = <real>
**      iacoustic_accel = <real>
**      iacoustic_vel = <real>
**      ipoint_volume_accel = <real>
**      ipoint_volume_vel = <real>
**      ipressure = <real>
**      lighthill = <real>
**      point_volume_accel = <real>
**      point_volume_vel = <real>
**      pressure = <real>
**      energy_load
**      iplane_wave
**      modalforce
**      plane_wave
**      randompressure
**      thermal
**      thermal_load
... (250 more lines of output)
**      heat_flux = <real>
**      iheat_flux = <real>
**      heat_volumetric = <real>
**      iheat_volumetric = <real>
**  +/- ...(<real>) (%|percent)  <-- DEPRECATED (5.14)
**
********************************************************
```

Output 4.5. `error detail = verbose`

Finally, we consider an example where a boundary region is defined but no boundary condition is specified. In this and similar cases, the default behavior is to output all lines of syntax that are valid in the current context, which in some cases can be quite long. In contrast, `error detail = brief` will output a truncated list.

```
BOUNDARY
  sideset 1
    fixed
  sideset 2
    # missing
END
```

Input 4.8. Input with error

```
************************************************
**  FATAL ERROR:
**  BOUNDARY... Invalid syntax:
**
**  = One of the following must be specified: =
**  AccelX = <real>
**  ...
**  pml_element use block <block> (hex)
**
************************************************
```

Output 4.6. `error detail = brief`

```
************************************************
**  FATAL ERROR:
**  BOUNDARY... Invalid syntax:
**
**  = One of the following must be specified: =
**  AccelX = <real>
**  AccelY = <real>
... (40 more lines of output)
**  slosh = <real>
**  infinite_element use block <block>
**  pml_element use block <block> (hex)
**
**  ==== Parsed input lines: ==================
**  BOUNDARY
**  sideset 1
**  fixed
**  sideset 2
**
************************************************
```

Output 4.7. `error detail = normal` (default)

4.5. Loads

This section introduces the input deck syntax for specifying one or more structural loads. More detailed descriptions of specific kinds of loads is presented in the Boundary Conditions section 8.3.

Loading conditions are specified within the **load** or **loads** section. The following example illustrates several types of loading conditions.

```
LOADS
  nodeset 3
    force = 1.0 0. 0.
    scale = 1000.
    function = my_load_function
  nodeset nose
    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
  block wing
    thermal_load
    function = 1
  block 12
    thermal_load
    function = 3
  sideset 7
    pressure 15.0
  sideset tail
    traction = 100.0 20.0 0.0
    coordinate my_coord
  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 are applied to sections of the mesh such as node sets, side sets, blocks, node lists (see Section 8.1), or the entire body. Certain loads can only be applied on certain parts of the mesh, for example pressure and traction loads can only be applied on sidesets. All loads applications are additive.

The components of each load specification are listed in Table 8-6. The syntax followed is to first define the region over which the load is to be applied (either **nodeset**, **sideset**, **block**, **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 region.

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 4.9).

4.5.1. The Load and Loads Sections

There is a subtle distinction between a **load** input deck section and a **loads** section. A **load** section must be given a name and that load section can then be applied in one or more solution cases. See Section 5.1.1 for information on **load** specifications for multicase solutions. On the other hand a **loads** section has no name and is automatically included in all the multicase solutions that don't explicitly specify a load. If a **load** is used in a solution case and a **loads** section also exists the **loads** will be ignored for the solution case. Note there are some solution cases where load can only be applied via the global **loads** input. Additional some solution cases require a load case to be set, even if that load case is empty.

The following example illustrates the input. Note, the load the identifier ('57' in this case) is often an integer but any unique string is allowed.

```
LOAD 57
  nodeset 3
     force = 1.0 0. 0.
     scale = 1000.
     function = 2
  nodeset 5
     force = 0. -1 0
END
LOADS
  body
    gravity 0 1 0
    scale = -9.8
END
```

At most one **loads** section can be specified for and analysis but there may be multiple **load** sections in the file.

4.5.2. *Scale Factors for the Load*

Several factors define the actual load value ultimately applied:

- load type magnitude, for example **pressure** = 5
- overall scale factor, for example **scale** = 10
- load function, for example **function = myFunc** note, there are a wide variety of function types that can define values of load that are variant on time, space, frequency, or other factors.
- 'distribution factor' defined on the mesh. The distribution factor is an advanced feature used to introduce the spatial variation of a quantity in the Exodus mesh file. Distribution factors can be difficult to check, and in some cases if entered as zero in the input mesh would zero out the load. Use the Boolean parameter **AllowExodusDistFacts** to ignore Exodus distribution factors.

These factors are applied multiplicatively.

4.6. Solution Options

Sierra/SD provides over 50 different analysis types, each described in the largest Chapter 5 of this Manual. This section describes the corresponding options used to configure these analyses.

The options described in Table 4-8 and in the following paragraphs are part of the input deck `Solution` section. None of the keywords are required. Note that in multicase solutions most of these parameters may be applied separately within each case (see section 5.1.2).

Table 4-8. – **Sierra/SD** Solution Options.

Parameter	Type	Default	Description
restart	none|read|write	none	Controls restart input and output. See section 4.6.2 for more info.
lumped		off	Lump the mass matrix (as opposed to consistent)
lumped_consistent		off	Mix lumped and consistent matrix for reducing dispersion error in some cases
solver	<string>	auto	Select solver to use for case, default depends on model size and structure
ConstraintMethod	Lagrange penalty	Lagrange	Method for applying MPCs
scattering	false| true	false	Treat acoustic loads as scattering loads rather than incident loads
symmetrize_struct_ _acous		off	Force structural-acoustic system matrices to be symmetric

4.6.1. Flush

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.

flush=N flushes every $\mathtt{N}^{th}$ time steps.

4.6.2. Restart

The `restart` option controls both the creation and retrieval of restart files, which allow you to save and resume a solution. There are two main types of analysis that can use restart: transient and eigen. Transient restart saves the current state of a transient solution in a separate file, enabling you to save progress and continue solving for more time steps at a later time. Eigen restart saves computed modes and enables the calculation of additional modes later on.

The restart command has four options:

none : Ignores and doesn't read or write restart files. Existing files remain unchanged. This is the default if no restart option is specified.

read : Reads existing restart files.

write : Writes restart files, overwriting any previous ones.

auto : Combines "read" and "write", where previous restart files are optional. If restart files exist, they are read. If no restart files exist, then the solution case will start from scratch. **Note**: this option has been deprecated. Use a combination of `restart = read`, `restart = write`, and the `restart_file_checking` parameter instead.

The default file names of restart files to read and write are generated based off of a combination of mesh name and case name. The restart `read` option can take optional `from` and `num_procs` clauses to define a non-default location for the file containing the restart data. Use the **restart_file_checking** parameter (section 4.4.1) to control the behavior when the restart file is not found.

This is illustrated by additional examples for both transient and Eigen restart below.

4.6.2.1. Restart In Transient Analysis

For transient restart **only**, `restart = write` can take optional `to` clause to define a non-default write location. **Note**: `restart = write to <file>` is only valid for transient restart - any write location will be ignored for eigen restart.

```
SOLUTION
  case First399Steps
    transient
    restart=write
    time_step 1e-6
    nsteps 399
END

FILE
  geometry_file='mesh.g'
END
```

A Transient analysis with option write, restarts extend the simulation time. The input deck at left is for a 399 time step simulation that writes a restart file. `mesh-a.rst_trans`.

```
SOLUTION
  case Steps400to510
    transient
    restart=read
    time_step 1e-6
    nsteps 101
END

FILE
  geometry_file='mesh.g'
END
```

Input deck at left is used to read the default restart file `mesh-a.rst_trans`, then simulate 100 more steps. Time steps 201 to 310 are appended to the existing exodus output and history files.

The next example modified the previous example to use non-default restart file names. The initial 200 steps simulation writes the restart file `first-a.rst_trans` and the output file `first-a.e`. The second input deck utilizes the **from** command to read `first-a.rst_trans`, simulate time steps 400 to 500, and outputs to the new file `second-b.e`.

```
SOLUTION
  case First200Steps
    transient
    restart=write
    time_step 1e-6
    nsteps 200
END

OUTPUTS
  database name = ''first.e''
END
```

```
SOLUTION
  case Second100Stesps
    transient
    restart=read from ''first-a.rst_trans''
    time_step 1e-6
    nsteps 100
END

OUTPUTS
  database name = ‘‘second.e’’
END
```

Example 3: The first input deck, run in parallel on two processors, for 140 steps and writes restart files named

`first-a.rst_trans.2.0` and `first-a.rst_trans.2.1`, as well as output files `first-a.e.2.0` and `first-a.e.2.1`. The second input deck, run on a different number of processors (three), reads `first-a.rst_trans.2.0` and `first-a.rst_trans.2.1`, solves an additional 160 steps, and outputs to new files `second-b.e.3.0`, `second-b.e.3.1`, and `second-b.e.3.2`.

```
SOLUTION
  case Restartable
    transient
    restart=write
    time_step 1e-6
    nsteps 140
END

OUTPUTS
  database name = ‘‘first.e’’
END
```

```
//Input deck 2
SOLUTION
  case b
    transient
    restart=read from ‘‘first-a.rst_trans’’ num_procs 2
    time_step 1e-6
    nsteps 300
END

OUTPUTS
  database name = ‘‘second.e’’
END
```

The frequency of writing transient output restart files is controlled by the **flush** command. The transient restart file is typically saved with the last two steps as a backup in case one becomes corrupted.

Transient restart is expected to result in exact solutions. For example, if an intermediate restart step is used in the example, it will generate the identical solution steps as if the total 300 steps were run in a single analysis.

When restarting a multicase solution, the latest allowable restart time will be used. Consider the following solution block:

```
SOLUTION
  case one
    transient
    restart=read
    restart=write
    time_step 1e-6
    nsteps 140
  case two
    transient
    restart=read
    restart=write
    time_step 1e-5
    nsteps 300
END
```

Sierra/SD will search for restart files that would be valid for case one and case two. The furthest along available restart file will be used and the analysis will continue from that point. If a restart file exists for case two, **Sierra/SD** will restart into case two. If no restart file exists for case two, **Sierra/SD** will restart into case one if possible. If no restart file exists for either case, the analysis will start from scratch (depending on the value of the `restart_file_checking` parameter).

4.6.2.2. Restart in Eigen

The Eigen restart function saves and reads the computed modes from an exodus database, which is created by the OUTPUTS command block. If the restart command is set to **write** in the Eigen solution, displacement output will be automatically activated in the OUTPUTS block to make it possible to use the output file for restarting.

Example 1: the first input deck solves for 40 modes and writes the output to a file named `mesh-a.exo.` The second input deck then reads this file and solves for an additional 10 modes, resulting in 50 modes. The modal output of the second run will be added to the existing exodus output and history files.

```
//Input deck 1
SOLUTION
  case a
    eigen
    restart=write
    nmodes 40
END

FILE
    geometry_file = 'mesh.exo'
END
```

```
//Input deck 2
SOLUTION
  case a
    eigen
    restart=read
    nmodes 50
END

FILE
    geometry_file = 'mesh.exo'
END
```

Example 2: The first deck will solve for 20 modes and write an output file named `first-a.e`. The second deck will read the previously computed 40 modes from `first-a.e` utilizing the **from** option and solve an additional 20 modes. A new output file `second-b.e` will be created which contains all 60 modes.

```
//Input deck 1
SOLUTION
  case a
    eigen
    restart=write
    nmodes 40
END

OUTPUTS
  database name = ``first.e''
END
```

```
//Input deck 2
SOLUTION
  case b
    eigen
    restart=read from ``first.e''
    nmodes 60
END

OUTPUTS
```

```
  database name = ''second.e''
END
```

Example 3: Eigen restart also allows for N to M restart with the optional **num_procs** command. This enables reading of an Eigen restart file that was decomposed for a different number of processors. N to M restart is useful when the first Eigen solution was computed using many processors but the subsequent solution case, such as modaltransient, can be computed using a smaller number of processors. This can be beneficial because the Eigen solution is computationally intensive and memory-intensive, while modal transient is relatively inexpensive. An example of N to M restart is shown below where the first input deck is run using 100 processors, and the second input deck reads the previously computed Eigen solution and then moves on to modal transient on 5 processors.

```
//Input deck 1
SOLUTION
  case eig
    eigen
    restart=write
    nmodes 75
END

OUTPUTS
  database name = ''beam.e''
END
```

```
//Input deck 2
SOLUTION
  case eig
    eigen
    restart=read from ''beam-eig.e'' num_procs 100
    nmodes 75
  case trans
    modaltransient
    nsteps 1000
    time_step 1.0e-6
END

OUTPUTS
  database name = ''beam.e''
END
```

Eigen solution is a memory and computationally intensive process, and the output/restart files are written only after the solution is completed. Simulations that exhaust their queue time do not provide restart files.

While Eigen restart allows for computing additional modes, it may result in slight differences from solving for all modes at once, as it follows a different algorithmic path. The known modes are first read and compressed out of the system, and then additional modes are computed on this compressed system.

Incrementally computing a few more modes with multiple Eigen restart is not recommended, as it can result in lower numerical accuracy compared to solving for all modes at once.

4.6.2.3. Usage Tips and Guidelines

Setting the **restart_file_checking** parameter to `WARN` or `IGNORE` can be useful when running the same input deck repeatedly in the same directory to generate additional modes or transient time steps. However, care should be taken as certain typos, like renaming a case, may have unexpected effects like a previous restart being unrecognized triggering an expensive solution computation.

It is not advisable to alter the model's input deck, such as its material properties, element formulations, boundary conditions, contacts, constraints, etc., during a restart as the outcomes can be unpredictable. For instance, if the material properties are modified during an Eigen restart, then the

- Eigen modes linked to the previous material properties will be loaded,
- Eigen modes are compressed from the matrices, and
- New modes will be calculated using the updated material properties.

The results of this process are likely to be both physically and mathematically incorrect.

Restart files are stored in the ExodusII format, which enables easy access and manipulation of the data using various standard tools. The naming conventions and formats used for restart files are described in Table 4-9.

4.6.2.4. Restart Solution Case Support and Limitations

Only the following solution cases support restart.

- Eigen
- transient
- NlTransient
- modaltransient
- QEVP (with Anasazi and sa_eigen methods)

Modal simulations, such as Modalfrf, use a restarted Eigen solution.

Mixing case types in transient cases, such as using a restart file from a modaltransient analysis in a transient or NlTransient analysis, is possible but may result in information loss and mapping ambiguities, and is not typically recommended.

The restart options for QEVP solution are limited, as it can only read already computed modes but cannot calculate new modes incrementally.

Solution	file name	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.
transient, NlTransient, modaltransient	*example-out.rst_trans.exo*	The two most recent time steps are written. They are only written at the “flush” interval.

Table 4-9. – Restart file format and contents for various solutions.

4.6.3. 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.

auto Use the best known solver. Generally this is recommended, and is the default. The matrix of solvers versus solution types is messy, and generally the best solution will be found by using this option.

GDSW The Generalized Dryja, Smith, Widlund (GDSW) solver is based on a domain decomposition preconditioner which combines overlapping Schwarz and iterative substructuring concepts.[19] The GDSW solver is well suited to solving problems with large numbers of constraint equations. It has also been observed to be 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.

Generally no user input is required for specification of a solver. Usually the specification can be omitted or specified as `auto`. If a solver is requested and unavailable in a given version of the code, 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 5.1.2), or it may be individually specified within the case framework. The default value is `auto`. In the example shown below GDSW will be used for the modal analysis, and the `auto` selection for the for transient dynamics and direct frequency response. If “input_summary” is specified in the “ECHO” section (see section 9.9) then the solver information will be echoed to the results file.

```
SOLUTION
 solver=auto
 case eigenvalue
   eigen
   nmodes=50
   solver=gdsw
 case NlTransient
   NlTransient
    time_step 1e-6
    nsteps 400
 case frf
   directFRF
END
```

4.6.4. *Lumped – option*

To use a lumped mass matrix for all solution cases add option `lumped` anywhere in the `Solution` section. Most off diagonal terms are set to zero, and the corresponding the diagonal terms are increased to conserve mass.

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 3×3 matrix.

For acoustic simulations Hex8 elements are recommended due to the discovery of the mass matrix minimizing dispersion error[15] corresponding to the Hex8 stiffness matrix. It is selected by adding `lumped_consistent` to the `Solution`.

This mass matrix is conjectured to also enhance structural simulations. The CFL number c dt/dx depends on the sound speed, c, the element size, dx, and the time step size, dt. A study[42] of the `lumped_consistent` mass matrix reached the following conclusions.

- In the most accurate simulations, time step size is scaled with mesh size to maintain a constant CFL number.
- Simulation accuracy depends on the temporal (e.g. Newmark) as well as the spatial (finite element) discretization.
- A `lumped_consistent` mass matrix can enhance accuracy.

- With Newmark Beta time integration, the most accurate simulations used a consistent mass matrix and a CFL near 7/10.

4.6.5. *Constraintmethod – option*

The `ConstraintMethod` option is defined in the `Solution` section to indicate how multipoint constraints (MPC) will be applied. MPCs applied using `Lagrange` multipliers. Inverse problems sometimes use an experimental penalty method.

4.6.6. *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. A review is presented in subsubsection Acoustic Scattering subsection Acoustic and Structural Acoustic Boundary Conditions section Acoustics and Structural Acoustics of the Theory Manual 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. A function tailored for this specific purpose may be used. [1]

4.6.6.1. symmetrize_struc_acous – option

By default, coupled structural acoustic discretizations are symmetric. This is accomplished[45] by scaling the acoustic equation by a -1. In some cases 8.1.9 scaling is impossible, and cases the code internally reverts to the nonsymmetric formulation. To use the nonsymmetric formulation, set **symmetrize_struc_acous** to false. In a multicase solution **symmetrize_struc_acous** can vary from case to case. The pros and cons of two approaches have never been studied.

4.7. GDSW

GDSW is the workhorse for parallel solutions. It is the default linear solver. Many **Sierra/SD** features require that GDSW be the linear solver. In this manual, "the solver" is the linear solver. This section describes the GDSW parameters. Table 4-10 describes the basic solver parameters. Parameters for advanced usage are given in Tables 4-11 and Table 4-12. Report problems using

[1] 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 4.10.10, 4.10.12 and 4.10.14.

the GDSW solver with the default solver parameters to the Sierra help system at `sierra-help@sandia.gov`.

To use a non-default linear solver in the solution section set `solver=GDSW`. Non-default parameters are set in an optional GDSW section. The GDSW section applied to each solution case. The first subsection of this Section 4.7.1 presents a way to specify different GDSW parameters for each solution case. Subsection 4.7.1 describes additional parameters. Subsection 4.7.2 describes GDSW output. The corresponding parameters are summarized in Table 4-13. Subsection 4.7.3 describes some techniques for improving performance and reliability. Subsection 4.7.4 reviews theory related to solution accuracy. The GDSW Helmholtz solver is a relatively new capability. Subsection 4.7.5 presented the parameters for the parallel frequency response linear solver. The parameters are summarized in Table 4-16. Report any problems using the new Helmholtz solver to `sierra-help@sandia.gov`.

Table 4-10. – GDSW Section Options. (Basic)

Variable	Values	Default	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 (CPU execution) or 5000 (GPU execution) or 0 (serial execution)	number of stored search directions used to accelerate solver convergence (see also num_vectors_keep)
prt_summary	*integer*	3	output flag: 0 - no summary 1 - summary 3 - detailed summary

useParallelCoarseSolver is currently BETA release.
Enable with the "`--beta`" command-line option. The option of using a parallel coarse solver is considered beta because of the complexity of the user interface.

solver_tol The iterative linear solvers, including GDSW, approximate the solution u of each linear system, $Au = f$ by the determining a sequence of approximate solutions, $\{u_k\}_{k\geq 0}$, and converging to u_k such that

$$||f - Au_k||_2 < \mathrm{solver_tol}||\mathrm{f} - \mathrm{Au_o}||_2.$$

solver_tol (default 10^{-6}) is the requested accuracy of the computed solution as measured by the relative this residual error. Also see the corresponding parameters 4.4, 4.4. If only one MPI rank is used, then the linear solver is direct, and this relative residual norm is reduced to the double precision $\epsilon \sim 10^{-16}$.

orthog Convergence may be accelerated by storing and recycling search directions from previous solves. This feature requires additional memory, but may significantly reduce iterations and speed up calculations considerably. It is possible for the application of these vectors to be unstable, in which case the **orthog** parameter may be reset to zero. The default value of the

Table 4-11. – GDSW Section Options 1. (Advanced)

Variable	Values	Default	Description
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
default_solver	*integer*	1	1-direct: Esmond Ng's sparse direct solver 3 - Pardiso for Pardiso sparse direct solver (Intel MKL only), 6-NoPivot: Clark's sparse direct solver
num_rigid_mode			note: see **parameters**, Section 4.4.
constrain_rbms	*x y z rotx roty rotz p*		Enables solution of static system with rigid body modes
max_numterm_C1	*integer*	250	max num terms for Type 1 constraints
coarse_option	*integer*	1	0 - additive coarse correction, 1 - multiplicative coarse correction
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 options auto, small, large
coarse_size	*integer*	0	0 (auto), 1 (small), 2 (large)
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*	2.5e-9	singularity tolerance for processing constraints
con_row_tolerance	*real*	0.1	pivoting tolerance for processing constraints
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 (in exact arithmetic) 1-full: full $\Phi^T A \Phi$ matrix
bailout	*keyword*		If keyword is found, ignore errors
coarsening_ratio	*integer*	1000	coarsening ratio for multilevel solver

Table 4-12. – GDSW Section Options 2. (Advanced)

Variable	Values	Default	Description
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
enforceActualResidual	*integer*	0	0-no/1-yes
graphPartitioner	*integer*	0	graph partitioner for multilevel solver 0-Parmetis, 1 PHG in Zoltan
num_vectors_keep	*integer*	900	related to orthog; see discussion below
stag_tol	*real*	0.01	Used to detect stagnation
dd_solver_output_file	*string*	dd_solver.dat	Output name for domain decomposition solver diagnostic file
krylov_solver_output_file	*string*	krylov_solver.dat	Output name for Krylov solver diagnostic file
useParallelDirectSolver	*integer*	0	0-no/1-yes
useParallelCoarseSolver	*integer*	0	0-no/1-yes
coarseDirectSolverNumProcs	*integer*	all	number processors for coarse problem
useSuperLUDist	*integer*	0	0-no/1-yes: whether to use SuperLU-Dist
numProcRowSuperLUDist	*integer*		number of rows in SuperLU-Dist process grid
identify_low_quality_elements	bool	false	identify poorly shaped elements for special attention by solver
max_element_condition	double	Infinity	condition above which to consider element low quality
reportZeroDiagonals	*integer*	0	0-no/1-yes

orthog parameter is $1,000$ on CPU-only **Sierra/SD** analyses, $5,000$ on GPU-accelerated analyses, and 0 on single-processor CPU-only runs.

The num vectors keep option may be related. The dd solver file ends with a break-down of the solve time into four distinct parts of the solution method. One of these is orthogonalization. Orthogonalization time is proportional to the value of orthog. In cases where more memory is available and orthogonalization time is insignificant, experiments with orthog are worthwhile. For the Helmholtz linear solver the corresponding parameter is orthogH.

krylov_method A variety of Krylov iterative methods are available as shown in Table 4-11. The default should work fine in most instances. If convergence problems arise, switching to classic right preconditioned GMRES is recommended. (krylov_method = gmresClassic) without the use of any stored search directions (orthog = 0).

default_solver The subdomain sparse direct linear solver is specified either by name (Esmond, Pardiso, NoPivot) or by the corresponding integer $(1, 3, 6)$.

num_rigid_mode This keyword must not appear in the GDSW solver. See **parameters**.

constrain_rbms Tells GDSW to numerically constrain rigid body modes of a structure to do a static solution of a free floating structure. Must be used in conjunction with the **FilterRbmLoad** in the solution case to ensure no net rigid body load on the constrained structure. The rigid body modes associated with `x`, `y`, `z`, `rotx`, `roty`, `rotz` can be selectively constrained, as can `p` the pressure degree of freedom in acoustic analysis. See Section 8.3.21 for more details on this use case.

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 have few terms, and Type 2 constraints have many terms in each 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. If feasible, avoiding Type 2 constraints by increasing the *max_numterm_C1* in conjunction with a multiplicative coarse correction (*coarse_option=1*) will lead to a more efficient solution.

Avoiding Type 2 constraints generally reduces run times. To eliminate Type 2 constraints set the GDSW option *max_numterm_C1* to a sufficiently large number. The value that GDSW is using for *max_numterm_C1* is in the dd_solverdat file. It is called *maxNumTermsForType1Constraints*. Also, in the `dd_solver.dat` file note the value of *maxNumNonZeros in Tran Matrix*. To eliminate Type 2 constraints *max_numterm_C1* must be larger than the value of *maxNumNonZeros in Tran Matrix*.

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 fewer than a few hundred processors. However, as the number of processors (subdomains) becomes large, solving the coarse problem can become a bottleneck. Currently, 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 often reduces 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 4-11).

num_GS_steps is the number of orthogonalization steps of the stored search directions. Sometimes increasing it from its default to 2 is helpful. I know of no case of a value larger than 2 being necessary. It turns out that `num_GS_steps` does not apply to gmresClassic.

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. The con_tolerance is the minimum acceptable ratio of the constraint pivot to the maximum matrix pivot. The con_tolerance can viewed as a dimensionless parameter associated with numerical round off. The number of numerically redundant constraints in a model will typically be reduced as the `con_tolerance` is increased. Excessively high con_tolerance values could start to remove legitimate constraints.

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. Additionally, a message will be written if there exists constraints on the cusp of being removed as redundant. Again presence of such nearly redundant constraints should be investigated closely for correctness.

Redundant or nearly redundant constraints can be generated in a variety of circumstances. For example a closed loop of Rbars, Rbars between nodes of a rigid set, or overlapping rigid sets. Although the SD constraint filtering algorithm is designed to detect and handle such redundancies it is strongly recommended that redundant constraint be avoided in model definition as much as possible. See Section 3.8.6 for more details.

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 changes the number of iterations for each solve of a parallel run since GDSW is 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.

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`.

enforceActualResidual This option is used to enable strict enforcement of the solver tolerance. If needed, iterative refinement steps are taken within the solver in an attempt to satisfy the convergence criterion. Decreasing the relative residual to values no greater than the solver tolerance may be impossible. This can happen when the problem is poorly conditioned and/or the requested solver tolerance is too small. An error message is issued and the analysis stops if the specified solver tolerance cannot be achieved.

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

num_vectors_keep applies to krylov_method gmresClassic. The parameter orthog (default 1000) controls the number of stored search directions. We store search directions to make the linear solver faster. More is not always better. The point to understand is which search directions are stored. The first 1000 search directions are stored. On later solves, the first 900 are saved and recycled. 100 search directions from the current solve are used. The number 900 is the default value of num_vectors_keep. Sometimes the solution has changed significantly and none of the old search help the solver. `num_vectors_keep`= 0 tells GDSW to never recycle any of the stored search directions between solves.

num_GS_step_gmres is the parameter for the Krylov method gmresClassic corresponding to the `num_GS_steps` for other Krylov methods.

useParallelDirectSolver Whether to use parallel sparse direct solver for the whole problem. Best suited to Direct FRF solution case.

useParallelCoarseSolver Whether to use a parallel sparse direct solver for the coarse problem. This is the best choice if the number of MPI ranks is in the range 512 to 2048.

coarseDirectSolverNumProcs Number of processors to use if `useParallelCoarseSolver = yes`. The default is all.

reportZeroDiags This can be an integer, or it can be set to yes or no. The default is no (or 0). If reportZeroDiags is set to yes, then the global node numbers corresponding to degrees of freedom in the coefficient matrix with zero diagonals (and typically zero rows and columns) are output in the file node_zero.dat. Preconditioning matrices with zero diagonals is more difficult. Also, a zero diagonal might indicate a modeling error. In either case, it can be useful to have more information about these nodes.

useSuperLUDist Whether to use SuperLU-Dist parallel sparse direct solver even if Cluster Pardiso is available. Applies to both the full problem (`useParallelDirectSolver` and coarse problem (`useParallelCoarseSolver`) cases.

numProcRowSuperLUDist SuperLU-Dist uses a rectangular grid of processes. This option allows the user to override the default choice, which is to make the number of rows about equal to the number of columns. The parameter is ignored unless it divides the total number of processors. It has been suggested that for large problems, the number of columns should be larger than the number of rows, but we have not yet tested cases where this choice proved to be beneficial.

useSuperLUDist is currently BETA release.
Enable with the "`--beta`" command-line option. Like useParallelCoarseSolver, this capability is beta due to the complexity of user interface.

The generalized conjugate residual (GCR) version of right preconditioned GMRES is the default Krylov method. GCR has the limitation that `orthog` must be at least as large as `max_iter`. On the other hand with gmresClassic `max_iter` and `orthog` may be specified independently.

4.7.1. Options

The `GDSW` section specifies GDSW linear solver parameters for all the solution cases. Different solution cases may call for different parameters. For example, an eigenvalue problem may use a shift parameter to eliminate rigid body modes, but a statics analysis cannot shift. Linear solver parameters for an individual solution case are set using the `solver_options` keyword in the solution case, and adding the corresponding `solver_options` section.

The **solver_options** sections allow multiple definitions of the parameter. Each "solver_options" section may be called out from a separate case in a multicase solution block as illustrated in Input input 4.9. A "solver_options" section applies *only* to the GDSW solver.

Note that cases using a `solver_options` section will ignore any "global" options defined in the `GDSW` section, and will instead use the internal GDSW defaults for any options that are not explicitly defined.

```
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
```

Input 4.9. Multiple Solver Options Example

4.7.2. Diagnostics

There is a variety of solver diagnostic information output to the files `dd_solver.dat` and `krylov_solver.dat`. Here we describe six different measurements of solution accuracy.

The `dd_solver.dat` file is used primarily to set up a simulation: adjusting the number of GPUs and configuring the preconditioner for later simulations with similar models. It is also an *ad hoc* **Heartbeat file**.

To override the default names, "dd_solver.dat" and "krylov_solver.dat", use the options **dd_solver_output_file** and **krylov_solver_output_file** in either the **GDSW** or `solver_options` input deck sections.

actual final residual Let's say we want to solve the linear system $Ax = b$ for the vector of unknowns x given the right-hand side vector b. If x_a is the approximate solution from the solver, then the actual final residual is $||b - Ax_a||$, where $||b||$ denotes the 2-norm of b. The actual final residual is reported in krylov_solver.dat after each solve.

actual relative residual This is $||b - Ax_a||/||b||$ and shows up in column 6 of dd_solver.dat. In other words, this is the actual final residual divided by the norm of right-hand side vector b.

recursive final residual During GMRES or CG iterations, we can recursively calculate $b - Ax_a$ without having to directly calculate Ax_a (for efficiency reasons). In exact arithmetic the actual and recursive finals residuals are identical, but in practice they can be different because of round off errors. If the effects of round off are not too big, then the actual and recursive final residuals should be close. The recursive final residual is reported after each solve in krylov_solver.dat.

recursive relative residual This is the recursive final residual divided by $||b||$ and shows up in column 5 of dd_solver.dat.

constraint error residual The constraint equations for a problem can be expressed as $Cx = 0$, where C is the constraint matrix. The constraint error residual is a normalized measure of Cx_a and should be small relative to 1 if the approximate solution x_a satisfies the constraints well. This residual is reported after each solve in krylov_solver.dat.

equilibrium error For problems with constraint equations, we solve the linear system $\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{Bmatrix} x \\ \lambda \end{Bmatrix} = \begin{Bmatrix} b \\ 0 \end{Bmatrix}$, where x is the vector of unknowns, λ is the vector of Lagrange multipliers, and C^T denotes the transpose of the constraint matrix C. For the approximate solution vector x_a, the equilibrium error is $||b - Ax_a - C^T\lambda||$ and is reported after each solve in krylov_solver.dat. In the absence of round off errors, the equilibrium error and the actual final residual should be identical.

All these residual measures may be more than is usually of interest, but they can provide valuable information for cases when solver convergence is an issue.

Picking a suitable solver tolerance for GDSW or any other iterative solvers requires close attention. If the solver tolerance is too high, then simulation results may not have sufficient accuracy. Likewise, if the solver tolerance is too low, then more analysis time may be spent obtaining a needlessly over-accurate solution. Solving a problem with two or more values for the solver tolerance can be useful to help avoid unnecessarily accurate solves and to also ensure that the solves are accurate enough. For example, let's say you do a modal analysis with a solver tolerance of 1e-4 and 1e-5. If you don't see a concerning change in the modal frequencies, then either choice for the solver tolerance is probably fine for the modal frequencies themselves. If, however, the modal solution is used as the basis for a subsequent analysis such as modal transient, then it is recommended that the effects of solver tolerance on the final results also be considered. Similar comments regarding the choice of a solver tolerance also hold for other solution cases such as direct transient analysis.

Transient simulations typically uses multiple time step sizes , with one preconditioner per time step size. A dd_solver.dat file is generated for the initial time step size. At ensuing times, the corresponding data is appended to the existing dd_solver.dat file.

We could provide users with estimates for how small a solver tolerance is needed to guarantee a certain measure of accuracy, but these estimates would usually be way too pessimistic to be of any practical value.

Additional details and troubleshooting strategies for the GDSW solver can be found in § 4.7.3 and documentation available on the `compsim.sandia.gov` website. Relevant documentation includes GDSW 101 and the GDSW Solver Tutorial. Solver strategies for dealing with poor mesh decompositions caused by the presence of constraints equations or multiple physics (i.e. structural-acoustics problems) are described 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. Input 4.10 provides recommended options for minimum memory use.

```
GDSW
  overlap=0
  max_iter=50   // set  to minimum required for a solution
  krylov_method = gmresClassic
  orthog=0
  precision_option_0=single
  precision_option_coarse=single
END
```

Input 4.10. Minimum Memory Recommended Options. These options, while not usually optimal for speed, may use the lowest memory.

Table 4-13. – GDSW Section Boolean Options may be set using integers 0 (no) or 1 (yes), or no and yes.

Variable	Values	Default	Description
prt_coarse	0/1	0	print coarse matrix
prt_constraint	0/1	0	print constraint matrix
prt_memory	0/1	0	print memory information
prt_timing	0/1	0	print timing information
prt_interior	0/1	0	print interior matrices
prt_overlap	0/1	0	print overlap matrices
write_orthog_data	0/1	0	write orthogonalization data to file

4.7.3. Troubleshooting GDSW

Ideally, the linear solver should always return a solution that satisfies the requested accuracy in terms of the relative residual tolerance (10^{-6} by default). This subsection provides some troubleshooting guidelines for situations when this is not the case. Additional information is available at `compsim.sandia.gov` in the Sierra/SD online documentation *GDSW Solver Tutorial* and *GDSW 101*. If problems persist, please submit a Sierra help ticket or reach out to a member of the Sierra/SD development team for assistance.

- **Default Solver Parameters:** In case of solver convergence problems, it is recommended that one first verify that the default solver parameters do not work. A notable exception is for problem types like acoustics or structural acoustics. In parallel, decreasing *solver_tol* table 4-10, say to around 10^{-10}, may enhance simulation fidelity, including structural and acoustic responses.
- **Negative Shift for Modal Analyses:** A common mistake is to not specify a negative shift for modal analyses of structures with no essential boundary conditions. If one or more rigid body modes are present, then the stiffness matrix will be singular and the solver will likely have problems converging except in special cases (see Singular Solves bullet below). The recommended shift is $-(2\pi f)^2$, where f is an estimate of the natural frequency (in Hz) of the first flexible mode. Of course the modes are not known in advance. The point is that the shift does not need to be an accurate estimate. For modeling an ordinary (larger than a paper clip, smaller than a house) steel component in Imperial units -10^6 almost always works. Caution: specifying a negative shift that is too large in magnitude may help the linear solver, but it can cause the algorithm that solves the eigenvalue problem to either require too many linear solves or not converge at all.
- **Memory Considerations:** The linear solver requires that enough memory be available to store the factorizations of subdomain and coarse space matrices. If the subdomain or coarse matrices are too large, then the memory capacity of the computing resource will be exceeded and a run will fail. There are some different ways to address such problems. First, one may request a smaller number of processors per compute node. For example, the CTS-1 machine *eclipse* has 36 cores per compute node, but one may request that only half or even a smaller number of cores be used per node. This has the effect of providing more memory

for each subdomain (MPI rank). In the same spirit, another option is to run the problem on a larger number of processors. This will result in smaller subdomain matrices, but the size of the coarse matrix will increase. For large problems, it may be necessary to use a second coarse level to limit the coarse matrix size. Memory resources might also be exceeded by the storage of search directions used by the Krylov method (e.g. GMRES). The default maximum number of iterations is 1000, and reducing this number will result in memory savings. Additional information on memory usage is available in the online documentation *GDSW Memory Use Tutorial*.

- **Convergence Issues:** Due to the limitations of finite precision arithmetic, it may not always be possible to provide a solution which satisfies the specified relative residual tolerance. This is especially the case for poorly conditioned linear systems or unrealistically small solver tolerances. Discussion of this topic continues in the online document *Solver Accuracy Notes*

  Increasing the maximum number of iterations is one simple option that can result in successful solves. Other options for improving convergence are described below in separate bullets. If convergence to the specified tolerance still cannot be achieved, then one option is to increase the relative residual tolerance until the solves are successful. To confirm that the quantities of interest have converged, one or more additional runs with even smaller tolerance are recommended.

- **Subdomain Overlap:** The default value of the `overlap` solver parameter is two. This means that the original subdomains are extended by two layers of elements when determining their overlap. Increasing the overlap often reduces the number of iterations needed for convergence, but the memory requirements increase and each iteration will require more time. Try a value for overlap of 3, 4, or more and see what effects it has. Even larger values of overlap can be used for models with only shell elements due to the reduced demands on the subdomain solver.

- **Static Condensation:** For non direct frequency response problems, the linear solver eliminates subdomain interior residuals at each iteration by default and solves the resulting Schur complement system of equations. For models with higher-order elements (i.e. polynomial degree greater than 2), it is recommended that the `SC_option` parameter be set to `none` since elimination of residuals interior to the subdomain can be problematic (i.e. effects of round off errors can be more pronounced). Similar concerns are present for direct frequency response problems, but the default option for such problems is `none`.

- **Frequency Response Analysis:** The iterative solution of direct frequency response problems can be challenging. The coefficient matrix $K + i\omega C - \omega^2 M$ can be both indefinite and complex depending on the input circular frequency ω and the damping matrix C. A more thorough discussion of trouble shooting the Helmholtz linear solver[44] is in the **Sierra/SD** How To manual in section Frequency response linear solver.

  A distributed memory sparse direct solver is recommended if the problem is not too large (to fit in memory). A direct solver works well for structures that are not blocky and shell element models. In the GDSW solver block set `useParallelDirectSolver = yes`. If available, this uses the Intel code Cluster Pardiso. Otherwise, SuperLU-Dist is used. If it is desired to use SuperLU-Dist on a platform where Cluster Pardiso is available, the option `useSuperLUDist = yes` must be specified.

If the distributed memory sparse direct solver option is not viable, then convergence may be improved as the number of processors increases. This leads to smaller subdomains and better performance of the coarse part of the GDSW preconditioner. Some users have also seen improved convergence by using the non-default solver parameter settings `orthogH = 0` and `precondUpdateFreq = 1`. If the problem already has a fair amount of damping, it may also help to set `structural_damping = 0`. This manual documents the interface 4.7.5.

- **Constraint Equations:** Certain types of constraint equations like those introduced by RBE3 elements can cause challenges for the linear solver (see earlier discussion of the `max_numterm_C1` solver parameter, Type 1 constraints, and Type 2 constraints). Models with Type 2 constraints are generally harder to solve than those without them. Two lines in the solver output file `dd_solver.dat` relevant to avoiding Type 2 constraints are `maxNumNonZeros in Tran Matrix` and `maxNumTermsForType1Constraints`. The current value of the `max_numterm_C1` solver parameter is reported by `maxNumTermsForType1Constraints`. Type 2 constraints can be avoided by setting `max_numterm_C1` to be no less than `maxNumNonZeros in Tran Matrix`, but in some cases this can lead to excessive memory requirements for the solver. This topic is described further elsewhere [45].

  Another source of potential difficulty is the presence of dependent or nearly dependent constraints. This means that the coefficient matrix $\hat{C}$ in the constraint equations $\hat{C}x = 0$ has either an infinite or large condition number. There are filters inside of Sierra/SD to eliminate linearly dependent constraints, but some may still be passed to the solver. The solver also identifies and eliminates dependent or nearly dependent constraints, but this is not always foolproof. It is advised that an analyst carefully check their model's contact definitions to avoid potential problems in terms of model correctness or for the linear solver. Additional information on constraint equations and their interactions with the linear solver is available in the online documentation *Constraints*.

- **Singular Solves:** The stiffness matrix for a structure with no essential boundary conditions is singular. Thus, static analyses or modal analyses with a zero shift require special considerations. First, any rigid body mode component of the applied loads must be removed (see `FilterRbmLoad` 5.28.2 and `num_rigid_mode`). The GDSW solver is able to solve singular systems, but it is important that the rigid body modes actually exist. For example, consider a model with curved surfaces that are connected using node-face tied surfaces. If a dependent node is not located on the independent surface, then one or more of the rotational rigid body modes will be lost. If the solver expects there to be six rigid body modes but there are say only three, then solver convergence problems are likely to occur. To avoid this problem, ensure that your model does indeed have the specified rigid body modes.

- **Very Small Solver Tolerances:** As noted earlier, it may not be possible for the solver to provide a solution which satisfies the requested relative residual tolerance. Nevertheless, there are some situations where it may be possible to reduce the actual relative residual below what is possible using default solver parameters. Sierra/SD has a guardrail to error out if the actual relative residual is more than 100 times of that requested (see `linear_solver_bailout_factor` in the `Parameters` section) By increasing this parameter while reducing the solver tolerance table 4-10, it may be possible to reduce the actual relative residuals for challenging problems like structural acoustics.

- **Scalability:** If you add `prt_debug=1` or `prt_debug=yes` in the GDSW section, then the information needed to measure the load balance is written to a file. The file name is `subdomainData.dat`. A row of data in the subdomainData file has 8 columns of integers. There is one row per subdomain (or MPI rank or core). Typically, the integers in each column are defined as follows.

  1. Interior unknowns (i.e. number of unknowns for static condensation).
  2. Non-zeros in the factors of the interior matrix.
  3. Unknowns on subdomain boundary.
  4. Owned unknowns on subdomain boundary.
  5. Unknowns in subdomain.
  6. Owned unknowns in subdomain.
  7. Unknowns in overlapping subdomain.
  8. Non-zeros in the factors of the overlap matrix.

  In the special case of a diagonal preconditioner (`preconditioner_type=DIAG`), only 2 integers are printed.

  1. Unknowns in subdomain.
  2. Owned unknowns in subdomain.

  Each iteration sometimes accesses each matrix a constant number of times. Often times the non-zeros in the factors of the overlap matrix predict the load balance. Larger numbers are generally more important. If the interior unknowns are eliminated, then non-zeros in the factors of the interior matrix will sometimes predict the load balance. The unknowns in a subdomain is governs the load balance of matters related to `orthog`. As `orthog` or `orthogH` increases, the influence of the unknowns per subdomain increases.

4.7.4. Mathematical Conditioning Issues

The performance of the solver is closely tied to the mathematical conditioning of the equation system it is solving. Conditioning can be thought of as the ratio of the largest eigenvalue of the system to the smallest eigenvalue. The larger this ratio becomes the more difficult it will be to solve the system and the larger potential error there will be in the solution.

Many model features can increase (or worsen) model conditioning.

- Rigid body modes (which have zero eigenvalues) are a special case which is handled as carefully as is theoretically possible, but which nonetheless have robustness problems.
- Mesh refinement increases gradients.
- Poorly shaped elements can lead to ill conditioned linear systems
- Understanding Type 1 versus Type 2 constraints as discussed in the definition of the GDSW option *max_numterm_C1* can lead to substantial performance improvements.
- Models in which a stiff material abuts a soft material are poorly conditioned.
- Mixtures of high density and low density components.
- Meshes with mixtures of solid and structural elements.

A poorly conditioned system can cause difficulties for the solver, both from an accuracy and robustness standpoint.

- Long solve times or many iterations required for convergence
- Difficulty obtaining a low target residual during the solve
- Large difference between the 'recursive' and 'actual' residuals reported by the solver. This can also indicate the solver result has lower accuracy than intended.
- Significant localized errors in the result obtained at some nodes

4.7.5. Frequency Response Functions

Several additional SD training documents [44, Frequency response linear solver] and presentations describe the solver behavior, debugging, and usage guidelines in more detail. See the **Sierra/SD** training documents on 'GDSW Solver Accuracy Notes', 'GDSW 101', 'GDSW Memory Use Tutorial.

4.7.6. Parameters

SubdomainData is described in Section 4.7.3.

In Table 4-16 *Hprecond*'s option *custom* preconditions by

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

As usual $i = \sqrt{-1}$, ω is the circular frequency of excitation, and M, C and K are the mass, damping, and stiffness matrices, respectively. Here β and γ are the viscous damping and structural damping coefficients. Experience with the custom preconditioner is documented elsewhere.[44] The custom preconditioner can reproduce the other preconditioners. 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 $\alpha_K = 1$, $\beta_K = \gamma + \beta\omega$, $\alpha_M = 1$

Table 4-14. – GDSW Section Options (Advanced).

Variable	**Values**	**Default**	**Description**
version	*integer*	2	GDSW version
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 and precision_option_coarse = single
cull_method	*integer*	1	0 - none: reach maximum and then stop 1 - simple: reach maximum and then remove most recent ones 2 - eigen: cull search directions based on solution to eigenvalue problem
preconditioner_type	*integer*	2	1 (BDDC), 2 (GDSW) 3 (DIAG), 4 (NODAL)
interface_precond	*integer*	0	0 - do not require interface preconditioner 1 - require interface preconditioner
coarse_connectivity_option	*integer*	1	algorithm number for coarse elem connect
viscous_damping	*real*	0	viscous damping coefficient for preconditioner operator
structural_damping	*real*	0.12	structural damping coefficient for preconditioner operator
rbm_tolerance	*real*	1e-12	threshold for the rigid body mode residual norms reported in dd_solver.dat. Recommend usage of the RbmTolerance in the parameters section.
con_tolerance	*real*	2.5e-9	singularity tolerance for processing constraints
con_row_tolerance	*real*	1e-1	pivoting tolerance for processing constrains
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
use_epetra_coarse	*integer*	0	0 - do not use Epetra matrices for coarse correction 1 - use Epetra matrices for coarse correction
parmetis_option	*integer*	0	Parmetis option for coarse problem partitioning: 0 (PartKway), 1 (PartGeomKway)
diag_scaling	*string*	none	none - no scaling of operator matrix diagonal - symmetric diagonal scaling (diagonal-based) column - symmetric diagonal scaling (column-based)
correction_option	*integer*	0	0 (standard), 1 (MINRES)

Table 4-15. – GDSW Section Print Options.

Variable	**Values**	**Default**	**Description**
ML_print_coarse	*integer*	0	0 - no output 1 - print coarse stiffness matrix
ML_print_Phi	*integer*	0	0 - no output 1 - print interpolation matrix
pardiso_message_level	*integer*	0	0 - no messages 1 - print messages
prt_matrix	*integer*	0	0 - no output 1 - print out matrix in 3-column format 2 - print out matrix in CSR format
prt_subdomain_coarse	*integer*	0	0 - do not print subdomain coarse matrix 1 - print subdomain coarse matrix
prt_subdomain_PU	*integer*	0	0 - do not print sub partition of unity 1 - print sub partition of unity
prt_stabilization	*integer*	0	0 - do not print coarse stabilization matrices 1 - print coarse stabilization matrices
prt_interior	*integer*	0	0 - do not print interior matrices 1 - print interior matrices
prt_memory	*integer*	0	0 - do not print gdsw memory diagnostics 1 - print gdsw memory diagnostics
prt_debug	*integer*	0	0 - do not print subdomainData.dat 1 - print subdomainData.dat
write_orthog_data	*integer*	0	0 - do not write orthogonalization data to file 1 - write orthogonalization data to file 1 - ignore memory reporting
ML_print_coarse	*integer*	0	0 - do not print coarse stiffness matrix 1 - print coarse stiffness matrix
ML_print_Phi	*integer*	0	0 - do not print interpolation matrix 1 - print interpolation matrix

Table 4-16. – GDSW Section Options (Helmholtz).

Variable	Values	Default	Description
Hprecond	*integer*	5	Helmholtz preconditioner: 0-stiffness: Stiffness based 1-LG: Laird-Giles 2-custom: Custom 3-SL: shifted Laplacian 5-operator: Operator with damping
orthogH	*integer*	20	maximum number of stored search directions for Helmholtz problems
max_previous_sols	*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

for *operator*. Using the *stiffness* preconditioning option for ω near zero and for structures with rigid body modes is discouraged due to the near singularity of K.

Table 4-17. – GDSW Section Options (Solvers).

Variable	Values	Default	Description
default_solver	*integer*	1	1-direct: Esmond Ng's sparse direct solver 2-LDM: Clark's LDM' sparse direct solver 6-NoPivot: Clark's sparse direct solver
PTAP_solver	*integer*	1	solver for conjugate gradient matrix 0-diag: diagonal (holds in exact arithmetic) 1-full: full $\Phi^T A \Phi$ matrix

Table 4-18. – GDSW Section Options (tacho/gpu).

Variable	Values	Default	Description
tacho_alg_variant	*enum OR int*	default(2)	default(2)|new(3)
tacho_force_solution_method	*enum*	symlu	symlu|ldl|no_pivot_ldl

4.8. Sensitivity

Sensitivity to parameters is available for modal analysis,[45] Craig-Bampton reduction (CBR), static solutions and some transient solutions. An example input deck for modal analysis is given in the Section 11. In the case of CBR analysis, we refer to sections 5.3 and 5.3.1 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 deck. The following example illustrates the legal keywords. Valid keywords are identified in Table 4-19. Lists of numbers should follow the rules for integer lists detailed in Section 4.1.

```
SENSITIVITY
  values all
  vectors 1,3,5,7:9
  iterations 8
  tolerance 1e-7
  Attune
  AttuneNodeset sensitivity_nodeset
END
```

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 analysis, `vectors` refers

Keyword	argument	Description
values	"all"/"none" or list	eigenvalue selection
vectors	"all"/"none" or list	eigenvector selection
iterations	integer	number of eigenvector iterations
tolerance	float	convergence tolerance for eigenvectors
attune	n/a	enable attune output
AttuneNodeset	string	nodeset for reduced model

Table 4-19. – Sensitivity Analysis Keywords.

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 1:3,5               // range of cases/modes
```

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. If the eigenvector sensitivity approximation fails to converge to the desired tolerance in the specified number of iterations, a fatal error occurs.

4.8.1. 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 named after the input deck. A surrogate of the finite element model is determined using eigenvectors. Attune applies *only* to eigen sensitivity analysis, and the eigenvalues must be selected using `values`. 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.

To use `Attune` , please refer to the ATA website and on line documentation.[5]

Output. Sensitivity results are output to the same file as the nominal results. The arrangement of the output depends on the analysis. The statics nominal result is output, followed by the sensitivity result for each parameter. In eigenvalue problems, the nominal frequencies and eigenvectors are output, followed by the eigenvalue and eigenvector sensitivities with respect to the first parameter, the second parameter, etc. The eigenvalue sensitivities are placed in the time field of each output record, like 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.

Here is an example of how to request both eigenvalue and eigenvector sensitivities. The sensitivities are extracted using the Global variables, including sensitivities to `area` in Block 1000 and `thickness` in Block 101. The order of parameters can be determined in several ways. This examples uses the Global variables checked using the command line `explore` application in the Sierra SEACAS suite. The order of parameters can also be determined from information in the results file.

```
workstation_prompt> explore my_output.exo
EXPLORE select step 4
EXPLORE gvar
```

In the input deck, the parameter `nmodes` sets the number of requested mode shapes. In this case, `nmodes` has been set to 6. The next `nmodes` values are the $d\lambda/dP_1$ values associated with the first parameter, P_1. The corresponding vectors are $d\phi_i/dP_1$.

```
Global Time Step Variables
ModeNumber                      4.0000E+00
EigenFrequency                  3.5042E+03
EigenVectScale                  1.0000E+00
deriv_Block_1000_area      3.9362E+03
deriv_Block_101_thickness  1.4426E+07
```

```
EXPLORE times
EXPLORE exit
```

```
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 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 26
  area 0.10 +/- 5 percent      // relative tolerance specified
END

BLOCK 28
  thickness  +/- 1 percent     // relative to Exodus attribute
END
```

Note that the tolerances are specified on the parameters where they normally appear in the input deck. That is, these definitions do not appear in the **sensitivity** section.

The sensitivity quantities output to the exodus file are derivatives, and can be used to compute a first-order approximation to the change in an output quantity with respect to a parameter. For example, the change in an eigenvalue λ that depends on a parameter p can be approximated

$$\lambda(p + \Delta p) = \lambda(p) + \frac{d\lambda}{dp}\Delta p + o(\Delta p).$$

The quantity written to the exodus output is $d\lambda/dp$. The tables printed in the results and standard output also include the linear approximation to the change in λ,

$$\frac{d\lambda}{dp}\Delta p.$$

This approximation may be inaccurate if Δp is too large. The size of Δp is specified by the user in the input deck for each parameter included in the sensitivity study.

4.8.1.1. Derived Output Quantities

The sensitivity analysis methods use a semi-analytic method. Primary variables (usually displacement) are computed in terms of changes in stiffness and mass matrices, and resultant displacements are then computed analytically. See Figure 4-3. Many other output quantities are computed in terms of these primary variables using the standard output routines. Most of these output quantities depend linearly on the primary variables. For example, computation of the derivatives of strain can be readily computed using the chain rule, and may also employ the same procedures for strain computation. Let $\varepsilon = \kappa u$ define the strain/displacement relationship at a given point in the model. Here κ is a constant.

$$\begin{aligned} \frac{d\varepsilon}{dp} &= \frac{d\varepsilon}{du}\frac{du}{dp} \\ &= \kappa \frac{du}{dp} \end{aligned}$$

Code that computes ε from u may be used to compute $\frac{d\varepsilon}{dp}$ from $\frac{du}{dp}$.

Other variables are not linear with respect to the primary variables. For example, the strain energy or the von Mises stress include quadratic terms in displacement.

$$\begin{aligned} E_s &= u^T K u \\ \frac{dE_s}{dp} &= \frac{dE_s}{du}\frac{du}{dp} \\ &= u^T K \frac{du}{dp} \end{aligned}$$

These variables may not use the same code paths. Data is written, but is not correct for these variables.

Statics

$$\begin{aligned} Ku &= f \\ \Delta u &= K^{-1}(\Delta f - \Delta K u) \end{aligned}$$

Eigen

$$\begin{aligned} (K - \lambda M)\phi &= 0 \\ \Delta \lambda_i &= \phi^T \Delta K \phi - \lambda \phi^T \Delta M \phi \end{aligned}$$

Figure 4-3. – Semi-Analytic Methods for Sensitivity Analysis.

Solution Types Sensitivity analysis is available only for the solution types shown in Table 4-20. The primary application is in eigenvalue problems where the semi-analytic solutions can provide significant computation and accuracy benefit over a finite difference approach.[4]

Sensitivity Limitations

1. Subsection Eigen Sensitivity Analysis section Solution Procedures of the Theory Manual explains how sensitivity analysis may be performed using most solvers.

2. Outputs are limited to variables linear in displacement (4.8.1.1). In particular, output of von Mises stress is output for these solutions, but is not correct. That is the derivative of the von Mises stress is not the von Mises stress of the derivative of the stress.

Table 4-20. – Sensitivity Analysis Solution Type Availability.

Name	**Section**	Description
eigen	5.8	Normal Modes
statics	5.24	Linear Statics

4.9. Coordinates

Local coordinate systems may be defined to orient directional materials (section 6.6.1.3), define constraints section 4.11, boundary conditions section 8.1, or loads section 4.5 in local orientations or transform outputs to a local coordinate system (sections 9.2 and 9.3). The basic/default coordinate frame can also be explicitly referenced as coordinate “0”, if desired.

The ability to visualize and verify correctness of coordinate system definitions is a key element of the overall workflow.

At present, visualization support is limited to the material direction output used to orient directional materials (sections 6.6.1.3 and 9.8.11).

```
BEGIN RECTANGULAR COORDINATE SYSTEM <string>name
  ORIGIN = <real> <real> <real>
  Z POINT = <real> <real> <real>  # Point on Z̃ axis
  XZ POINT = <real> <real> <real> # Point on X̃Z̃ plane
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END

BEGIN RECTANGULAR COORDINATE SYSTEM <string>name
  ORIGIN NODESET   = <nodelist>    # Single-node nodesets
  Z POINT NODESET  = <nodelist>    # Node on Z̃ axis
  XZ POINT NODESET = <nodelist>    # Node on X̃Z̃ plane
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END
```

Syntax 4.3. Rectangular Coordinate System Syntax

```
BEGIN CYLINDRICAL COORDINATE SYSTEM <string>name
  ORIGIN = <real> <real> <real>
  Z POINT = <real> <real> <real>  # Point on $\tilde{Z}$ axis
  XZ POINT = <real> <real> <real> # Optional: point on $\tilde{X}\tilde{Z}$ plane; sets
                                  # location of azimuthal angle theta=0
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END

BEGIN CYLINDRICAL COORDINATE SYSTEM <string>name
  ORIGIN NODESET  = <nodelist>    # Single-node nodesets
  Z POINT NODESET  = <nodelist>   # Node on $\tilde{Z}$ axis
  XZ POINT NODESET = <nodelist>   # Optional: node on $\tilde{X}\tilde{Z}$ plane; sets
                                  # location of azimuthal angle theta=0
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END
```

Syntax 4.4. Cylindrical Coordinate System Syntax

```
BEGIN SPHERICAL COORDINATE SYSTEM <string>name
  ORIGIN = <real> <real> <real>
  Z POINT = <real> <real> <real>  # Point on $\tilde{Z}$ axis
  XZ POINT = <real> <real> <real> # Optional: point on $\tilde{X}\tilde{Z}$ plane; sets
                                  # location of azimuthal angle theta=0
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END

BEGIN SPHERICAL COORDINATE SYSTEM <string>name
  ORIGIN NODESET  = <nodelist>    # Single-node nodesets
  Z POINT NODESET  = <nodelist>   # Node on $\tilde{Z}$ axis
  XZ POINT NODESET = <nodelist>   # Optional: node on $\tilde{X}\tilde{Z}$ plane; sets
                                  # location of azimuthal angle theta=0
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
END
```

Syntax 4.5. Spherical Coordinate System Syntax

```
BEGIN CONICAL COORDINATE SYSTEM <string>name
  ORIGIN = <real> <real> <real>
  Z POINT = <real> <real> <real>  # Point on $\tilde{Z}$ axis
  XZ POINT = <real> <real> <real> # Optional: point on $\tilde{X}\tilde{Z}$ plane
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
  ANGLE = <real>
END
```

```
BEGIN CONICAL COORDINATE SYSTEM <string>name
  ORIGIN NODESET  = <nodelist>    # Single-node nodesets
  Z POINT NODESET  = <nodelist>   # Node on Z̃ axis
  XZ POINT NODESET = <nodelist>   # Optional: Node on X̃Z̃ plane
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
  ANGLE = <real>
END
```

Syntax 4.6. Conical Coordinate System Syntax

```
BEGIN ELLIPSOIDAL COORDINATE SYSTEM <string>name
  ORIGIN = <real> <real> <real>
  Z POINT = <real> <real> <real>  # Point on Z̃ axis
  XZ POINT = <real> <real> <real> # Point on X̃Z̃ plane
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
  AXIS STRETCHING = <real> <real> <real> #angle in radians
END

BEGIN ELLIPSOIDAL COORDINATE SYSTEM <string>name
  ORIGIN NODESET  = <nodelist>    # Single node nodeset
  Z POINT NODESET  = <nodelist>
  XZ POINT NODESET = <nodelist>
  ROTATE <real> ABOUT AXIS X|Y|Z  # angle in radians
  AXIS STRETCHING = <real> <real> <real> #angle in radians
END
```

Syntax 4.7. Ellipsoidal Coordinate System Syntax

The prefix to `COORDINATE SYSTEM` specifies the type of local coordinate system. Supported types are: `RECTANGULAR` (Cartesian), `CYLINDRICAL` (Polar), `SPHERICAL`, `CONICAL` (a cylindrical system with an aperture angle), and `ELLIPSOIDAL` (a spherical system stretched by `AXIS STRETCHING`).

A local element coordinate system is defined by a set of three control points. These points may be defined using absolute locations (`ORIGIN`, `Z POINT`, `XZ POINT`) or using nodelists in the mesh file (`ORIGIN NODESET`, `Z POINT NODESET`, `XZ POINT NODESET`) that *contain exactly one node each.* The control points define a local $\tilde{X}\,\tilde{Y}\,\tilde{Z}$ Cartesian coordinate system centered at the coordinate system origin. For cylindrical, spherical, conical, and ellipsoidal systems the orientation of the basis vectors at individual nodes and elements is spatially dependent. In this documentation, the local basis vectors at each node or element will be denoted as $\hat{r}$, $\hat{s}$, and $\hat{t}$.

Note: coordinate systems defined by nodesets use only the initial location of the nodes and do not update with model displacement.

- `ORIGIN`: Required. Specifies the origin location for the new coordinate system.
- `Z POINT`: Required. Specifies a point on the $\tilde{Z}$ axis for the new coordinate system. For `SPHERICAL`, this defines where the zenith (polar) angle ϕ is 0.
- `XZ POINT`: Required only for `RECTANGULAR` and `ELLIPSOIDAL`. Third point to specify the $\tilde{X}\tilde{Z}$-plane of the new system. Must not lie on the $\tilde{Z}$ axis. If not specified, an arbitrary axis will be chosen based on the global coordinate system. For `CYLINDRICAL` and `SPHERICAL`, this defines where the azimuthal angle θ is 0. `XZ POINT` also affects behavior at ill-defined points such as the origin of `SPHERICAL` systems, or along the $\tilde{Z}$ axis of `CYLINDRICAL` systems (see figure 4-8).
- `ANGLE` (`CONICAL COORDINATE SYSTEM` only): Required. Defines the angle in radians to rotate the axial direction away from the $\tilde{Z}$ axis (see figure 4-6).
- `AXIS STRETCHING` (`ELLIPSOIDAL COORDINATE SYSTEM` only): Required. Defines the aspect ratio to stretch a spherical system (see figure 4-7).
- `ROTATE`: Processed last. Specifies a rotation (in radians) of the coordinate system about one of its own axes. The conical system is effectively a cylindrical system with a local rotation about the $\tilde{Y}$ axis. Only one rotation may be specified.

The three coordinate control points are illustrated in figure 4-4. The coordinate system types are depicted in figures 4-5 to 4-7.

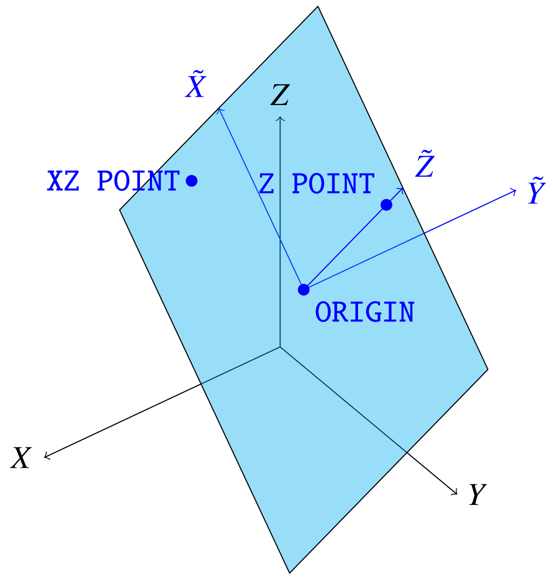


Figure 4-4. – Coordinate system definition vectors: note that `XZ POINT` determines the $\tilde{X}$ axis, but need not lie along it, which is the case depicted in this figure.

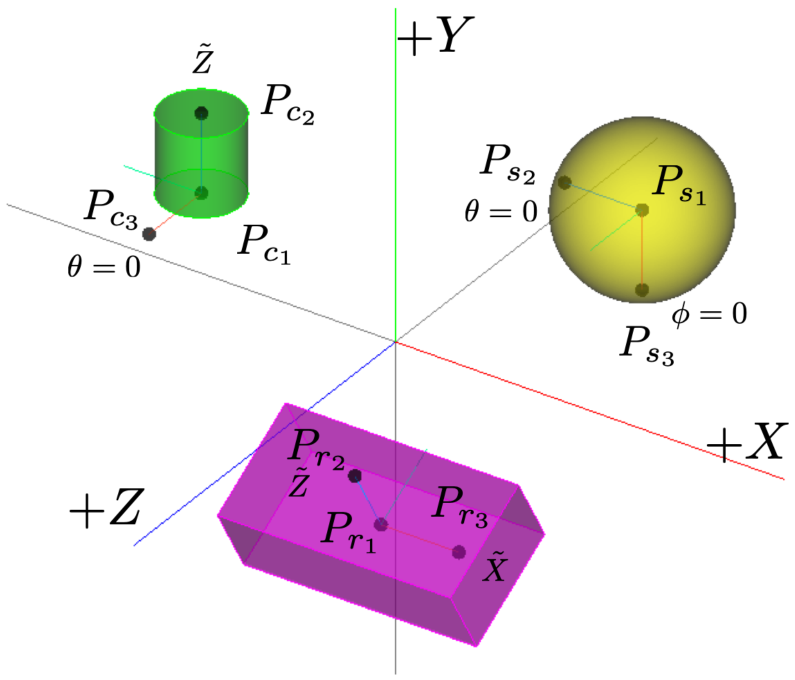


Figure 4-5. – Three examples (rectangular, spherical, and cylindrical) of transformed coordinate systems are given. The center of the rectangular block $(6 \times 2 \times 3)$ is located at $(3, -1, 5)$, and rotated 30 degrees about the X axis. The center of the sphere (radius of 2) is located at $(5, 4, -2)$. The center of the cylinder (radius of 1, height of 2, and rotated 90 degrees about the X axis) is located at $(-5, 2, 0)$. The new coordinate systems are defined respectively of each of the geometries, and are indicated by $\tilde{X}\,\tilde{Y}\,\tilde{Z}$ and the subscripts (r, s, and c).

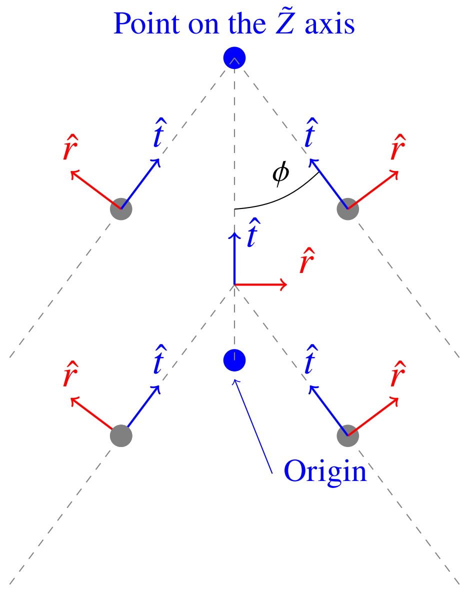


Figure 4-6. – Conical Coordinate System Definition at $\tilde{X}\tilde{Z}$ plane

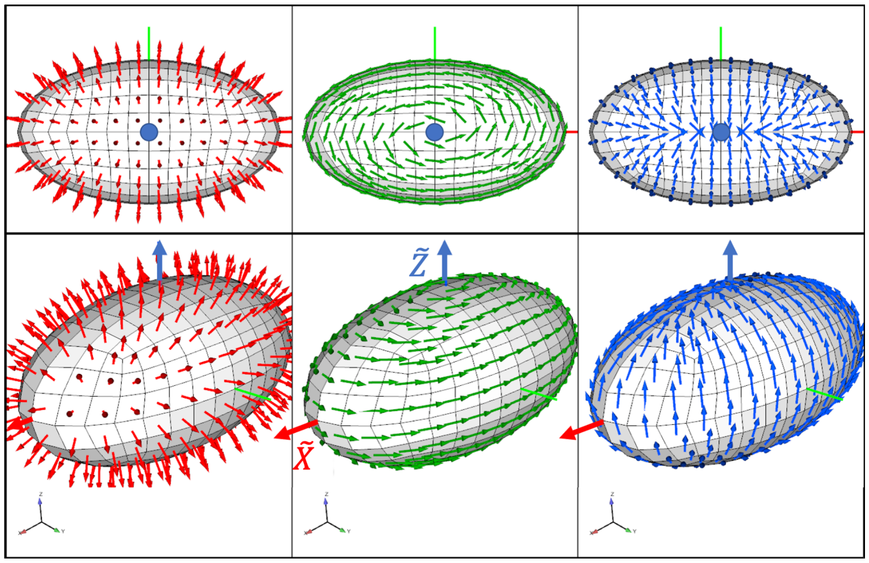


Figure 4-7. – Ellipsoidal Coordinate System Using axis_stretching = (2.0, 1.1, 1.0). $\hat{r}$ is red, $\hat{s}$ is green, and $\hat{t}$ is blue.

Examples of rectangular, spherical, and cylindrical coordinate systems are given in figure 4-5. In those examples, we wish to define coordinate systems for three geometries: a brick, a sphere, and cylinder. The corresponding input deck is shown below.

```
begin rectangular coordinate system rectangular_system
  origin   3 -1 5
  z point  5 -1 5
  xz point 3 0.7321 6
end
coordinate spherical coordinate system ball_like
  origin 5 4 -2
  z point 5 2 -2
  xz point 3 4 -2
end
begin cylindrical coordinate pin_system
  origin -5 2 0
  z point -5 4 0
  xz point -5 2 2
end
```

Input 4.11. Example coordinate system input

For cylindrical, conical, spherical, and ellipsoidal systems, the local basis vectors $\hat{r}$, $\hat{s}$, and $\hat{t}$ are not all well-defined when sampled along the $\tilde{Z}$ axis. At the origin, all coordinate systems fall back

Coordinate System	History Variable	Corresponding Coordinate
Cylindrical	X	r
	Y	θ
	Z	z
Spherical	X	r
	Y	θ
	Z	ϕ

Table 4-21. – Coordinate Names for history files.

to a local Cartesian system with $\hat{r}$ along the $\tilde{X}$ axis, $\hat{s}$ along the $\tilde{Y}$ axis, and $\hat{t}$ along the $\tilde{Z}$ axis. This is also true for the cylindrical and conical systems on the $\tilde{Z}$ axis. Spherical and ellipsoidal systems on the $\tilde{Z}$ axis will instead maintain the first vector ($\hat{r}$) pointing radially outward, the third vector ($\hat{t}$) pointing along the $\tilde{X}$ axis, and the second vector ($\hat{s}$) as either the positive or negative $\tilde{Y}$ axis (generating a right handed system). This maintains the normal and tangential properties of the three vectors for spherical coordinates. See figure 4-8 for a visual representation of this behavior.

The **Exodus** mesh input file also has the ability to define local coordinate frames. These frames will be read and available during an analysis. In the case of a coordinate frame being defined in both the **Exodus** file and the input file, the input file definition will be used, and a warning will be logged.

In spherical coordinates, it may help to consider the Cartesian frame $(\tilde{X}, \tilde{Y}, \tilde{Z})$ with the same orientation as (r, θ, ϕ):

$$\begin{aligned}\tilde{X} &= r\sin(\phi)\cos(\theta) \\ \tilde{Y} &= r\sin(\phi)\sin(\theta) \\ \tilde{Z} &= r\cos(\phi),\end{aligned} \tag{4.9.1}$$

$$0 \leq \phi < \pi, \quad 0 \leq \theta \leq 2\pi.$$

If the user specifies a coordinate system in the `History` section, notice that its applicability may be somewhat limited, though convenient. In particular, only a single history file is written in each analysis, and only one coordinate frame may be outputted per node (see section 9.2). The history file will display variables as Cartesian regardless of coordinate choice. Table 4-21 shows the corresponding values for cylindrical and spherical coordinates. Finally, we provide a full example of how we can use the coordinate definitions. We use the same coordinate definitions from figure 4-5 and input 4.11, and we define the brick as block 1, the sphere as block 2, and the cylinder as block 3, respectively. We also specify a sideset 100 on the brick, a nodeset 200 on the sphere, and a nodeset 300 on the cylinder as shown in figure 4-9. We wish to apply a radial force emanating from the middle of the cylinder, and we want to record the output accelerations on the

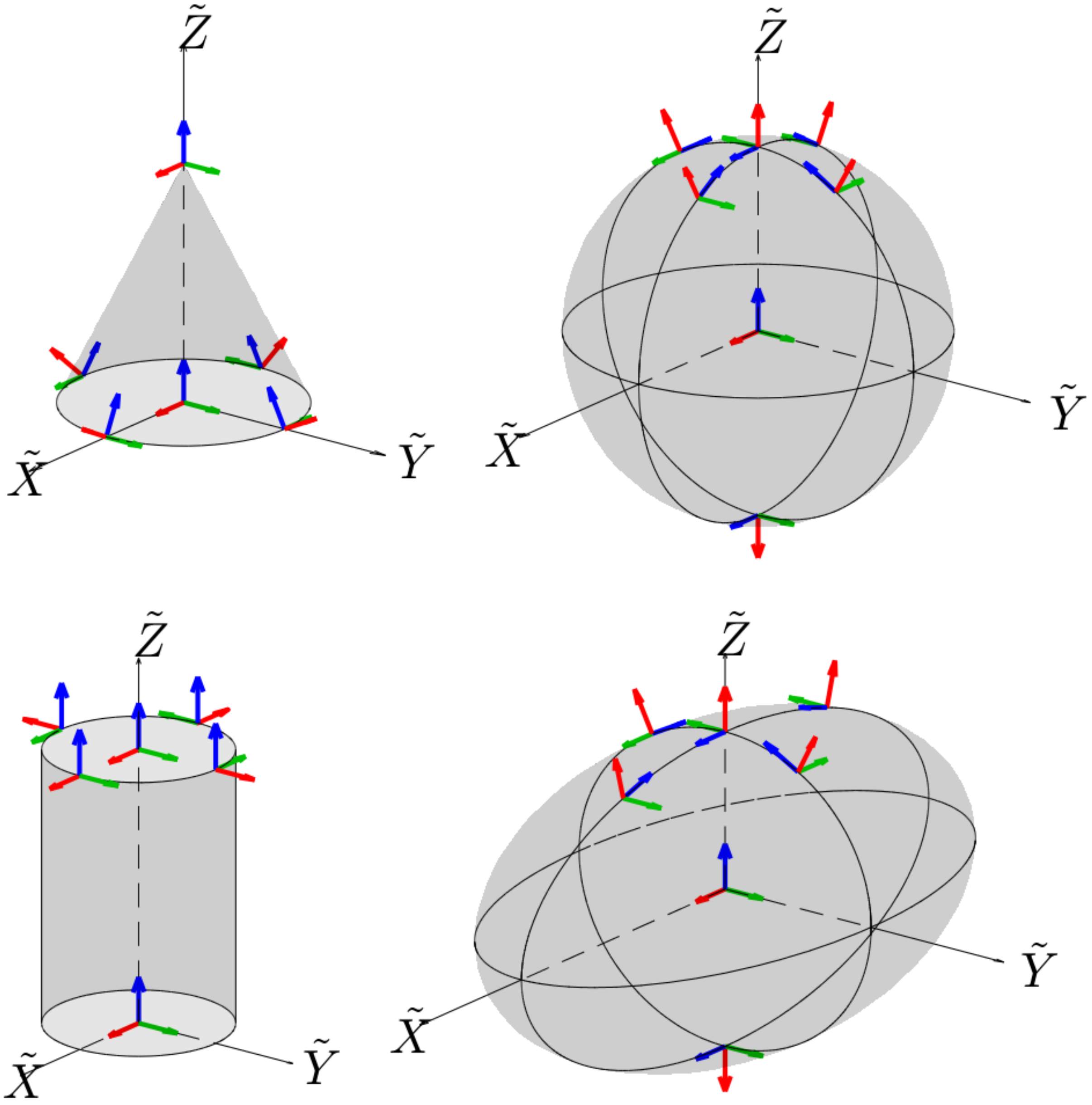


Figure 4-8. – Coordinate system behavior near the $\tilde{Z}$ axis. $\hat{r}$ is red, $\hat{s}$ is green, and $\hat{t}$ is blue.

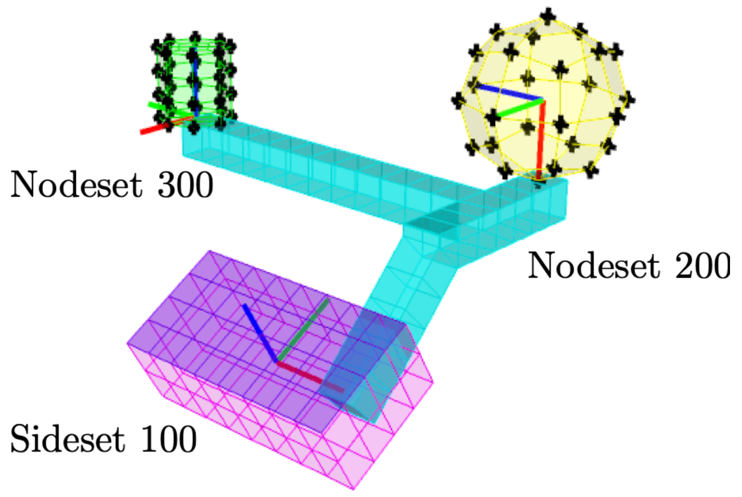


Figure 4-9. – An example of how coordinate systems are used. Accelerations are measured on the sphere (nodeset 200) due to a radial force applied on the cylinder (nodeset 300).

sphere. The corresponding input deck is attached, with associated coordinate systems defined in input 4.11.

```
LOADS
    nodeset 300
    coordinate ball_like
    force = 1 0 0
END

HISTORY
    nodeset 200
    coordinate pin_system
    accel
END
```

Tractions can also be specified on a particular coordinate frame, but special care is required. We refer the reader to section 8.3.3 for using arbitrary coordinates with tractions.

4.10. Function

Time, frequency and/or spatially dependent functions for transient and frequency response analyses are defined using the `function` section. The following are simple examples of the use of a time dependent `function`.

```
FUNCTION test_func1
  type LINEAR
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION poly_fun
// a pulse of duration .05
// peak value 6/7 at 1/49 sec
// pulse(t)=8*sqrt(t)-686*t^2
  name "Smooth Pulse with Duration 0.05"
  type POLYNOMIAL
  data 0. 0.
  data .5 8.
  data 2. -686.
END
```

The function name follows the function keyword. This identifier is used to reference the function in other parts of the input deck. It can be any string. In early versions of **Sierra/SD** only integer function names were supported. Function definitions have the keywords TYPE, NAME and DATA.

1. TYPE to define the functional form,
2. NAME Additional reference information for the function. This may be printed to error messages or other informational output regarding the function, however it is the identifier that is always used to call the function from other parts of the input deck.
3. DATA for the functional parameters.

Other function definitions may require more parameters.

4.10.1. Function Offset/Shifts

Function input/output values can be offset and scaled. The syntax for defining these transformations is below.

```
X|abscissa offset = <real, default=0>
Y|ordinate offset = <real, default=0>
X|abscissa scale = <real, default=1>
Y|ordinate scale = <real, default=1>
```

Given a function $y = f(x)$, the transformed output would be:
$y = y_{scale} \cdot \left[f(x_{scale} \cdot [x + x_{offset}]) + y_{offset} \right]$.

Note that function offsets/shifts are currently only enabled for single-input, single-output functions. Otherwise, a warning will be issued and the offsets/shifts will be ignored.

4.10.2. Linear Functions

For linear functions, the data elements are pairs specifying the independent variable (e.g., time) and the corresponding function value. Evaluation of the function at unspecified points is performed via linear interpolation. In order to enforce uniqueness of the interpolant, **Sierra/SD** ignores points with an independent variable value less than that of the previous point. For example, the last point in the `ignored_point` function is ignored.

```
FUNCTION ignored_point
  type linear
  data 0.00 0.
  data 0.01 1.
  data 0.05 1.
  data 0.04 0. // ignored: column 1 may not decrease
END
```

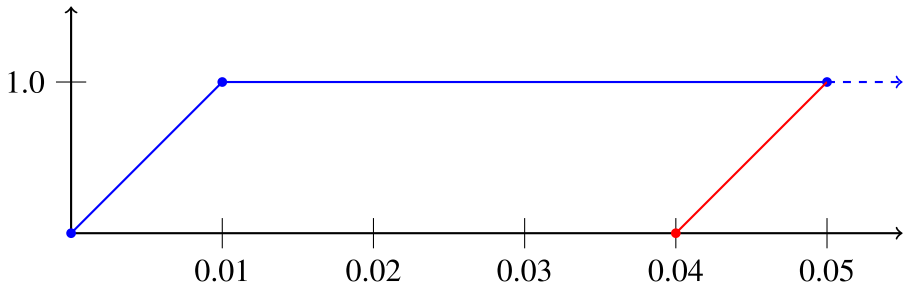


Figure 4-10. – Linear function "ignored_point".

Figure 4-10 shows the graph of function ignored point.

Extrapolation refers to evaluating the function before the first given time or after the last given time. Linear functions use the value of the nearest data point to extrapolate. For example, the value of the following function at 0.03 is 0.5.

```
FUNCTION extrapolation
  type linear
  data 0.00 0.
  data 0.01 1.
  data 0.02 0.5
END
```

Figure 4-11 shows the graph of the *extrapolation* function.

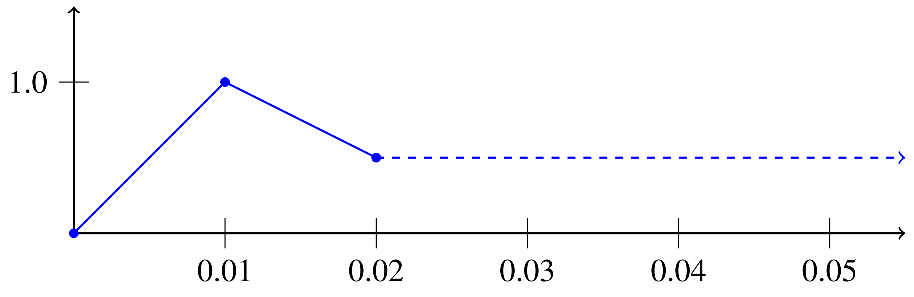


Figure 4-11. – Linear function "extrapolation".

If an independent variable value is listed multiple times, then the function value at that point is the average of the specified function values. Function 5 is pictured in Figure 4-12.

```
FUNCTION 5
  name "value3hundrethslistedtwice"
  type linear
  data 0.00 0.
  data 0.01 1.
  data 0.03 1.
  data 0.03 0.5   // f(3/100)=3/4
END
```

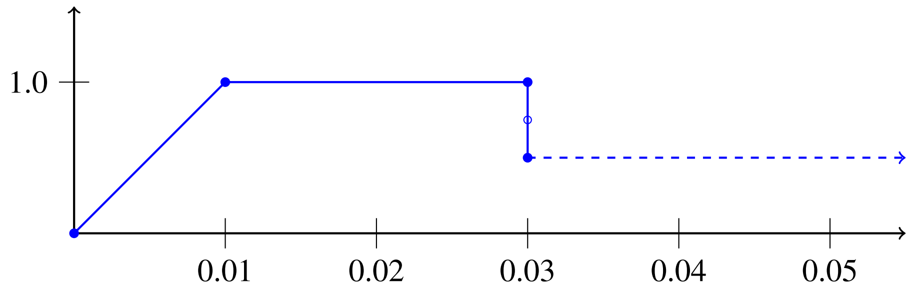


Figure 4-12. – Linear function #5. "multiple_fun".

4.10.3. Sierra SM Piecewise Linear Functions

Some limited Sierra **Sierra/SM** syntax is supported for specifying a piecewise linear function in **Sierra/SD**. The data entered through this syntax is interpreted as a linear function, and follows all the rules specified above. This comes with limitations. Only the `piecewise linear` function type is supported, and some features are missing. Most importantly, ending the function with "END FUNCTION <string>" is not supported. This comes from an incompatibility with existing **Sierra/SD** syntax; "FUNCTION <string>" happens to be valid **Sierra/SD** syntax in this scope and context.

```
BEGIN FUNCTION step
  Type is Piecewise Linear
  BEGIN VALUES
    0.00, 0.
    0.01, 1.
    0.05, 1.
  END VALUES
END
```

4.10.4. *Functions using Tables*

Functions may be specified by reference to a linearly interpolated `table` (as discussed in section 4.10.19). The table must be of dimension=1. One-dimensional tables behave identically to the linear functions described above. However, they will typically be much faster and more memory efficient than linear functions, especially as the data size grows.

The function in the following example is a tabular representation of the data of Figure 4-11 and Function “extrapolation” 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
```

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.

4.10.5. Polynomials

The data pairs for polynomials are exponent and coefficient pairs. The independent variable taken to any real power will always be evaluated as positive. Duplicate exponents are handled by summing their coefficients.

```
FUNCTION 6
  name "quadratic_polynomial"
  type polynomial
  data 0.0 0.
  data 1.0 1.
  data 2.0 0.1
  data 1.0 0.5  //  f(t) = 3t/2 + (t^2)/10
END
```

4.10.6. LogLog Functions

Loads may be applied with log log functions in frequency domain analyses. For example log log tables are used for random vibration inputs. Option `LogLog` applies linear interpolation on an log log tables plot so that only the corner frequencies need be specified. An example follows.

```
FUNCTION 168
  name "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
```

4.10.7. SamplingRandom

A Random Pressure load has user specified spatial and temporal correlations. Random Pressure loads are described in the Theory Manual in section Loads and Materials, subsection Random Pressure Loading. If the desired random function is a function of time only, then the `SamplingRandom` function described here provides a simpler mechanism for applying the load.

Note that

$$\omega_c = 2\pi \times \text{cutoff\_freq},$$

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	*string*	defaults to $\sin(x)/x$	function for time interpolation
scale_function	string	defaults to $\sigma(z) = 1$	

†Specify either `cutoff_freq` or `omega_c`, but not both.

Table 4-22. – SamplingRandom function parameters.

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 8.3.12. 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_summary” in the “ECHO” section of the input file (see Section 9.9).

4.10.8. RandomLib Functions

There are two tests of the RandomLib function. One is a verification test.

In many cases, a random load on a structure may need a spatial correlation with other loads on the structure. The **RandomLib** function was created to address this need. [2]

[2]The RandomLib function is an external library interface to **Sierra/SD**. Additional functionality and the interface to other applications are described in separate documentation.

Run time parameters for “RandomLib” functions are listed in Table 4-23, and an example is provided in input 4.12. Each parameter is described in more detail below.

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 boundary conditions on a nodeset that covers the same nodes as the sideset. This nodeset is assumed to have the name “surface_1_nodes” where the “1” in this case corresponds to sideset 1. These nodal loads are typically generated within MATLAB code and merged with the **Exodus** file definition.

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	*int/string*	*sideset id/name where pressures are applied*

Table 4-23. – RandomLib function parameters.

```
FUNCTION 55
  type=randomlib
  interp=none
  sideset=1
  exo_var scalar pressure
END
```

Input 4.12. Example RandomLib Function Specification

type The specification “`type=randomlib`” is *required* to reference the `randomlib` function and its capabilities.

interp A restart file contains time samples of a random function. **Sierra/SD** references these values at each time step to properly load the function. Figure 4-13 shows how interpolation influences the actual value returned.

sideset Pressure is applied over a single sideset of the model. This sideset must match the definition in the load section. [1]

[1]Data for the **Exodus** file is usually provided using specialized tools such as `mkrandloadrst`. A 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. Actual time history data is associated with a nodeset which includes the same nodes as the sideset.

exo_var Specification of the name of the nodeset variable that represents the nodal loading keyword `exo_var` is required. The format should be "`exo_var scalar pressure`," where 'pressure' is the name of the variable used in the **Exodus** file.

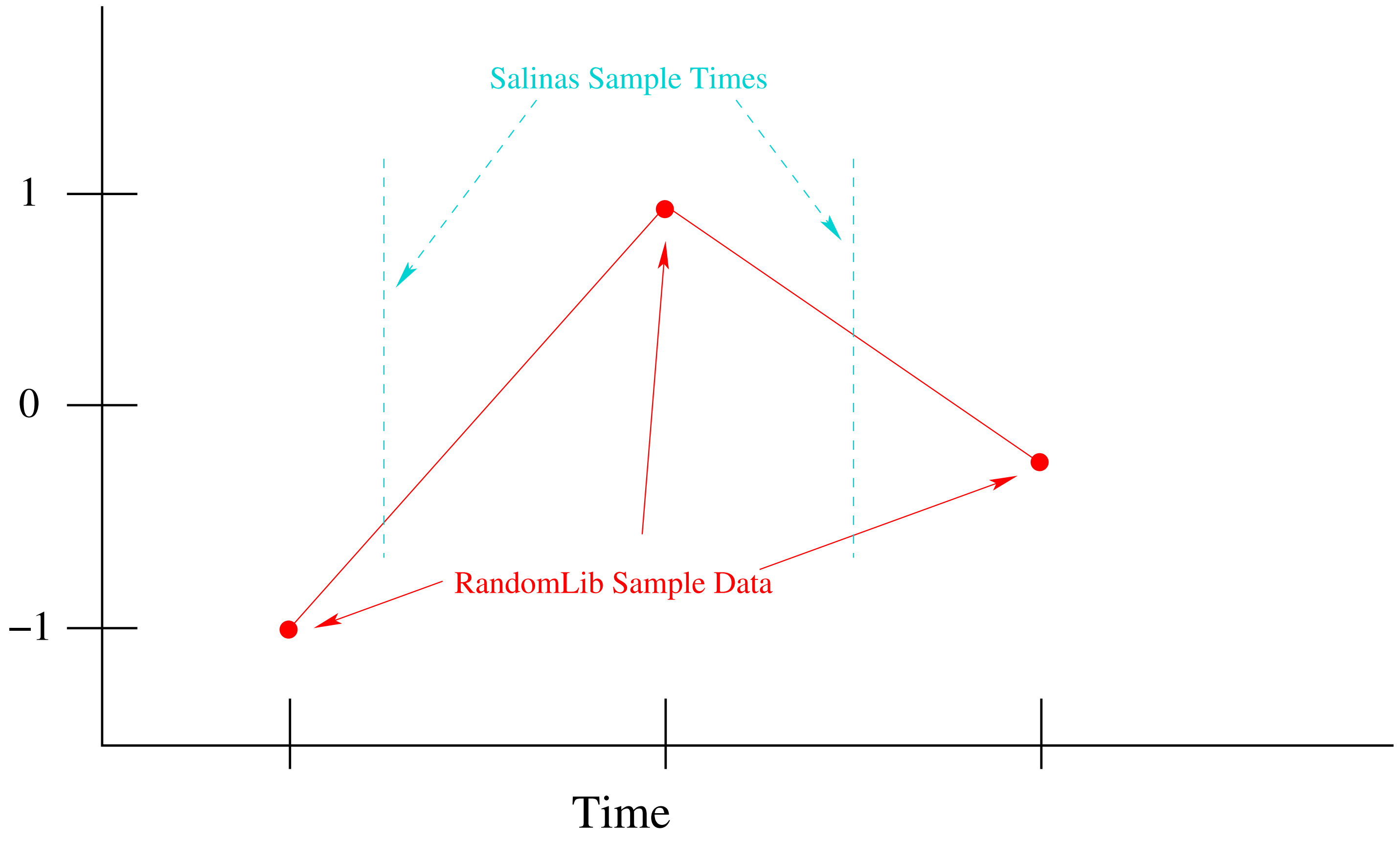


Figure 4-13. – RandomLib Temporal Interpolation. Because of different time steps in the RandomLib data library and the **Sierra/SD** time step algorithm, the function value returned depends on the time interpolation algorithm. With `interp=none`, the first value returned to **Sierra/SD** is about -1.0, 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.

4.10.9. Analytic Functions

Analytic functions can be used define complex mathematical functions directly in the input deck. An example of the input for this type of function is:

```
FUNCTION sine
    type analytic
    evaluate expression = "sin(2 * pi * t)"
END
```

Sierra/SD uses the STK[56] expression parser.

Analytic functions can be used in a variety of contexts such as boundary conditions, material properties, or user output.

4.10.9.1. Analytic Expression Parser Syntax

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

Note: analytic function variables are case *insensitive*, including any local variable definitions, and there is currently no warning for multiply-defined variables (the last variable name “wins”), so caution should be taken to ensure that duplicate variable names are not used.

Expressions compute an output value as a function of input values. The input value of an expression is called an independent variable of the expression. An expression can use any name for the expression independent variable. The examples shown here use `x` for the independent variable.

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

```
function sinx
  type is analytic
  evaluate expression is "sin(x)"
end
```

Example: Return a piecewise linear interpolated ramp function. For x values less than 0.0 the function will return 0.0. For x values between 0.0 and 0.5 the function will return y values linearly interpolated between 0 and 100. For x values greater than 0.5 the function will return 100:

```
function pressure
  type is analytic
  evaluate expression is "   \#
    (x <= 0.0) ? (           \#
      0.0                    \#
    ) : (                    \#
      (x < 0.5) ? (          \#
        x*200.0              \#
      ) : (                  \#
       100.0                 \#
      )                      \#
    )                        \#
   "
end
```

Operators and Functions Available within Expressions The following functionality is currently implemented for the expressions:

Operators Valid arithmetic and Boolean operations are as follows:

Symbol	Operation
`+, -`	plus, minus
`*, /`	times, divide
`^`	power
`==, !=`	equivalent, not equivalent
`>, <`	greater/less than
`>=, <=`	greater/less than or equal to
`!`	not
`&&, ||`	and, or
`?, :`	ternary if, then, else

Parentheses `()`, represents standard mathematical meaning of operation ordering.

Component indexing `value[i]`, accesses the *i*th component of a multi-component variable such as a vector, tensor, etc. This index is one-based. For a vector `vec`,

```
vec_x = vec[1]
vec_y = vec[2]
vec_z = vec[3]
```

for a symmetric tensor `sym`,

```
sym_xx = sym[1]
sym_yy = sym[2]
sym_zz = sym[3]
sym_xy = sym[4]
sym_yz = sym[5]
sym_zx = sym[6]
```

and for a full tensor `full`,

```
full_xx = full[1]
full_yy = full[2]
full_zz = full[3]
full_xy = full[4]
full_yz = full[5]
full_zx = full[6]
full_yx = full[7]
full_zy = full[8]
full_xz = full[9]
```

Math functions Valid mathematical operations are as follows:

Function	Operation
`abs(x)`	absolute value of `x`
`mod(x,y)`	modulus of `x|y`
`min(x,y)`	minimum value of `x`, `y`
`max(x,y)`	maximum value of `x`, `y`
`sign(x)`	sign operator: -1 if `x` is negative, 1 if positive
`ipart(x)`	integer part of `x`
`fpart(x)`	fractional part of `x`

Power functions Valid functions for expressions containing exponents are as follows:

Function	Operation
`pow(x,y)`	x^y
`pow10(x)`	10^x
`sqrt(x)`	$\sqrt{x}$

Trigonometric functions Valid trigonometric operations are as follows:

Function	Operation
`acos(x)`	arccosine of `x`
`asin(x)`	arcsine of `x`
`asinh(x)`	inverse hyperbolic sin of `x`
`atan(x)`	arctangent of `x`
`atan2(y,x)`	arctangent of `y/x`, signs of `x` and `y`
`cos(x)`	cosine of `x`
`cosh(x)`	hyperbolic cosine of `x`
`sin(x)`	sine of `x`
`sinh(x)`	hyperbolic sine of `x`
`tan(x)`	tangent of `x`
`tanh(x)`	hyperbolic tangent of `x`

Logarithm functions Valid logarithmic operations are as follows:

Function	Operation
`log(x)` or `ln(x)`	natural logarithm of `x`
`log10(x)`	base-10 logarithm of `x`
`exp(x)`	e^x

Rounding functions Valid mathematical rounding operations are as follows:

Function	Operation
`ceil(x)`	smallest integer value not less than `x`
`floor(x)`	largest integer value not greater than `x`

Random functions Valid functions for generating random data are as follows:

Function	Operation
`random()`	random real number $0.0 \leq x < 1.0$
`random(x)`	seeds the random number generator
`time()`	elapsed time in seconds since January 1, 1970

Coordinate conversion functions Valid functions for converting to and from polar and Cartesian coordinate systems are as follows:

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

If, then, else ternary syntax `A ? B : C`, where A is a Boolean statement such as `(x < 2.0)`, B is a statement (or set of statements) to be evaluated if A is true, and C is a statement (or set of statements) to be evaluated if A is false. **Note**: these statements can *only* be used for return values, i.e. you cannot define local variables inside a ternary statement.

Other utilities Miscellaneous other utility functions are as follows:

- `cos_ramp(x, xstart, xend)` defines a cosine ramp loading function. For `x < xstart`, it returns 0.0. For `x > xend`, it returns 1.0. If `xstart <= x <= xend`, the function has the value `(1.0-cos(Pi*(x-xstart)/(xend-xstart)))/2.0`. The primary purpose for the cosine ramp is to provide smooth loading. If attempting to run a nearly quasistatic problem with the dynamic solver, a displacement boundary condition applied via the `cos_ramp` will generally give a smooth response for a given loading time.
- `cycloidal_ramp(x, xstart, xend)` defines a cycloidal ramp function. For `x < xstart` it returns 0.0. For `x > xend`, it returns 1.0. If `xstart <= x <= xend`, the function has the value `(x-xstart)/(xend-xstart)-1/(2*pi)*sin(2*pi/(xend-xstart)*(x-xstart))`. The primary purpose for the cycloidal front ramp is to provide smooth loading for prescribed displacements and velocities. This function has continuous acceleration derivatives at `xstart` and `xend`, providing a smoother response than the `cos_ramp` for prescribed displacement boundary conditions.
- `haversine_pulse(x, xstart, xend)` defines a haversine pulse function. For `x < xstart`, it returns 0.0. For `x > xend`, it returns 0.0. If `xstart <= x <= xend`, the function has the value `pow(sin(Pi*(x-xstart)/(xend-xstart)),2)`. Design shock loads for components are often specified as haversine acceleration pulses; this function is included to make it easier to apply this loading.

Constants. There are three predefined constants that may be used in an expression. These three constants are `e`, `pi`, and `two_pi`. Note that these constants are reserved variable names and thus cannot be redefined in an expression.

Math symbol	Expression	Value
e	`e`	2.7182818284...
π	`pi`	3.1415926535...
2π	`two_pi`	$2 * 3.1415926535\ldots$

Also, there are two predefined constant functions that can be used, `SIERRA_CONSTANT_FUNCTION_ZERO` and `SIERRA_CONSTANT_FUNCTION_ONE`. These two functions are equivalent to defining functions with constant expressions “0.0” or “1.0”.

4.10.9.2. Input Variables for Analytic Functions

In their simplest form analytic functions have a single implicit independent input variable.

For example,

```
FUNCTION myFunc
  type analytic
  evaluate expression = ‘‘1 + y + y^2’’
END
```

```
FUNCTION left
  type analytic
  name "other_left"
  evaluate expression = "1500*pow(sin(1.0e5*t*pi/9),2)"
END
```

For these expressions, what `y` or `t` is will depend on context. For example, when defining material properties (section 6.5.6), it is the element centroid temperature. Typically, it would be time, such as if the function is used within a boundary condition or as part of a user output 9.5.6.

Alternatively, one or more input independent variables can be set for the function. Many advanced usages of analytic functions require this, for example a load that is a function of both time and spatial position. Independent variables to the analytic function are specified via `expression variable` commands.

Note that an "expression variable" line is required for *each* independent variable, even if the name and meaning happen to be the same (e.g. `expression variable time = time`). Also, note that analytic functions may be evaluated at each application node on the structure (i.e. each node in a nodeset).

Some quantities are predefined by **Sierra/SD** as global or nodal fields for the analysis. The list of supported input variables is given in table 4-24 and syntax 4.8.

⚠ explicit time input is currently BETA release.
Enable with the "- -beta" command-line option.

```
expression variable <string> = input|time|coord|disp|
 ↪ velocity|acceleration|nodeId
```

Syntax 4.8. Expression Variable Syntax

Variable	Description
input	Function input value. Could represent time, temperature, etc.
time	Current simulation time (transient only). (beta capability) ⚠
coord	Undeformed coordinates at each node.
disp	Deformation vector at each node.
velocity	Velocity vector at each node.
acceleration	Acceleration vector at each node.
nodeid	The global node ID at each node. Note: *unmapped* **Exodus** *node id* (1:N)

Table 4-24. – Predefined Analytic Input Variables.

The following is a spatial boundary condition example using multiple input variables:

```
FUNCTION spatialFunc
  type analytic
  expression variable c = coord
  expression variable t = input
  evaluate expression = ‘‘
    cx = c[1];
    cy = c[2];
    cz = c[3];
    t * sqrt(cx^2 + cy^2 + cz^2)’’
END
```

This function has two independent variables: the function input (typically analysis time) and nodal coordinates. The nodal coordinates then have three components X, Y, and Z. The return value of this function depends on the coordinate of each node and the input time.

Additionally, nodal, element, or global fields may be used in the "expression variable" line to define independent analytic function variables. Typically, this includes variables read from the input mesh or variables computed via a 'user output' command, or variables that are output to exodus output or history files.

Global expression variables is currently BETA release.
Enable with the "- -beta" command-line option.

```
expression variable <string> = nodal <string>
   ↪ (from OUTPUTS|HISTORY|FREQUENCY)
   ↪ (at time <real>)|(at step <int>)
expression variable <string> = element <string>
   ↪ (from OUTPUTS|HISTORY|FREQUENCY)
   ↪ (at time <real>)|(at step <int>)
expression variable <string> = global <string>
   ↪ (from OUTPUTS|HISTORY|FREQUENCY)
   ↪ (at time <real>)|(at step <int>)
```

Nodal variables may be used for functions that evaluated at nodes (e.g. boundary conditions or user-defined nodal fields), or element centroids (e.g. material properties or user-defined element fields). In contrast, element variables may only be used for functions evaluated at element centroids.

By default, functions used in user-defined variables (section 9.5.6) will look for the `expression variable` fields in the same region where the user-defined field is declared. In all other cases, **Sierra/SD** will look for the fields in the OUTPUTS region (which also includes any fields read from the input mesh). A `from` clause may be appended to the `expression variable` line to override this behavior and always read fields from a particular output region.

Fields that exist on the input mesh (or copied from a transfer mesh) will typically use data from the last step. To use values from a different step, an `at` clause may be may be appended to the `expression variable` line.

Caution: switching steps is expensive, so this option should be avoided for functions that will be called frequently (every element, every node, etc). In some cases, alternative commands may be used for the same effect, e.g. the `time` and `step` options to the TRANSFER `copy variable` command (section 4.3.2).

For example, the following function would apply a load based on a function of input (representing time) and an input mesh nodal variable (`adagio_force_x`).

```
TRANSFER mesh_with_data.exo
  copy variable adagio_force_x
    variable type = node
END
```

```
FUNCTION rx
  type analytic
  expression variable t = input
  expression variable v = nodal adagio_force_x at
  evaluate expression "v*sin(2*pi*t)"
END

LOADS
  nodeset 1 force 1 0 0 function rx
END
```

Finally, references to other functions may also be used in the "expression variable" line to define independent variables. There are some limitations on the types of functions that are valid when used as an expression variable. For example, any analytic functions must not directly use `disp`, `velocity`, or `acceleration` as their own expression variables (although the same effect can still be achieved via the general nodal expression variable interface, e.g. `expression variable dispX = nodal dispX`).

function expression variables is currently BETA release.
Enable with the “`--beta`” command-line option.

```
expression variable <string> = function <string>
```

For example the following that uses one function to convert a displacement from meters to millimeters and uses that result in another function to apply a displacement dependent force.

```
FUNCTION dispX_in_mm
  type analytic
  expression variable d = disp
  evaluate expression ‘‘1000*d[0]’’
END
FUNCTION fx
  type analytic
  expression variable t = input
  expression variable dx_mm = function dispX_in_mm
  evaluate expression ‘‘t^2 * disp_in_mm’’
END
LOADS
  nodeset 1 force 1 0 0 function fx
END
```

4.10.9.3. Limitations of Analytic Functions

Analytic functions cannot be used for acceleration boundary conditions in transient analysis as time-integration of analytic functions is not implemented.

Analytic functions are case-insensitive. PI, Pi, pI and pi are equivalent. If it is not, then that is a bug. Note however that if a user defines pi and Pi differently, the expression variable parser treats them as the same, using the last definition.

This shared **Sierra/SD** and **Sierra/SM** documentation explains two capabilities that differ in one subtle way. **Sierra/SM** "is" and "are" translates to **Sierra/SD** =. Although **Sierra/SM** syntax treats '=/is/are' equivalently, **Sierra/SD** syntax does not recognize "is" or "are."

In **Sierra/SD**, vectors are expected to be the base name with X/Y/X appended and tensors are expected to be the base name with either 6 symmetric, or 9 full field components appended.

Use of variables in analytic expressions that match function names, such as 'sin', 'cos', or 'time' are not recommended as this can cause parsing ambiguity.

4.10.10. Plane Wave (Time Domain)

Plane wave functions are tested in acoustic scattering problems. A load on a surface is analytically described as an incident plane wave in terms of the following parameters.

Keyword	**Values**	**Description**
type	plane_wave	identifier keyword
Direction	3 *reals*	wave direction $\boldsymbol{e}_k = \boldsymbol{k}_0/\lvert\boldsymbol{k}_0\rvert$
material	*string*	acoustic material
K0	*real*	wavenumber, $k_0 = \lvert\boldsymbol{k}_0\rvert$
origin	3 *reals*	wave origin, $\boldsymbol{x}_0$

An acoustic material specifies the wave speed c_0 and fluid density ρ_0, and the **load** specifies the pressure amplitude p_0. The angular frequency $\omega = 2\pi f$ determines the wave number $k_0 = \omega/c_0$. The corresponding time-harmonic plane wave and velocity are

$$p = p_0 \cos\left[k_0 c_0 t - \boldsymbol{k}_0 \cdot (\boldsymbol{x} - \boldsymbol{x}_0)\right] \qquad \boldsymbol{v} = \boldsymbol{e}_k \frac{p}{\rho_0 c_0}\,.$$

```
LOADS
sideset 1 // acoustic
  acoustic_vel = 1.0
  function = 65
sideset 2 // structure
  pressure = 1.0
  function = 65
END
```

Input 4.13. Example Planewave Function Specification

```
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

TIED DATA
  surface 1,2
END
```

Input 4.14. Planewave Continued

Without the `tied data` block, there would be no interaction between the acoustic domain and the structure. Instead, the boundary of the acoustic domain is rigid, and the scattered pressure field is from a rigid boundary instead of from a structure with the specified material properties.

4.10.11. Plane Wave (Frequency Domain)

Similarly, in the frequency domain, an applied plane wave is defined in terms of the following parameters.

Keyword	**Values**	**Description**
type	plane_wave_freq, iplane_wave_freq	identifier keyword
Direction	3 *reals*	wave direction $\boldsymbol{e}_k = \boldsymbol{k}_0/\lvert\boldsymbol{k}_0\rvert$
material	*string*	acoustic material
origin	3 *reals*	wave origin, $\boldsymbol{x}_0$

[h] The wave speed c_0 and the density ρ_0 are specified by the choice of material, the frequency f is specified in the `frequency` block, and the particle velocity amplitude $u_0 = p_0/\rho_0 c_0$ is specified in the **loads** block. See Example input 4.16. A time-harmonic plane wave (with the time-dependence dropped) can then be written as

$$p(\boldsymbol{x}) = p_0 e^{-i\boldsymbol{k}_0\cdot(\boldsymbol{x}-\boldsymbol{x}_0)} . \tag{4.10.1}$$

```
LOADS
  sideset 1 // acoustic
    acoustic_vel = 1.0
    function = 66
  sideset 1
    iacoustic_vel = 1.0
    function = 67
  sideset 2 // structure
    pressure = 1.0
    function = 66
  sideset 2
    ipressure = 1.0
    function = 67
END
```

Input 4.15. Example PlanewaveFreq Loads

```
FUNCTION 66
  type = plane_wave_freq
  Direction = 1 0 0
  origin = 0 0 0
  material = air
END
FUNCTION 67
  type = iplane_wave_freq
  Direction = 1 0 0
  origin = 0 0 0
  material = air
END
```

Input 4.16. Example PlanewaveFreq Specification

The syntax for the plane wave frequency function does not require a wavenumber, `K0` , but is otherwise the same as **plane_wave** function from the previous section. Note that both a real and an imaginary plane wave frequency function are applied to their corresponding real and imaginary loads—which is currently necessary to apply the correct phase shift between real and imaginary parts of propagating waves—and that real and imaginary loads should be given identical amplitude, direction, and origin.

4.10.12. *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 4-25. 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 $t' = \frac{\hat{d}\cdot[\vec{x}-\vec{x}_o]}{c_o}$.

$$P = P_o \cdot e^{-\beta\cdot(t-t')} H(t-t') \tag{4.10.2}$$

A standard planar step wave function can be defined by using $\beta = 0$. This is the default behavior if no beta parameter is specified.

Keyword	**Values**	**Description**
type	planar_step_wave	identifier keyword
Direction	3 *reals*	wave direction $\vec{d}$
material	*string*	acoustic material
origin	3 *reals*	wave origin, $\vec{x}_o$
beta	*real*	exponential decay factor, β

Table 4-25. – Planar Step Wave Parameters.

4.10.13. *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 the structure and a velocity load on an acoustic model. Parameters are listed in Table 4-26. Figure 4-14 illustrates the geometry.

Keyword	**Values**	**Description**
Type	spherical_wave	identifier keyword
origin	3 *reals*	wave origin, $\vec{x}_o$
reference_location	3 *reals*	reference location $\vec{R}$
material	*string*	acoustic material (alternate to C0)
pressure_function	*string*	new function for user supplied pressures

Table 4-26. – Spherical Wave Parameters.

A spherical wave is used only in transient dynamics analyses. An example input is described in input 4.17. 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.

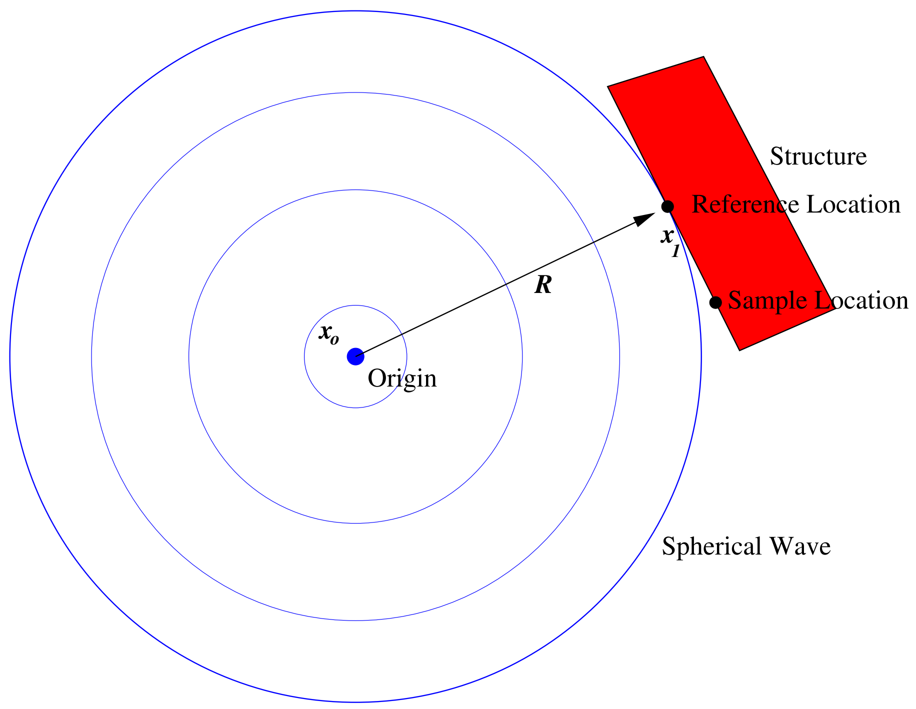


Figure 4-14. – Spherical Wave Geometry.

```
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
  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
```

Input 4.17. Spherical Wave Example

4.10.14. Undex Structural Acoustic Loads

For Navy scattering applications, the “multicycle_bubble” and " “Undex_shock_wave” functions provide a numerical function for analysis of exterior shock loading. The parameters of the loading are listed in Table 4-27. Details of the theory and implementation are available from the Navy Surface Warfare Center, Carderock Division (NSWC/CD). An example input is shown in input 4.18.

Keyword	Values	Default	Description
type	*single_decay* *double_decay* *hicks_bubble* *Undex_shockwave*	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	
material	*string*	required	acoustic material

Table 4-27. – Undex Load Parameters. input coordinates are in inches.

```
FUNCTION 67
  type = hicks_bubble
  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
```

Input 4.18. Example Hicks Bubble Function Specification

A “free_surface_flag” of one indicates generation of an applicable image source above the surface of the water, where a “free_surface_flag” of 0 indicates a load without a free surface. In this routine, “z” is 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.

One special type of shock wave (contained within the general “Undex” shock wave function definition), is that of a single decay shock wave. The single decay function has the following simple analytical solution:

$$P(w,r,t) = \begin{cases} 0.0 & t < \text{toa}(r) \\ \frac{P_{\max}(w,r)}{\exp\{(t-\text{toa}(r))/\theta(w,r)\}} & t > \text{toa}(r) \end{cases}$$

Where:

$$\begin{aligned} w &= \text{charge weight (lb. TNT)} \\ r &= \text{standoff distance (ft)} \\ t &= \text{time (seconds)} \\ c &= \text{sound speed in water (ft/s)} \\ P_{\max}(w,r) &= k_1 \left(\frac{\sqrt[3]{w}}{r}\right)^{a_1} \\ k_1,\ a_1 &= \text{similitude constants} \\ \theta(w,r) &= k_2 \sqrt[3]{w} \left(\frac{\sqrt[3]{w}}{r}\right)^{a_2} \\ k_2,\ a_2 &= \text{similitude constants} \\ \text{toa}(r) &= r/c = \text{time of arrival} \end{aligned}$$

4.10.15. Fluid Structure Interaction

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 the fluid. 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, use `restart = read` and/or `restart = write` 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 4-15 shows the infrastructure for FSI. There are many details to use of this coupling, such coupling is turned with the "acoustic_coupling" transient solution option.

Coupling to the Sierra code Fuego through Sierra Toolkit Transfers is currently being implemented. Users of current sprint releases must be careful to not use the "Fuego_coupling" transient solution option, as this is only partially implemented.

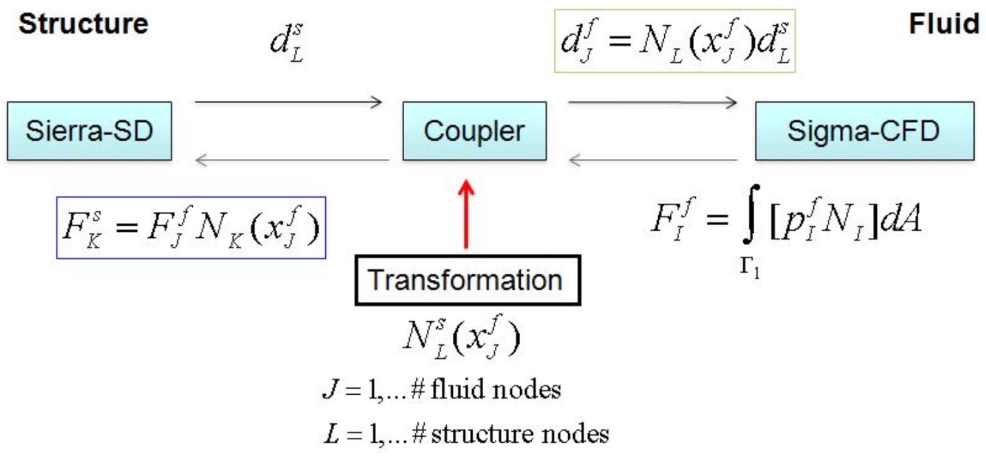

Structure
d_L^s
$d_J^f = N_L(x_J^f)d_L^s$
Fluid
Sierra-SD
Coupler
Sigma-CFD
$F_K^s = F_J^f N_K(x_J^f)$
$F_I^f = \int_{\Gamma_1} [p_I^f N_I] dA$
Transformation
$N_L^s(x_J^f)$
$J = 1,\ldots\#$ fluid nodes
$L = 1,\ldots\#$ structure nodes

Figure 4-15. – Fluid-Structure Interaction (FSI) Infrastructure.

4.10.16. Blending

In some applications, a combination of functions is necessary to accurately describe the loading. For example, a blended shock/bubble function is applied to a ship surface. The shock wave reaches the surface first, and is the effective loading until the bubble function arrives. Once the bubble arrives, the shock may be ignored, as it may not propagate through the cavitated region. This is illustrated for the first crossing case in Figure 4-16, and a typical input for this is shown in input 4.19. Parameters of the blended function are shown in Table 4-28. Several examples for the more general Nth crossing case are shown in Figure 4-17, with a typical input shown in input 4.20.

```
LOADS
  sideset 55
    pressure 1
    function 55
END
```

Keyword	**Parameter**	**Description**
Method	*string*	“first_crossing”, “second_crossing” or “Nth_crossing”
N	*int*	(for “Nth_crossing”) number of crossings
Primary Function	*string*	shock function
Secondary Function	*string*	bubble function

Table 4-28. – Blended Function Parameters.

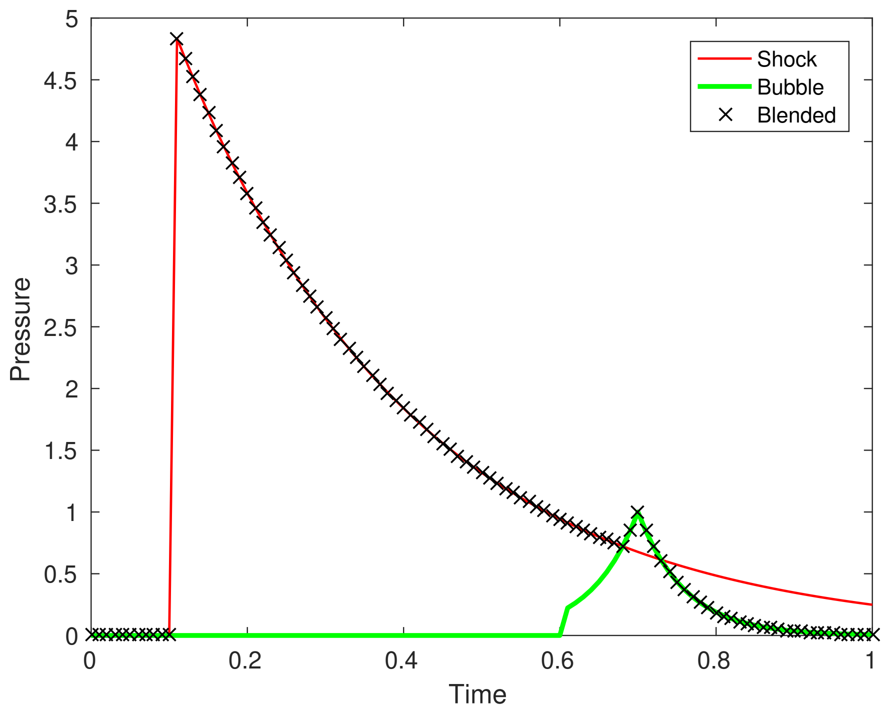


Figure 4-16. – Illustration of first crossing blended function.

```
FUNCTION 55               // blended shock/bubble
  type = blended
  method = first_crossing
  primary function 551
  secondary function 552
END

FUNCTION 551
  ...
END

FUNCTION 552
  ...
END
```

Input 4.19. Blended First Crossing Function Example

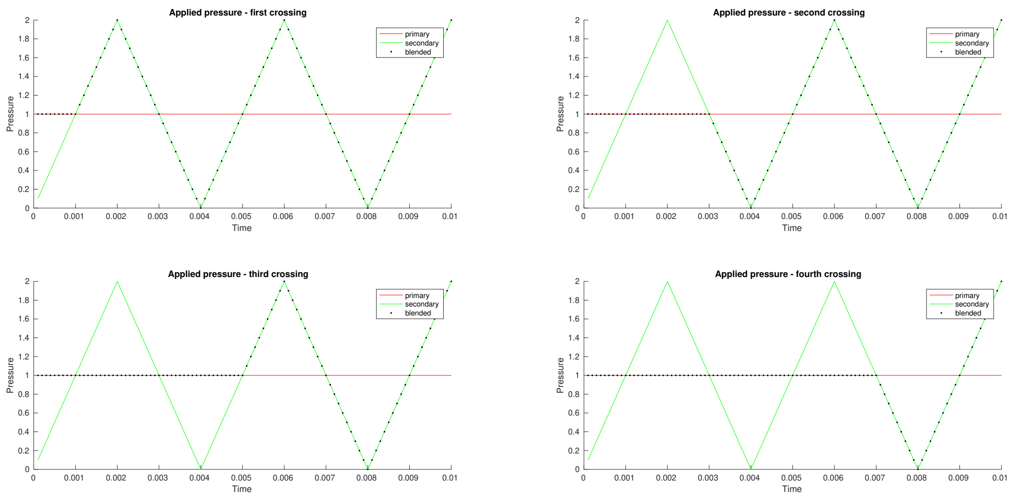


Figure 4-17. – Illustration of Nth crossing blended functions.

```
LOADS
  sideset 55
    pressure 1
    function 55
END

FUNCTION 55               // blended shock/bubble
  type = blended
  method = Nth_crossing
```

```
    N = 2
  primary function 551
  secondary function 552
END

FUNCTION 551
  ...
end

FUNCTION 552
  ...
end
```

Input 4.20. Blended Nth Crossing Function Example

4.10.17. 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. 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).

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 func11 scale 1.0
  data 1,2
    real function func12
    imag function func121 scale -3.0
  data 2,2
    real function func22 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", "full"/"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 1×1. 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 A data entry specifies one matrix-function entry. It must be immediately followed by the matrix location row and column pair. Again, no spaces may be inserted in the location entry. The keywords are real and imag.

- Real identifies entry real component. It must be followed by a function reference (see Section 4.10), and an optional scale factor.
- "Imag" identifies the entry imaginary component. It must be followed by a function definition, and an optional scale factor.

nominalt `nominalt` Used only for echoing the matrix values. If *input_summary* is specified as an "ECHO" option (see Section 9.9) 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.

4.10.18. Alternate Table Interface

An alternate `Table` input is provided. See Section 4.10.19 for details about tables. However, for many inputs, the individual specification of each function on each matrix element is both tedious and inefficient. Table input 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.

Application of table input to matrix-functions requires three tables: Real valued data, Imaginary valued data, and A table which associates each nonzero row and column of the matrix-function with appropriate rows of the real valued and imaginary valued data. Each has a keyword.

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 input 4.22.

The table entry has a fixed step. Each column must have the same number of values.

```
MATRIX-FUNCTION 1
  name 'spectral density'
  dimension=2x2
  symmetry=Hermitian
  real Table real_data
  imag Table imag_data
  Table INDEX index_data
END
```

Input 4.21. Example Matrix-Function

```
TABLE real_data
  size=3 550  // 550 freq samples, 3 matrix locations
  delta=1 0.5
  datafile='real_data.txt'
END
TABLE imag_data
  size=1 550  // 550 freq samples, 1 matrix location
  delta=1 0.5
  datafile='imag_data.txt'
END
TABLE index_data
  size=3 4
  rowfirst // transpose matrix for simpler input
  dataline // row col real imag
               1   1    1   0 // 1st real data, no imag
               1   2    2   1 // 2nd real data, 1st imag
               2   2    3   0 // 3rd real data row, no imag
END
```

Input 4.22. Example Matrix-Function Tables

Each matrix entry in the matrix-function must reference a row of a two-dimensional table. In the table columns 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”.

4.10.19. Table

A (1 dimensional) table is implemented by including a file with one value per line. 1-dimensional tables have identical behavior to linear functions (section 4.10.2) while typically being much faster and more memory efficient, especially for many data points.

A small section in the main input deck specifies the initial time and time increment. The data must be sampled at a uniform interval.

Tables are used by being referenced in other sections of the input deck. Tables offer support for multi-dimensional data (up to dimension 4). Note that tables of dimension greater than 1 are complicated.

Each Table includes some required and optional parameters, as shown below.

Table 4-29. – Table Section Options.

Parameter	Default	Description
dimension	*optional*	number of dimensions in the table
size	*required*	table size in each direction
datafile	*required*	ASCII 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 table shape. For example, `dimension=2` indicates a table of XY values. If this parameter is not defined, the dimension will be automatically inferred from the number of entries in `size`. If it is defined, it must match the inferred value, and thus only serves as a check on the table size.

The `size` parameters indicate the individual table hyper-cube dimensions. 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 text file containing the table data values is specified using the `datafile` parameter. Data values are separated by white space. The layout of the file is not important, but the order is important. The first dimension cycles the fastest. For a `dimension=2` table, the file list begins with the entries for column 1. The number of entries in the file must match the table size. Comments are not 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 specified. The format is identical to the datafile. It is efficient 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.

$$\text{origin}_i < \text{range}_i < \text{origin}_i + (\text{delta}_i \cdot \text{size}_i) \tag{4.10.3}$$

Evaluations of the table for regions outside the valid range will use of the value of the nearest data point, just as with linear functions.

In contrast to a **function** (see Section 4.10), tables require memory only as needed. All processors store the full input deck 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. 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 'multi_dimensional_table.txt'
END
```

4.11. Multipoint Constraints

Multipoint constraints (MPC) are constraint equation applied directly to the stiffness matrix. Some analysis codes treat them as pseudo elements. An MPC is not an element, and is inaccessible through **Exodus**. It is a displacement constraint,

$$\sum_i^n c_i u_i = r.$$

The (nonzero) coefficients c_i are real. The number of (nonzero) coefficients is assumed to be approximately 1. The u_i are displacement of degrees of freedom. By default, r vanishes.

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. For GDSW the constraints are specified as Lagrange multipliers which involves 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 inconvenienced analysts who must ensure that the constrained DOF are independent and never specified more than once.

Each MPC is specified in the input deck 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 [2]; see section 4.9. The MPC will be stored internally in the basic coordinate system (coordinate frame 0). The input consists of a triplet listing the node or nodeset, a degree of freedom string, and the coefficient of that degree of freedom. The degree of freedom 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.

GDSW is required for inhomogeneous MPCs. The solution method must not be QEVP. **Sierra/SD** will exit with a fatal error if a model including non-homogeneous MPCs violates either of these requirements.

The MPC can apply some general commands that apply to the whole MPC as shown in Table 4-30. Additionally, the MPC can include multiple lines that define the MPC equation entries as shown in Table 4-31.

Table 4-30. – General MPC commands.

Keyword	Type	Description
coordinate	*string*	Optional coordinate frame for MPC
rhs	*real*	Optional right-hand side constant

Table 4-31. – MPC Equation lines.

Command Line	Description
integer x|*y*|*z*|*rx*|*ry*|*rz real*	Equation entry using single global node id
nodeset *nodeset_id x*|*y*|*z*|*rx*|*ry*|*rz real*	Equation entry using nodeset with EXACTLY one node

In this first example the MPC is defined with respect to a coordinate system `dir1`. The displacement at the x degree of freedom of node 4 is constrained to be equal to the average x degree of freedom of nodes 2 and 3 plus 5, i.e. $u_{(4,x)} - 0.5 * u_{(2,x)} - 0.5 * u_{(3,x)} = 5.0$.

```
MPC
  coordinate dir1
  4 x 1.0
  2 x -0.5
  3 x -0.5
  rhs 5.0
END
```

[2]At this time, all the nodes in an MPC must be associated with the *same* coordinate system.

In this next example, the x degree of freedom of the node in the nodeset `101` is constrained to be equal to the y degree of freedom of the node in the nodeset with name `aft_plate`. Note that both nodesets must contain exactly one node.

```
MPC
  nodeset 101 x 1.0
  nodeset aft_plate y -1.0
END
```

Different algorithms are used to eliminate constraints in serial and in parallel. In some problematic cases, this leads to differences in simulation results. The only way to change constraint elimination algorithm is to switch between using one MPI rank or using more than one MPI rank.

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 increases robustness and solution performance with no penalty for accuracy. Thus, rigid links are recommended whenever possible in parallel solutions.

Replacing rigid links with stiff beams may cause ill conditioning and simulation inaccuracies.

5. SOLUTION CASES

Sierra/SD supports a wide variety of different analyses or solution methods. Input consists of an **Exodus** mesh file and a text input deck. Solution methods are specified in the text input deck in the solution section.

The `Solution` section defines the type of physics to simulate. Analysis types are shown in Tables 5-1, 5-2, 5-3, 5-4. Relevant options are given in the detailed description of each solution case. Also, general options are shown in Table 4-8 and are described in Section 4.6.

Table 5-1. – Eigenvalue Solvers.

Solution Type	Description
eigen	Modal solution to extract natural vibration modes of K,M
aeigen	Modal solution with Anasazi
buckling	Modal solution solving for buckling modes
CBR	Craig-Bampton reduction for creation of superelements
geometric_rigid_body_modes	Exact analytic definition of the rigid body modes
blk_eigen	Modal solution on a block by block basis
largest_ev	Largest eigenvalue of K,M
QEVP	Solution to quadratic eigenvalue problem

Table 5-2. – Modal Solution Types.

Solution Type	Description
ModalFrf	Frequency response using modal displacement or modal acceleration
modalranvib	Random vibration using modal superposition
modalshock	Shock response spectra using
modaltransient	Transient analysis using modal superposition

Table 5-3. – Direct Solution Types.

Solution Type	Description
Directfrf	Direct computation of frequency response functions
NlStatics	Nonlinear static solution
NlTransient	Nonlinear transient solution
statics	Static solution
transhock	Shock response spectra via direct implicit transient
transient	Linear transient solution

Table 5-4. – Preprocessing and Postprocessing Solution Types.

Solution Type	Description
receive_sierra_data	Input stress, displacement, and material state from a preload **Sierra/SM** analysis
model_check	Check input for errors, generate and output diagnostics, no solve
CJdamp	Approximate modal damping contributions
DDAM	Dynamic design analysis method (U.S. Navy)
preddam	Gather data for use by DDAM solution case
gap_removal	Do contact search and apply the contact gap removal algorithm
superposition	Expand superelement results to physical degrees of freedom
tangent	Compute tangent stiffness after a nonlinear load step
tsr_preload	Input data for thermal structural response
fatigue	Postprocess modal random vibration results to predict high cycle fatigue life
MPF	Compute and output modal participation factors
waterline	Determine waterline of a floating structure

5.1. Defining Solution Cases

It is important to distinguish the two different kinds of Solution section. A Solution section may do one analysis. In this case a `loads` section is used. Or a Solution section may do multiple analyses. Here each case may specify a `load` section. Here's an example of a Solution section that does one analysis. The eight lowest eigenvalues of a structure are requested.

```
SOLUTION
  eigen
  nmodes 8
END
```

A more complex analysis could perform a static preload, followed by computation of updated tangent stiffness, and then a (linearized) eigendecomposition with this input:

```
SOLUTION
  case 'NonlinearStatics'
    nlstatics
    load=10
  case 'tangent'
    tangent
  case 'eig'
    eigen
    nmodes 8
END
```

5.1.1. *Multicase Solutions*

The solution methods of Tables 5-1, 5-2, 5-3, and 5-4 may be a part of a multicase solution. Using one input deck, it is possible to run any valid sequence of simulations. For example, a static preload, followed by computation of updated tangent stiffness, and then a (linearized) eigendecomposition.

In a multicase analysis, each analysis subsection begins with the word `case` followed by a unique label. Cases are run sequentially from top to bottom. Optional indentation to emphasize this hierarchical relationship is recommended. The label is used to set the output file name and is used in a variety of informational, warning, and error messages. Note: the label name for each case must be unique.

The results and outputs from one case will be used to drive subsequent cases. For example the modes computed by an **eigen** solution case will be used in a subsequent **modaltransient** solution case. As another example the stresses and displacements read in from `receive_sierra_data` solution case will be used to define the initial geometry and geometric stiffness for computing eigenvalues.

In a multicase solution, the system matrices (mass, stiffness and damping) will typically be computed once. Matrix updates between solutions may be specified by selecting the **tangent** keyword (see Section 5.26).

5.1.2. *Multicase Options*

A solution type has specific options. Only transient solution cases have time steps for example. On the other hand, Table 5-5 shows options that may apply to each solution case. They 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 to that sub-block. In the example given in 5.1.3, the `restart` options are thus `none` for most sub cases, `buread` for the eigen analysis and `read/write` for the linear transient.

Table 5-5. – Multicase Options.
These parameters may be specified as defaults above the case specifications, or they may be specified for each case to which they apply.

Option	Description	Options
restart	Restart options	see Section 4.6.2
solver	selection of solver	see Section 4.6.3

The default load applied to each solution case in a multicase solution is controlled by the input deck `Loads` section 4.5. An input deck may specify a sequence of solution cases. In order to specify a load for a specific solution case only, in addition an input deck may solve contain one or more labeled `Load` sections. A load specified in a `load` section overrides the load in the **loads** section. A particular solution case simply specifies the load by name. In the example given in 5.1.3, load ‘10’ will be applied during the nonlinear statics solution case (case ’Nonlinear_Statics’). Generic syntax for the Load and Loads input deck sections is presented in the General Commands section 4.5.

5.1.3. *Multicase Example*

In the example which follows, a nonlinear statics computation is followed by a tangent stiffness matrix update. An eigendecomposition of the updated matrix is computed. Two sets of **Exodus** output files will be written. Output from the statics calculation will be in files of the form *‘example-nlstatics.exo’*. Eigenvalue results will be in the form *‘example-eig.exo’*. The **tangent** solution normally produces no output in the **Exodus** format.

`Transient` and **modaltransient** solution cases are run sequentially in time. For the example below ‘Trans1’ will start at time 0.0 and step through 100 steps of at a step size of 1e-8 and then through 4000 steps at a step size of 1e-6. The final time value of case ‘Trans1’ will be $10^{-8}10^2 + 10^{-6} \times 4\ 10^3 = 0.004001$. Case ‘Trans2’ will start at 0.004001 and run an additional $10^{-4} \times 210^2$. This will end at time 0.024001.

```
SOLUTION
  restart=none
  title='example multicase'
  case 'nlstatics'
    nlstatics
    load=10
  case 'tangent'
    tangent
  case 'eig'
    eigen
    restart=read
  case 'Trans1'
    transient
    restart=read
    restart=write
    time_step 1e-8 1e-6
    nsteps    100  4000
    flush 50
    rho=0.9
    load=20
  case 'Trans2'
    transient
    restart=read
    restart=write
    time_step 1e-4
    nsteps    200
    flush 10
    load=20
END
```

Input 5.1. Multicase Example with Trans1 and Trans2

5.2. CJdamp Solution Case

Parameter	**Type**	**Default**	**Description**

Table 5-6. – CJdamp Solution Case Parameters.

The `CJdamp` solution provides a method to compute the equivalent modal damping terms introduced from material damping in lightly damped viscoelastic materials.[31] `CJdamp` is an approximate method which assumes that the mode shapes and frequencies are not modified by the damping. The modal damping is related to the fraction of energy in block.

The CJdamp method is effectively a post-processing step following an eigendecomposition and must be run in a multicase solution. 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.

$$SE^i_j = \sum_{elem}^{\text{in block j}} \phi_i^T K^{elem} \phi_i \tag{5.2.1}$$

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

$$R^i_j = SE^i_j / TSE^i \tag{5.2.2}$$

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

$$\zeta_i = \frac{1}{2} \sum_j R^i_j \eta_j(f_i) \tag{5.2.3}$$

$\eta_j(f_i)$ is the CJetaFunction contribution from block j evaluated at the natural frequency of mode i. More information is available in Section 6.5.9.

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

An example follows:

```
SOLUTION
  case eig
    eigen
    nmodes=30
  case Johnson
    CJdamp
  case frf
    ModalFrf
END
```

5.3. Craig-Bampton reduction Solution Case

Parameter	Type	Default	Description
nmodes	*Integer*	10	Number of modes to extract. See section 5.8.1
shift	*Real*	-1.0e6	Shift to apply to matrix system to allow solving singular systems. See section 5.8.2
untilfreq	*Real*	Inf	Target frequency to reach. See section 5.8.3
correction	*node|values| vectors*	values	Correction method for rigid body modes
RbmDof	*string*	123456	Defines which rigid body modes to which correction=vectors applies
use_residual_vectors	*File*	false	Activates residual vector calculations.
resid-ual_vectors_node_list_file	*File*	N/A, no file read	Only valid when use_residual_vectors is `true`. Designates the nodelist file to be read from to calculate residual vectors.
resid-ual_vectors_solver_options	*String*	N/A, uses default solver options	Only valid when use_residual_vectors is `true`. This allows a user to designate a `SOLVER_OPTIONS` section specific to the residual vector calculation. If a `solver_options` parameter is used for the eigen solution, it will not be applied to the residual vectors calculation unless it is also specified with the resid-ual_vectors_solver_options parameter.
ModalFilter	*string*	none	Modal filter to define modes to retain. See section 5.8.4

Table 5-7. – Craig-Bampton reduction Solution Case Parameters.

It can be advantageous to reduce a model to its interface degrees of freedom. Reducing a subsystem model to a linearized, Craig-Bampton model can greatly improve computation time of subsequent analysis and potentially make it possible to share 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. These terms are defined as:

Fixed interface modes. These are eigen modes of the structure if we *fix* the interface, by setting interface degrees of freedom to zero. These modes are represented by Φ. The analyst decides how many of these modes to retain.

Constraint Modes These are the response of the structure if all interface degrees of freedom are clamped except one. That degree of freedom has an imposed displacement of 1.0. These are not `modes` in the usual sense, but they provide a spatial basis. Represented by Ψ, there are as many of these constraint modes as there are interface degrees of freedom.

The **Craig-Bampton reduction** solution reduces an entire structural model to its reduced system and transfer matrices. Solution Parameters are listed in the table above, and correspond to the parameters required for an eigendecomposition (Section 5.8). In addition, a **CBModel** section must be defined elsewhere (see Section 5.3.1). Any boundary conditions specified are applied before reducing the model.

We note that sensitivity analysis can be performed in **Craig-Bampton reduction**, though the process is somewhat different from other types of sensitivity analysis. Section 5.3.1 contains more information about sensitivity analysis in Craig-Bampton models.

The method will write system matrices and general information. Each parameter 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. **nmodes** selects the number of fixed interface modes. The fixed interface modes are eigenvectors 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 accuracy. See section 5.8.1 for more information.

untilfreq: Alternative to nmodes for specifying fixed interface modes. See section 5.8.3 for more information.

shift: Used in conjunction with `nmodes` or `untilfreq` for the solution of singular eigen systems. See section 5.8.2 for more information.

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 eigenvector space, but the negative eigenvalues will be adjusted to zero.

vectors This option is available in models that have no boundary conditions besides those associated with the constraint modes, so that the CBR model is floating and has six rigid body modes. If boundary conditions are present, there is a fatal error. Additionally, if the model does not have six rigid body modes for another reason (for example a gap contact constraint that impedes rotation) then the correction=vectors option is not well posed, should not be used, and could cause a serious degradation of model behavior. The correction=vectors will compute zero-energy eigenvectors geometrically (exactly), 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 improve the usefulness of the resulting model.

If `correction=vectors` is selected, one may also optionally determine which of the 6 zero-energy modes are used. `RbmDof` is the parameter to use to select those modes. It 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.

As an example,

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

MPCs may not share nodes with interface nodes in CBR.

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 dimension of the model is the sum of the numbers of constraint and fixed interface modes.

Static solutions with all interface degrees of freedom clamped are part of the reduction process. If the interface dofs do not fully constrain the system, then the linear system may be singular. In such cases the solution is not reliable. Due diligence includes verifying the reduced order eigendecomposition against the full system. One may also compare the retained mass.

The CBR solution is not tested with preload or any other multicase solutions besides a preceding eigen solution.

Inertia Tensor for Craig-Bampton Reduction

A reduced inertia matrix, $\boldsymbol{I}_{vv}$ may be output from a CBR (Craig-Bampton Reduction) analysis. The Inertia Tensor may be used to apply initial conditions to the superelement in some applications. The input deck syntax is described in the **CBModel** section, 5.3.1. $\boldsymbol{\Phi}$ is the matrix of mode shapes used for the CBR analysis. It consists of both fixed-interface modes and constraint modes. $\boldsymbol{I}_{vv}$ is defined by

$$\boldsymbol{I}_{vv} = \boldsymbol{\Phi}^T \boldsymbol{R},$$

The number of rows in $\boldsymbol{I}_{vv}$ is the number of CBR modes, and the number of columns is the number of rigid body vectors. For example, for the three translational rigid-body modes and assuming three degrees-of-freedom per finite element node,

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

Mass Inertia Matrix for Craig-Bampton Reduction

The Mass Inertia Matrix is used in some applications to apply a load to the interior degrees of freedom of a model. [1] The mass inertia matrix is defined as,

$$I_m = \boldsymbol{\Phi}^T M \boldsymbol{R}.$$

where M is the mass matrix of the unreduced model.

[1]Neither the inertia tensor, nor the mass inertia matrix may be applied in **Sierra/SD**. They are output quantities.

5.3.1. *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 described in Section 5.3.

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 5-8. – CBModel Parameters.

Keyword	type	Description
nodeset	*int/string/list*	**Exodus** nodeset name(s) and/or id(s).
sideset	*int/string/list*	**Exodus** sideset name(s) and/or id(s).
format	*string*	specifies the output format. MATLAB - MATLAB.m format DMIG - NASTRAN DMIG format DMIG* - NASTRAN long DMIG format netcdf - netcdf format†
file	*string*	specifies the file name for output.
GlobalSolution	*bool*	‘yes’ to compute the eigenvalues of the reduced system.
inertia_matrix	*bool*	‘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
spoint_offset	*integer*	offset for spoints in DMIG format

†The netcdf format is the database upon which exodusII is built. `SEACAS` tools, MATLAB and Python can all read netcdf files.

The input deck keywords shown in Table 5-8 are described below.

nodeset: The **nodeset** keyword specifies the nodes to be placed in the interface. Nodesets are defined in the **Exodus** file. A nodeset ID or name 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 9.2, and follows the rules for integer lists detailed in Section 4.1.

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 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. It contains only the reduced system matrices (no maps, coordinates, etc). The MATLAB format is a convenience.

file: The `file` keyword is required to specify the output file name.

GlobalSolution: As a convenience, we will optionally compute the eigenvalues 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. Moreover, users are strongly advised to review section Craig-Bampton Model Reduction of **Sierra/SD** How To[44] before computing superelements.

inertia_matrix: The inertia matrix defined in Section 5.3 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.

spoint_offset NASTRAN DMIG output defines "spoint" variables that store generalized degrees of freedom. The identifiers the generalized output variables must be unique. The range of identifier output may be specified with this option. By default, "spoint_offset" is 10000 which means spoints are numbered as 10001, 10002, 10003, etc.

Specify "displacement" in the **outputs** section (9) to output the constraint modes and fixed interface modes that were used as a basis to generate the reduced order system to the **Exodus** file. First the fixed interface modes are output, followed by the constraint modes. The eigenvalues of the fixed interface modes correspond to the mode shapes. For the constraint modes an integer index replaces the eigenvalues. These modes may be visualized and evaluated using any of the standard tools.

Data in Table 5-9 can be written to a file. NumC, NumEig, Kr, Mr, Cr, and cbmap are written by default, but may contain zeros if applicable.

OTM Output is currently BETA release.
Enable with the "`--beta`" command-line option.

The Output Transfer Matrix (or OTM) depends on data in the `history` section (see Section 9.2). Specifically, the output nodes and elements, and the output variables need to be specified in the `history` section in order to activate OTM output. Without all of those data points, the OTM will not be written. OTM output is also only available with the "–beta" flag. 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. Other arguments in the history section will be ignored. We think of the OTM as having 6 parts.

$$T = \begin{bmatrix} \Phi_u & \Psi_u \\ \Phi_\epsilon & \Psi_\epsilon \\ \Phi_\sigma & \Psi_\sigma \end{bmatrix}, \quad \begin{bmatrix} u \\ \epsilon \\ \sigma \end{bmatrix} = T \begin{bmatrix} q \\ u_\Gamma \end{bmatrix}. \tag{5.3.1}$$

Table 5-9. – Data output for Craig-Bampton Reduction.

Variable	Description
NumC	number of constraint modes
NumEig	number of fixed interface modes
Kr	Reduced stiffness matrix.
Mr	Reduced mass matrix with **wtmass** applied.
Cr	Reduced damping matrix. Only available for dashpots and block proportional damping.
cbmap	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. **Number** — **Description** 1 — **x** 2 — **y** 3 — **z** 4 — Rotation **x** 5 — Rotation **y** 6 — Rotation **z** 7 — acoustic pressure The "cbmap" has the same number of rows as Kr or Mr.
OutMap	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. This is only output with a properly populated history section and the "–beta" flag.
OTM	Output Transfer Matrix to provide a transfer function from the interface dofs to internal degrees of freedom or other results. This is only output with a properly populated history section and the "–beta" flag.
OutElemMap	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.This is only output with a properly populated history section and the "–beta" flag.
OTME	Output Transfer Matrix to provide a transfer function from the interface dofs to internal elements. This is only output with a properly populated history section and the "–beta" flag.

The amplitude q of the internal constraint modes is typically computed in the next level analysis. Also, u_Γ is the vector of interface displacements. The fixed interface modes (eigenvalues 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
  file='junk.ncf'
END
```

The reduced inertia matrix $\boldsymbol{I}$ for Craig-Bampton Reduction defined in Section 5.3 may be computed and written to the super-element files with the reduced mass and stiffness matrices. $\boldsymbol{I}$ can be written to the results file in either netcdf, MATLAB or DMIG format. In the `CBModel` section

```
CBMODEL
  nodeset 1
  format = netcdf
  file = model.ncf
  GlobalSolution = yes
  inertia_matrix = yes
END
```

The inertia_matrix = yes line requests the output of the inertia tensor. The default value of `inertia_matrix` is no.

NOTE:
The OTM output capability 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). The standard CBR output matrices can be used to build a superelement in **Sierra/SD**, *so a user can perform a Craig-Bampton reduction to generate a reduced order model of a portion of the structure. A follow-up analysis could use this as a superelement. See details in Figure 5-1.*

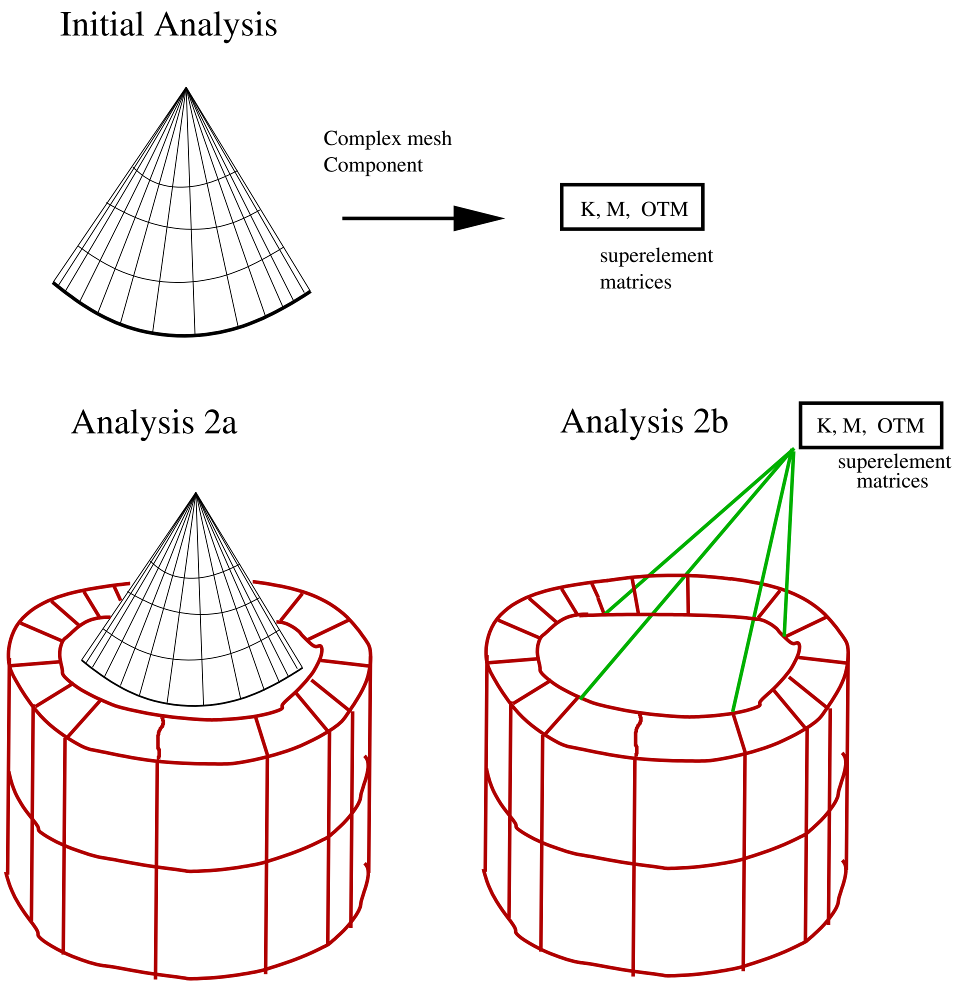


Figure 5-1. – An initial analysis using CBR can be applied to reduce a complex component to much smaller matrices. In subsequent analyses the superelement replaces the complex component in the system analysis. There is little loss of accuracy, but significant computational benefit.

5.3.2. *Sensitivity Analysis*

Sensitivity output for Craig-Bampton reduction requires both the **sensitivity** block of Section 4.8, and the `sensitivity_method` keyword in the `CBModel` block. The default `sensitivity_method` is constant vector.

The output differs from 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 CBR Sensitivity Analysis discussion in the section Solution Procedures of the Theory Manual for further details.

The reduced stiffness matrix $\kappa = T^T KT$ is computed from the Craig-Bampton transformation matrix, T and the stiffness matrix, K. The similar expression determines 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 `sensitivity_method` approach to the sensitivity of κ with respect to a parameter p ignores the dependence of the transformation matrix $T = T_o$ on p.

$$\frac{d\kappa}{dp} \approx \frac{T_o^T \left(K(p+\Delta p) - K(p)\right) T_o}{\Delta p}$$

The `finite_difference` method uses forward differences. 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

$$\frac{d\kappa}{dp} \approx \frac{T_1^T K(p+\Delta p)T_1 - T_o^T K(p)T_o}{\Delta p} \tag{5.3.2}$$

As long as the system has no repeated modes, this approximation converges to the sensitivity as Δp goes to zero. If there are repeated modes in the transformation matrix T, then the perturbed transformation matrix T_1 will re-order the repeated modes contributing to T_o. This corrupt the difference operation in equation 5.3.2.

As the constant vector method uses T_o only, for systems with repeated modes the constant vector method is recommended.

Sensitivity analysis of a Craig-Bampton model has different output from other types of sensitivity analysis. Depending on the `format` parameter (see Table 5-8), it is written in either MATLAB or **netcdf** format. The default is `netcdf`.

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

$$\frac{\partial\kappa}{\partial p_1}, \frac{\partial\kappa}{\partial p_2} \tag{5.3.3}$$

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 5.3.3 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 7.32.

5.4. preddam Solution Case

Parameter	**Type**	**Default**	**Description**
ModalFilter	*string*	none	Modal filter to define modes to retain. See section 5.8.4
load	*string*	none	Load section for gravitational loading

Table 5-10. – preddam Solution Case Parameters.

The `Preddam` solution case is intended to be run as part of a multicase solution, as a preparatory step for a subsequent `DDAM` analysis (section 5.5). It should be preceded by an `eigen` solution case (section 5.8), but does not necessarily have to be followed by a `DDAM` solution case.

`Preddam` utilizes the eigenvectors and system mass matrix produced in the preceding `eigen` solution to calculate and filter the modal participation factors, modal weights, individual modal weight percentage, cumulative modal weight, and cumulative modal weight percentage. See table 5-10 for a list of valid `Preddam` section parameters.

`ModalFilter` is implemented as a part of **Sierra/SD** and may be used as a part of other solution methods. It provides a means of filtering data taken from the modal analysis and the participation factors. More information may be found in sections 5.8.4 and 5.13.

A `load` block is required. It applies to the value of gravitational loading (-386.4). The direction must match DDAM analysis direction defined in the `ModalFilter` block and in the subsequent `DDAM` solution case.

5.5. DDAM Solution Case

Parameter	Type	Default	Description
analysis_direction	*vertical*| *athwartship*| *fore_aft*		`vertical` → Z-direction `athwartship` → Y-direction `fore_and_aft` → X-direction
ship_type	*surface_ship*| *submarine*		See NAVSEA documentation
mount_type	*hull*|*deck*| *shell_plating*		See NAVSEA documentation
response_type	*elasticplastic*| *elastic*		See NAVSEA documentation
velocity_coeffs	<*real*>(4)		See NAVSEA documentation
acceleration_coeffs	<*real*>(4/5)		See NAVSEA documentation

Table 5-11. – DDAM Solution Case Parameters.

The U.S. Navy Dynamic Design Analysis Method (DDAM) is an established procedure employed in the design of 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** User's manual and in the DDAM Primer. The user is expected to be fully familiar with both NAVSEA publications.

DDAM is focused on five main phases: problem formulation, mathematical modeling, coefficient computation, dynamic computation, and evaluation. DDAM as implemented in **Sierra/SD** focuses on the *evaluation* phase.

A DDAM analysis in **Sierra/SD** is divided into three solution cases: case 1 (`eigen`), case 2 (`Preddam`), and case 3 (`DDAM`). Case `DDAM` may only be run following `Preddam` and `eigen`.

`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 outputs (force, displacement, stress, etc.), and the NRL sums of those outputs.

Nodal displacements, velocities, accelerations, and forces; element-wise stresses and derived quantities; and NRL sums of the above are all written to the output **Exodus** file using the `ddamout` output keyword. See section 9.7.28 and in particular section 9.7.28.

Some commonly used post-processing tools for **Exodus** outputs are Paraview for graphical visualization and `explore`, `blot`, and `exo2mat` for data extraction.

An example input follows.

```
SOLUTION
  case modal
    eigen
      nmodes = 4
  case filter
    preddam
      ModalFilter VERTICAL
      load 1
  case
    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

OUTPUTS
  ddamout
END
```

Note: Ship directions (**analysis_direction**) must match coordinate directions. See table 5-11 for the meaning of each value. The direction must also match the `ModalFilter` and `load` sections used in the `Preddam` solution case.

DDAM has 6 main capabilities in **Sierra/SD**.

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

extra modes, in the `ModalFilter` block use `add`.

2. Incorporate a 6g minimum load requirement. This is not a user option.
3. Complex models and multiple element types.
4. Superelement integration.
5. Symmetric boundary conditions.
6. Parallel computation.

The user may verify `Preddam` (section 5.4) and `DDAM` by examining filtered modes, participation factors, modal weights, shock design coefficients, and values found in the following text files:

1. `Preddam` ⟶ `PREDDAM_RESULTS.txt`
2. `DDAM` ⟶ `DDAM_RESULTS.txt`

> There are several limitations to the DDAM approach. The equipment to be analysed must be represented as a linear elastic system with discrete modes. Also, damping is neglected. For very low frequency (VLF) systems, DDAM may not be appropriate, and, where closely-spaced modes exist, DDAM may produce excessive responses.

5.6. Direct Frf Solution Case

Parameter	**Type**	**Default**	**Description**
interpolate points	*integer*	0	Number of additional points to interpolate for Padé expansion. If zero, no interpolation is performed.
interpolate order	*integer*	20	Order of the rational function for Padé expansion.
load	*LOAD ID*	N/A	Load(s) to apply
flush	*integer*	50	Defines how often results are written to the exodus results file. See Section 4.6.1

Table 5-12. – Direct Frf Solution Case Parameters.

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.,

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

where $\overline{u}$ is the Fourier transform of the response u, and $\overline{f}$ is the Fourier transform of the applied force. The matrix equation is then solved for each frequency. When a direct solver is used, this means that a complex factorization must be performed once per output. This is time-consuming, and the **ModalFrf** may be a better option for many situations (see Section 5.15).

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 frequency response function is evaluated at the frequencies specified as described in Section 9.3. In a `frequency` section set `freq_step`, `freq_min`, and `freq_max`. Also, set an application region. Examples are presented in inputs 5.8, 5.9 and 5.13.

In addition to the output that is sent to the `frq` file, output may also be written to the **Exodus** file, provided that the keywords (such as acceleration) are specified in the **outputs** section. If nothing is specified in the **outputs** section, then nothing is written to the **Exodus** output files.

The expression "frf" is often interpreted as the *ratio* of output/input. There are 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.

5.6.1. Multiple Loads

Multiple Loads is currently BETA release.
Enable with the "`--beta`" command-line option.

If the response to multiple loads is desired, **Sierra/SD** provides a **beta** capability to evaluate multiple loads at each frequency. This provides a cost-effective alternative to using multiple solution cases. Its use is demonstrated in the following example,

```
SOLUTION
  DirectFrf
    load foo bar baz
END
```

Input 5.2. Multiple Direct FRF Loads

where each of `foo`, `bar`, and `baz` are valid `LOAD` sections (section 4.5.1). By default, each load will generate a separate output file, with names such as `mesh-case-load.exo` `mesh-case-load.frq` and `mesh-case-load.h`, where `mesh` is the root of the input mesh name (or database name – section 9.1.1), `case` is the case name (or "out" for unnamed single-case solutions), and `load` is the load name ("foo", "bar", or "baz").

Setting `concat_multi_rhs=true` in the `PARAMETERS` section (section 4.4) will force results from multiple loads to be output to the same result file. Note that in that case, the resulting output will include duplicate time-stamps (frequencies), one for each load.

5.6.2. *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. [2] The two additional parameters required for the expansion are listed in table 5-12. They are described below, and an example is shown in input 5.3. The theory is described in reference.[6]

```
SOLUTION
  case out
    DirectFrf
    Interpolate Points = 50
    Interpolate Order = 18
END
```

Input 5.3. 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.

5.7. Model_Check Solution Case

Parameter	Type	Default	Description

Table 5-13. – Model_Check Solution Case Parameters.

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

The keyword `model_check` will cause **Sierra/SD** to form matrices. No solve will be done. The main reason to use this solution case is to output diagnostics to help debug model setup. Outputs available include mass properties, matrix diagonals, metrics around element shape, constraint diagnostics, etc. Additionally, this case can be used to write out MATLAB format matrices with the 'mfile' option for custom debugging or post processing.

5.8. eigen Solution Case

Parameter	**Type**	**Default**	**Description**
nmodes	*Integer*	10	Number of modes to extract. See section 5.8.1
shift	*Real*	-1.0e6	Shift to apply to matrix system to allow solving singular systems. See section 5.8.2
untilfreq	*Real*	Inf	Target frequency to reach. See section 5.8.3
fluidloading	*yes/no*	no	Turns on added mass for approximate Wet Modes calculation.
use_residual_vectors	*File*	false	Activates residual vector calculations.
load	*LOAD ID*	N/A, no load used	Only valid when use_residual_vectors is `true`. Designates the load(s) to be used to calculate residual vectors.
resid-ual_vectors_node_list_file	*File*	N/A, no file read	Only valid when use_residual_vectors is `true`. Designates the nodelist file to be read from to calculate residual vectors.
resid-ual_vectors_solver_options	*String*	N/A, uses default solver options	Only valid when use_residual_vectors is `true`. This allows a user to designate a `SOLVER_OPTIONS` section specific to the residual vector calculation. If a `solver_options` parameter is used for the eigen solution, it will not be applied to the residual vectors calculation unless it is also specified with the resid-ual_vectors_solver_options parameter.

Table 5-14. – eigen Solution Case Parameters.

The **eigen** solution case computes eigenvalues and mode shapes of a system representing the natural vibration modes. The parameters **NegEigen**, **eig_tol**, and **eigen_norm** are described in sections 4.4 and 4.4

Eigenvectors, Φ, are stored in **Exodus** files as time dependent displacements. The time of an eigenvector is the frequency. `SEACAS` library tools expect times to be unique, occasionally causing benign warning messages when there are multiple modes at the same frequency. The mode shapes multiplied by the consistent mass matrix 9.7.38 are also available.

5.8.1. Option nmodes

Nmodes is the number of modes to compute. The eigenvalues are computed beginning with the lowest frequency mode and working up. The calculation continues until `nmodes` have converged. Modal analysis uses iterative Lanczos procedures. These methods build a Krylov subspace from which the solution is determined. The Krylov subspace method is designed to find the lowest frequencies, which is typically what is needed for structural analysis. More linear solves and longer run times are required to compute more modes and reach higher frequencies.

If `nmodes=all`, then all the finite modes of the structure are computed using dense linear algebra. No shift is required. The mass matrix may be either full rank or singular. Constraints may be present. This option is only available in serial. The number of dofs must be < 1000.

5.8.2. Solving Singular Systems with Shifts

There is a trade off between reducing the number of linear solves required to solve an eigenvalue problem or reducing the cost per linear solve. This trade off is controlled using the shift. The number of solves per eigenvalue decreases as the shift decreases in magnitude, and at the same time the cost per linear solve increases. Direct linear solvers require a shift for singular (floating) systems. Iterative methods are more reliable when applied to positive definite linear systems than they are when applied to positive semi-definite linear systems, The eigenvalue problem is defined as,

$$(K - \omega^2 M)\phi = 0. \tag{5.8.1}$$

K and M are positive semi-definite matrices. If the shift $\sigma < 0$, then $K - \sigma M$ is positive definite. Solution cases generally involve positive definite matrices except direct frequency response.

Here K and M are the stiffness and mass matrices respectively, and ω and ϕ are the eigenvalues 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 Krylov subspace is built by repeated solving equations of the form $Ku = b$. For floating structures, or structures with zero-energy 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 ensuring that no component of the null vectors ever occurs in b. This approach is equivalent to computing the pseudo inverse of K.

The geometric rigid body mode capability 5.30 can be used to analytically compute the standard rigid body modes and deflate them from a singular system. This is recommended for all floating structures.

Shifting. The second method involves solution of the shifted problem,

$$((K - \sigma M) - \mu M)\,\phi = 0. \tag{5.8.2}$$

This system has the same eigenvectors, ϕ, as the original equation, and its eigenvalues, μ, are related to the originals by $\mu = \omega^2 - \sigma$.

On serial platforms, a small negative shift is normally sufficient to solve the problem due to the high accuracy of serial direct solvers. For parallel solution, a reasonable shift value is usually given by

$$\sigma = -\omega_{elas}^2,$$

where ω_{elas} is the expected first nonzero (or elastic) eigenvalue. Because **Sierra/SD** cannot compute an optimal shift value a priori, a default of -1 is used and a warning is written if the shift used is well outside of the expected range. In practice, a shift of $-1e6$ is often used in **Sierra/SD** input, as this gives reasonable results for extracting frequencies anywhere near 150Hz, assuming the time unit is seconds. This is often the general frequency range of engineering interest. However, an adequate shift may vary some depending on the units and the properties of an analysis.

The shifted problem benefits from the fact that $K - \sigma M$ can be made non-singular (except in 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 the recommended value of $\sigma = -\omega_{elas}^2$.

If σ is too large, many solves will be required to determine the eigenvalues, which consequently slows convergence. If σ is too small each linear solve may be near singular, requiring many solver iterations or being unable to reach the target residual at all resulting in a 'SOLVER OUT OF BOUNDS' error.

Another consequence of an excessively large or small negative shift is that potentially not all redundant zero eigenvalues may be found. These modes may be found by correcting the shift, tightening tolerances, or by restarting.

The shifted eigenvalue problem is more reliable. Set the `grbm_tol` to a small value (e.g. 1e-20)(or use the default), 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 and no zero energy modes, the system is not singular and a shift is unnecessary.

Example

A representative `Solution` section for an eigendecomposition with a **shift** of -10^6 following. A dozen modes are requested. This shift would be appropriate for a system where the first elastic mode is approximately 150Hz.

```
SOLUTION
  eigen
  nmodes 12
  shift -1.0e6
END
```

5.8.3. UntilFreq Option and Modal Restart

The `untilfreq` keyword provides an additional method of controlling the eigenvalues 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 eigendecomposition. **Sierra/SD** restarts a maximum of **5** times.

Sierra/SD uses the **ARPACK** Lanczos solver for the eigen problem. This solver maintains the orthogonality of the eigenvectors for a single batch of modes. However, when restarted, the solver must deflate out the previously computed modes. There can thus be a slight loss of orthogonality due to round off. With repeated restarts, the effect can significantly reduce accuracy.

Additionally, modal restart can be manually controlled via the multicase solution. In the example below, the first 1000 modes are computed in 'eig1'. Next, `eig2` restarts and reads and deflates those 1000 modes from the system. In this solution case, an additional 500 modes are computed, for 1500 modes. Restarting a modal analysis in this way can compute modes with less memory than computing all modes at once. An excessive number restarts of this nature (more than about five) will lead to accuracy loss from round off errors during deflation.

```
SOLUTION
  case 'eig1'
    eigen
    nmodes 1000
  case 'eig2'
    eigen
    restart = read
    nmodes 1500
END
```

5.8.4. ModalFilter Option

The optional **ModalFilter** keyword provides a means of reducing the modes retained for output and for subsequent analysis. For more details, see Section 5.13.

Use of this parameter within the **eigen** solution case is *deprecated* in favor of the new **modalfiltercase** solution case (Section 5.13).

You can also put the modal filter into a separate case, called `preddam` 5.5 .

5.8.5. Fluid loading Option

Although a coupled structural/acoustic eigenvalue problem is quadratic, under certain approximations, the fluid may be represented as an added mass on the structure, and a real eigenvalue problem results as described in subsection Wet Modes or Added Mass section Solution Procedures of the Theory Manual. The `fluidloading` enables that added mass calculation. For wet modes all material blocks that use an acoustic material are treated alike as a fluid domain for mass loading.

In an early implementation the lowest modes of the acoustic stiffness matrix were used to approximate the added mass. As a sanity check, one mode of the acoustic stiffness matrix is approximated.

A fatal error is returned if infinite elements are detected. Fluid loading is not applicable to models that use infinite elements for two reasons. Infinite elements lead to a nonsymmetric eigenvalue problem that is less well-supported than other modal analyses. Second, the fluid region stiffness matrix is singular, implying an infinite added mass.

5.8.6. Rigid Body or Zero Energy Modes

Rigid body modes represent rigid body translation or rotation of a structure. A normal free-free structure would be expected to have 6 rigid body modes. A structure with constraints may have fewer than 6 rigid body modes. On the other hand a structure with multiple disconnected pieces may have more than 6 rigid body modes. To compute *any* rigid body modes, one must request all of them. For structures with 6 rigid body modes, `nmodes` ≥ 6. Otherwise, the solution case is likely to fail. Prior to computing the wet modes, a Geometric Rigid Body Modes solution case 5.30 is added and the parameter `num_rigid_mode` is set to 6.

5.8.7. Residual Vectors

residual vectors is currently BETA release.
Enable with the "`--beta`" command-line option.

The residual vectors capability utilizes modal truncation augmentation (MTA)[18] and provides a method to represent the modes not retained in the eigendecomposition. It is particularly useful to accurately capture the static response of a structure without many modes.

The residual vectors capability is activated by the `use_residual_vectors` option. An example `solution` block is given here for a case with six rigid body modes.

```
SOLUTION
  title 'Sample MTA solution procedure'
  case 'eigen'
    eigen
      nmodes=30
      shift=-1000.   // needed for floating
      solver=gdsw
      use_residual_vectors = true
      residual_vectors_node_list_file forcingNodes.txt
END
```

This would output a file with 30 modes plus a residual vector for each DOF of each node in `forcingNodes.txt`. This is likely more than is needed to get an accurate static response, but there is not currently a capability to create a residual vector using a more specific loading designation.

Alternatively, a user can use a set of static loads to create residual vectors resulting in far fewer residual vectors being formed while providing a similar accuracy in subsequent analyses for a known load environment. This can be done by replacing the `residual_vectors_node_list_file` with `load = LOADID`. Other load conditions can be added if requested.

5.8.7.1. Restart with Residual Vectors

Restart is enabled for eigen solutions that used residual vectors, however, extra caution should be exercised when using this capability to avoid bad behavior. The restart capability here is best used to add residual vectors to an Exodus file containing base modes from a previous run. The restart capability should **not** be used for:

- calculating additional base modes off of a file containing residual vectors
- adding additional residual vectors on top of previously generated residual vectors

For example, If a user wanted to add 2 residual vectors to a file with 10 base modes, they read a restart file and set nmodes to 10. Then, they will have a resulting file with 12 "modes" where the last 2 "modes" are residual vectors. If a user wanted to compute more residual vectors, they can add more loads, but still must keep nmodes set to 10. This will recompute the residual vectors and add new ones. Changing nmodes and restarting with the residual vector "modes" embedded in the restart file can result in bad behavior and should never be done. If more base modes are desirable, the residual vector "modes" must 1st be removed using a **SEACAS** tool like ejoin.

When using restart to feed into a subsequent analysis, there are two options. The 1st option uses the same format as the previous example where nmodes is set to 10, `use_residual_vectors` is enabled, and the loads are all fed in. This will recompute the residual vectors on each subsequent run, but will have the most robust error checking. Residual vector calculations are relatively inexpensive, so this is the recommended method. Alternatively, a user can select `restart = read` on eigen, set nmodes = 12, and turn residual vectors off. The code will not know that the last 2 modes are residual vectors, and will warn about bad relative residuals on the residual vector "modes". This is expected. In this case, take extra care to damp the residual vector "modes" to get best results. This is the only way to avoid recalculating residual vectors when using restart.

5.9. aeigen Solution Case

Parameter	**Type**	**Default**	**Description**
nmodes	*Integer*	10	Number of modes to extract. See section 5.8.1
shift	*Real*	-1.0e6	Shift to apply to matrix system to allow solving singular systems. See section 5.8.2
untilfreq	*Real*	Inf	Target frequency to reach. See section 5.8.3
implicit_restarts	*integer*	5	Number of allowed restart steps, default same as **eigen**
anblocksize	*integer*	1	Number of subspace vectors added each step, default same as **eigen**
anrho_shift	*Real*	0.0	Shift of infinite eigenvalues of constrained problem
anverbosity	*integer*	17	Level of verbosity of log files, see below for details
subspace_size	*integer*		Maximum number of subspace block Krylov-Schur basis vectors. By default, picks based on number of requested eigenvalues.

Table 5-15. – aeigen Solution Case Parameters.

The `AEigen` modal solution uses the more recent Krylov-Schur method in the Anasazi[7] package. Note that the eigenvalue methods of Section 5.8 use the closely related `ARPACK`[33] package. Modal solution methods all control accuracy using the parameter `eig_tol`. Shifts are used as discussed in Section 5.8.2.

Anasazi has support for GPU architectures. It also supports block linear solvers, although no true block linear solvers are available.

The amount of diagnostic information or level of verbosity is changed used `anverbosity` . The default value is 17, implying that the Anasazi solvers will output errors, warnings, and timing details. Each verbosity type is controlled by a single bit in the integer `anverbosity`. These types are listed in Table 5-16. 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.

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 5-16. – AEigen Verbosity Table.

Example Input 5.4 shows how to select the Krylov-Schur method. 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 eigenvalues equivalent to the example given in 5.8 for **eigen**. The verbosity level specifies that after computing the eigenvalues, the solver will print status information (number of iterations, current eigenvalues) and timing statistics.

```
SOLUTION
  aeigen
    nmodes 12
    shift -1.0e6
    anblocksize 1
    anverbosity 25
END
```

Input 5.4. Aeigen example

5.10. Largest_Ev Solution Case

Parameter	**Type**	**Default**	**Description**
residual norm relative threshold, between 0 and 1, try 0.01			
info	*Integer*	0	if positive (say 3), then write ARPACK debugging information to the standard output stream

Table 5-17. – Largest_Ev Solution Case Parameters.

The `Largest_Ev` analysis determines the largest eigenvalue of the system

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

The largest eigenvalue is typically mesh dependent. Users may need to increase the threshold from its default value to get a solution. The description of diagnostic information and the threshold [33] are part of the original ARPACK documentation.

5.11. Fatigue Solution Case

Parameter	**Type**	**Default**	**Description**
method	†	narrow-band	Fatigue calculation algorithm to use.
duration	<*real*>	1	Time duration, τ over which to integrate fatigue damage.

Table 5-18. – Fatigue Solution Case Parameters.

† The available fatigue algorithms are narrowband and Wirsching.

High Cycle Fatigue occurs after long periods of alternating stresses in the elastic range. The input parameters for fatigue-failure are shown in Tables 5-18 and 6-2. An example is shown in input 5.5. Fatigue analysis requires inputs in several sections.

1. The fatigue section in the **Solution** block must be immediately preceded by a modalranvib solution, with noSVD option (5.16). This defines the random stress moments needed for computation of the stress crossing rates.
2. The **Fatigue** section in the Solution block defines general fatigue parameters. Relevant parameters are outlined in Table 5-18.
3. The `material` section must include parameters necessary for computation of fatigue and damage. Sections without this input will not have a fatigue parameter output. Typical materials parameters required for fatigue analysis are found in Table 6-2, found in the materials section, 6.5.2.

```
SOLUTION
  title 'High cycle fatigue example'
  case modes
    eigen
    nmodes 10
  case vrms
    modalranvib
    noSVD
    lfcutoff 10
  case failure
    fatigue
    duration = 1.0
    method = Wirsching
END

BLOCK 100
  material aisi4140
END

BLOCK 200
  material steel
END

MATERIAL aisi4140
  E = 10e6
  density = 0.00075
  Fatigue_A1 = 31.5805
  Fatigue_A2 = -14.0845
  Fatigue_Stress_Scale = 0.001 // Psi to Ksi
  MaterialType = PeakStress // or StressRange
END

MATERIAL steel
  E = 30E6
  nu = 0.28
  density = 0.007
END
```

Input 5.5. High Cycle Fatigue Input Example. Fatigue is specified by block. In this example, there are two blocks. The material properties for block `aisi4140` include fatigue parameters, but the material properties for block `steel` do not include fatigue parameters. The fatigue calculation is performed for block 100.

5.11.1. User Output

If the **Fatigue** method is specified in the Solution section then output for fatigue will be provided. There is no need for a specification in the **outputs** section. The following fields will be output.

NarrowBandDamageRate: There are two possible equations for D_{NB} depending on if the material parameters specified for the S-N curve are based on a stress range or peak stress The Narrow Band damage per unit time, defined as:

$$\dot{D_{NB}} = \frac{\nu_o^+}{A}(\sqrt{2}\sigma_s F_{SS})^m \Gamma\left(\frac{m}{2}+1\right) \tag{5.11.1}$$

for materials defined based on Peak Stress, and

$$\dot{D_{NB}} = \frac{\nu_o^+}{A}(2\sqrt{2}\sigma_s F_{SS})^m \Gamma\left(\frac{m}{2}+1\right) \tag{5.11.2}$$

for materials defined base on the Stress Range. Equation (5.11.1) is used by default. A stress range type material can be specified in the input (MaterialType = StressRange). See the Theory Manual for details of the parameters.

WirschingDamageRate: The Wirsching damage per unit time. This may be thought of as a scaled value of the NB damage intensity.

$$\dot{D}_W = \lambda \dot{D}_{NB}.$$

ZeroCrossingRate: the expected positive zero-crossings intensity.

$$\nu_o^+ = \sqrt{\frac{M_2}{M_o}}$$

Damage: The selected damage rate multiplied by time.

$$D = \tau \dot{D} \tag{5.11.3}$$

Here $\dot{D}$ can be either the Narrow Band or Wirsching Damage Rate.

PeakFrequency: the expected peak occurrence frequency.

$$\nu_p = \sqrt{\frac{M_4}{M_2}}$$

The even moments, M_x (with x=0,2,4), are output as part of the random vibration computation, see (5.16). For example, $M2 = (\text{VRMS2}/2\pi)^2$.

5.12. buckling Solution Case

Parameter	**Type**	**Default**	**Description**
nmodes	*Integer*	10	Number of modes to extract. See section 5.8.1
shift	*Real*	-1.0e6	Shift to apply to matrix system to allow solving singular systems. See section 5.8.2
untilfreq	*Real*	Inf	Target frequency to reach. See section 5.8.3
bucklingSolver	†	ARPACK	Eigenvalue computation algorithm method to use.

Table 5-19. – buckling Solution Case Parameters.

† The available algorithms are ARPACK, Anasazi and ARPACK_Regular_Inverse.

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 tabulated below. Users are encouraged to review the discussion of buckling analyses in.[44]

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). Determining an effective shift is problem dependent, but no `shift` is needed when using the `ARPACK_Regular_Inverse` buckling solver option, and any provided `shift` value will be ignored in that case.

Although the `ARPACK_Regular_Inverse` buckling solver is not the default, it is much easier to use, and its results are more trustworthy than the default `ARPACK` option. Currently, all buckling tests in our test suite pass with either `ARPACK` or `ARPACK_Regular_Inverse`, with comparable run times and solution accuracy.

The `nmodes` parameter specifies the number of requested buckling modes. Its default value is 10.

Unlike ordinary modal analysis, 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
```

The lowest obtained eigenvalue is 100.0. The critical buckling pressure would be $P_{cr} = 100.0 \times 10.0 = 1000.0$. 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 of floating structures is not supported at this time. If global rigid body modes are present, the solution may not be correct.

Example A `Solution` section for buckling analysis with a **shift** of -10^6 looks like the following, if 1 mode is needed (i.e. if the use is confident that the modes are well separated).

```
SOLUTION
  buckling
  nmodes 1
  shift -1.0e6
END
```

5.13. ModalFilter Solution Case

Parameter	**Type**	**Default**	**Description**
ModalFilter	*string*	none	Modal filter to define modes to retain. See section 5.8.4
write_files	*all*| *none*| *exodus*| *history*	all	Controls which result files are written during this solution.

Table 5-20. – ModalFilter Solution Case Parameters.

A ModalFilter filters modes computed by an **eigen** solution case (Section 5.8). The filtered modes are then used by subsequent solution cases.

By default, the filtered modes are not output since the preceding **eigen** case outputs the all the mode shapes. To also output the filtered modes, add **write_files** to the solution case. Write files is also used in the section 5.18 solution case. You also need to specify one more thing about which files to output. Write files can be set to either all, exodus, frequency, history or none.

Controlling the modes retained for subsequent analyses can significantly reduce run time with little effect on the desired response. For example, a shell structure may have many hundreds of modes contributing to the normal mode response, and only a few that interact with the loads. [1]

If computing eigenvalues, e.g., for CBR, 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 input 5.6.

```
SOLUTION
  case eig
    eigen
    nmodes=500
  case filter
    modalfiltercase
    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
```

Input 5.6. Example ModalFilter Input

For this example, the following actions are performed in the filter.

1. The first 999 modes are removed. In this case 500 modes are computed, and all modes are removed.
2. The modes contributing the most to a cumulative modal effective mass are added. Modes sufficient for 80% contributions to the x, y, and z directions are added. Modes needed to achieve 20% 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. Modes 99, 100, 101 and 103 are added if they are not already included.

[1] The modes of large ship are an example. Only a few of the modes contribute to global bending or torsional modes. The remaining modes are local, and may not be of interest to the analysis.

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 4.1 for more details. Valid action keywords are listed in Table 5-21.

Keyword	Application Space
remove	integer list or “all”
add	integer list or “all”
cumulative mef	6 fractions
cumulative nmef	6 fractions

Table 5-21. – 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 (union) 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.

The Modal Effective Mass equals the Modal Participation Factor. Although the Modal Participation Factor is defined in the next section, the Normalized Modal Effective Mass is undefined.

5.14. Modal Participation Factor Solution Case

Parameter	**Type**	**Default**	**Description**
write_table	*Yes|No*	Yes	If yes write Γ table, if no write summary table
blockwise	*Yes|No*	Yes	If yes write blockwise summary to result table
RCID	*string*	123	A string representing rigid body modes to include in the calculation. 123 resents the translational degrees of freedom. 123456 includes all six degrees of freedom.

Table 5-22. – Modal Participation Factor Solution Case Parameters.

A *Modal Participation Factor* (MPF) is a quality of a mode shape. Another name for the *Modal Participation Factor* is the *Modal Effective Mass*. A MPF is a direction cosine of an eigenvector along one of the 6 rigid body modes. It measures the interaction of the modes with a gravity load or a base excitation.

The rigid body modes $\{R_i\}_{i=1}^{6}$ ignore any boundary conditions.

The modal participation factor of an eigenvector v of the constrained system is determined from the representation of v in the unconstrained space, using a lumped mass matrix 4.6.4,

$$\Gamma_{ij} = \frac{R_i^T M v_j}{\sqrt{(R_i^T M R_i)\,(v_j^T M v_j)}}. \tag{5.14.1}$$

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.

$$\text{MPF}_j = \sum_i^6 \Gamma_{ij}^2 \tag{5.14.2}$$

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 5.8). 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. [2] Then,

[2]The overall modal contribution is not the sum of the block wise contributions, and contributions from individual blocks may cancel other blocks. See Table 5-23.

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

Table 5-23. – MPF Summary data. Each mode, v_j, has contributions from each summary value.

$$\Gamma_{ij}^k = \frac{R_i^T M^k v_j}{\sqrt{(R_i^T M R_i)\,(v_j^T M v_j)}} \tag{5.14.3}$$

Options for the `MPF` method are listed in Table 5-22.

Summary data from the calculation is written to the results file as described in Table 5-23. 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 5.14.3. An example is provided in input 5.7.

The external text file is intended to be easily read by external programs such as the MATLAB "load" command. It therefore 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.

The optional external text file with file name extension mpf contains block-wise modal participation factors. The data is presented in tabular format, separated by white space. Data analysis software tools support 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 deck). 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.

```
SOLUTION
  case eig
    eigen
    nmodes=10
    shift=-1e5
  case out
```

```
    mpf
    blockwise=yes
    RCID=123
    write_table=yes
END
```

Input 5.7. Modal Participation Factor (MPF) Example

Modal Effective Mass

There are several definitions of the modal participation factor. The value from equation 5.14.1 is unit normalized such that the sum of the squares of Γ_{ij} over all modes equals unity. A related term is the modal effective mass.

$$\mathrm{MEFF}_{ij} = \sqrt{M_{oi}} \cdot \Gamma_{ij} \tag{5.14.4}$$

Here M_{oi} is total mass associated with each rigid body mode. The sum of the squares of MEFF_{ij} over all modes, j, for a given rigid body mode, i is the total mass associated with that RBM. Modal truncation decreases the sum. The `modal effective mass` table is output to the result file immediately after the modal participation factors. At the bottom of the table, the sum of the squares of the modal effective mass is given for each rigid body mode. The total mass and moments of inertia of the system are also given.

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 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 4.6.4).

5.15. ModalFrf Solution Case

Parameter	**Type**	**Default**	**Description**
complex	*True*|*False*		If true require use of a complex eigen solution. If false require use of a real valued eigen solution. If unset use the existing eigen solution.
lfcutoff	*Real*	-Inf	Exclude any modes below this frequency from the modal computation. Often used to exclude rigid body modes.
usemodalaccel			If set, use modal acceleration method
nrbms	*Integer*	0	Number of rigid body modes, needed for usemodalaccel
write_files	*all*| *none*| *exodus*| *frequency*	all	Controls which result files are written during this solution.

Table 5-24. – ModalFrf Solution Case Parameters.

Option **ModalFrf** is used to perform a modal superposition-based frequency response analysis. In other words, **ModalFrf** provides an approximate solution to the Fourier transform of the equations of motion. If $\overline{u}$ is the Fourier transform of the displacement, u, and $\overline{f}$ is the Fourier transform of the applied force, then

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

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

Modal damping can be applied with either real eigenvalues computed by **eigen** or complex eigenvalues shapes computed by **QEVP**. However, proportional damping is currently available only with real modes. For more details on damping, see Section 6.7.

5.15.1. 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, 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, but in the case of the modal acceleration method, a static correction term partially accounts for the truncated high frequency terms. Thus, that method is generally more accurate than the modal displacement method. The most accurate method, though also the most computationally expensive, is `DirectFrf` 5.6.

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 mode frequency and associated fractional modal damping respectively. In the case of complex modes, the equations need to be linearized as described in subsection Quadratic Modal Superposition section Solution Procedures of the Theory Manual.

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 deck. Also, see subsection Modal Frequency Response Methods section Solution Procedures of the Theory Manual.

The modal acceleration method more accurately computes the poles (or peaks) of the response and compares zeros of a function. The cost is an additional factor and solve. The method can be used on floating structures, but the presence of rigid body modes makes the solution much more difficult than eigen analysis. While a negative shift is recommended when computing eigenvalues of a floating structure to remove the singularity due to RBMs, no such approach is possible when using the modal acceleration method. Thus, accurately determining the global RBMs may require significant solver adjustments when using 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 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 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 with the application region (see Section 9.3). 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 not depend on the magnitude of `freq_step`. This parameter controls the quantity of output. Examples are shown in inputs 5.8, 5.9 and 5.13.

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 **outputs** section. If nothing is specified in the **outputs** 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
```

Input 5.8. ModalFrf Example Input

5.15.2. ModalFrf with Complex Modes

In the case when complex modes are used, 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 5.19.1. 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 does not support 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. Eigenvectors of repeated modes are not orthogonal. For more details, we refer to Section 5.19.1.

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

```
SOLUTION
  case eigenvalue
    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
```

Input 5.9. Complex ModalFrf Example Input

5.16. modalranvib Solution Case

Parameter	Type	Default	Description
noSVD			von Mises stress computation method, noSVD is less expensive, and provides additional stress moments.
lfcutoff	*Real*	0.1	Exclude any modes below this frequency from the modal computation. Often used to exclude rigid body modes.
truncationMethod	*none|displacement|acceleration*	none	Truncates modes with low activity
keepModes	*integer*	Inf	Keep the specified number of modes, kept modes are selected based on the highest modal activity.
checkSMatrix	*true|false*	true	†

Table 5-25. – modalranvib Solution Case Parameters.

† If checkSMatrix is true, then at each frequency, the symmetric positive semi-definite correlation matrix S is computed, and tested for positiveness. An indefinite correlation matrix indicates that fundamental error has occurred. If PSD output is requested, then matrix evaluations are enabled.

Option **modalranvib** is used to perform a modal-superposition-based random vibration analysis in the frequency domain. Root-mean-square (RMS) outputs, including von Mises stress, are computed for a given input random force function. The resulting power spectral density functions may also be output at locations specified in the `frequency` section. The forcing functions (one for each input) must be explicitly specified in the `ranloads` section (8.3.22). It must reference a **matrix-function** definition (see Section 4.10.17).

modalranvib should be used in a multicase solution after an **eigen** solution.

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 a manner suggested in [68] and discussed in [69] to estimate fatigue life in broad-band random excitation. Also, see the **Fatigue** section (5.11). 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`. It is possible to not truncate at all (`none`), or to truncate based on accelerations (`acceleration`). Acceleration contributions are 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 **keepmodes** remain. Note that this procedure is much different from truncating the higher-frequency modes. Modal truncation is important because all the operations compute responses that require $O(N^2)$ operations. Even if **keepmodes** is not entered, modes with modal activity less than 1 millionth of the highest active mode will be truncated.

The 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 9.3). 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. Examples are presented in inputs 5.8, 5.9 and 5.13.

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[52,41] 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, `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 trickier than most input. A number of blocks must be specified.

1. The `Solution` block requires the input for **eigen** analysis, and the keyword **modalranvib**.
2. The **RanLoads** block contains a definition of the spectral loading input matrix and other input. Note that the input, S_{FF} is separated into frequency and spatial components. The spatial component is specified here using **load** keywords. See Section 8.3.22. 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 (4.10.17).

4. There must be a **function** definition for each referenced spectral function. Functions of time or frequency are further described in Section 4.10.

5. There must be a `frequency` block that is used for integration and optionally also for output of displacement and acceleration output. See Section 9.3.

6. As an undamped system is singular, some type of **damping** is required. Modal damping terms are required. [3] See Section 6.7.

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. [4] The keywords **rotational_displacement** and **rotational_acceleration** in the `outputs` and `frequency` sections will output the corresponding RMS and PSD quantities respectively. Quantities output are listed in Table 5-26.

All other input should remain unchanged.

[3]Proportional damping, such as is applied with the `alpha` and `beta` terms, will *not* work in **modalranvib**.

[4]These stresses are linear functions of the displacement.

Keyword	Output Variable	Description
Vrms	vrms	Root Mean Squared von Mises Stress
	D1...D5	von Mises Stress SVD moments. Details in.[53]
	M0, M2, M4	von Mises Stress moments. noSVD
	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
rotational_displacement	RotXrms	Rotational X component of RMS displacement
	RotYrms	Rotational Y component of RMS displacement
	RotZrms	Rotational Z component of RMS displacement
rotational_acceleration	RotAxrms	Rotational X component of RMS acceleration
	RotAyrms	Rotational Y component of RMS acceleration
	RotAzrms	Rotational Z component of RMS acceleration

Table 5-26. – ModalRanVib Output to Exodus File. The stress spectral moments are neither computed nor output if `noSVD` is selected. The stress moments are available if `noSVD` is selected, and may be used for fatigue. The RMS values of displacements and acceleration are components of a Hermitian tensor. See Section 5.16.1 for details.

5.16.1. *Power spectral density*

When requested in the frequency block, 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. [5]

Like the input cross spectral forces, the output quantities are Hermitian, with 9 independent translational quantities per frequency, at each output node for each type of output. The method for transforming these quantities in alternate coordinate systems are in subsection Modal Frequency Response Methods section Solution Procedures of the Theory Manual. The rotational terms can be requested using the keywords **rotational_displacement** or **rotational_acceleration**. Note that the cross correlation terms for rotation are not output.

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

[5]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 (100 to 1000 times 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.

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

H_{lj} is the transfer function giving a_j/f_l, and S_{kl} is a power spectral density input. It has units

$$\text{force}/\text{Hz}.$$

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

$$G_{ij} = H^{\dagger}_{ki} \langle m_k a_k, a_k m_k \rangle H_{lj} \tag{5.16.1}$$

$$= (m_k^2) H_{ki} \langle a_k, a_k \rangle H_{kj} \tag{5.16.2}$$

Thus, the acceleration PSD must be multiplied by the square of the mass to get the force PSD. **Sierra/SD** applies the scale factor to the spatial force distribution (which is also squared), so the scale factor in **Sierra/SD** should be m_k.

5.17. modalshock Solution Case

Parameter	Type	Default	Description
srs_damp	*Real*	0.03	Damping coefficient used for the shock response spectra calculation

Table 5-27. – modalshock Solution Case Parameters.

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. More information about the about shock response spectra solution cases is given in Section 5.27 describing implicit-transient-based SRS.

A `frequency` block must also be included in the input deck for **modalshock** solution cases to define the frequencies and nodesets for computing the shock response spectra (see Section 9.3). 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 9.3). 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 controls the quantity of output. Examples are presented in inputs 5.8, 5.9 and 5.13.

5.18. modaltransient Solution Case

Parameter	Type	Default	Description
time_step	*Real*		Time step size
nsteps	*Integer*		Number of time steps to take
start_time	*Real*	0.0	Solution case start time
nskip	*Integer*	1	Results output frequency
rho	*Real*	1	Select time integrator
load	*Integer*		Load block to apply during solution case. See Section 4.5
write_files	*all*| *none*| *exodus*| *history*	all	Controls which result files are written during this solution.
lfcutoff	*Real*	-Inf	Exclude any modes below this frequency from the modal computation. Often used to exclude rigid body modes.
flush	*integer*	50	Defines how often results are written to the exodus results file. See Section 4.6.1
handoffStaticPreload	*bool*	false	Begin modaltransient solution from preloaded static state

Table 5-28. – modaltransient Solution Case Parameters.

Option **modaltransient** is used to perform a modal-superposition-based implicit transient analysis. Damping for the model is defined in Section 6.7.

The parallel solution of modal transient may be slower than expected. Only those DOFs requested for output are included in the solution; thus, requesting output on only a small subset of the model can improve solution speed.

If output is required at few locations, it is recommended to set `write_files = history`, or specify an empty **outputs** definition. An empty **outputs** section applies to all solution cases, whereas the **write_files** keyword applies only to the current solution case.

Forces and displacements on modal degrees of freedom are also available via the keyword **modalvars** in the **echo** definition.

modaltransient supports restart in the **eigen** part of the analysis, the **modaltransient** part, or both. In the latter case, two things would happen first. Any modes from the modal restart file are read, and the time history data from any previous transient restart files (direct or modal) is read. Afterwards, stepping in time continues.

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

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

and, subsequently, the **eigen** restart and **modaltransient** would be:

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

The previous **modaltransient** cannot restart from the last computed time step when computing additional time steps. To restart it is necessary to add option `write` in the **modaltransient** case. 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
```

5.19. QEVP Solution Case

Parameter	Type	Default	Description

Table 5-29. – QEVP Solution Case Parameters.

Note: Options for the **QEVP** solution case depend on the algorithm used and are documented below.

5.19.1. Quadratic Eigenvalue Methods Comparison

The quadratic eigenvalue problem is defined as,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{5.19.1}$$

The solution of the quadratic eigenvalue problem (5.19.1), 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 5-30 lists recommended procedures for different problem sets.

Anasazi: It is possible to use the Anasazi method, although more testing is needed. It 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 non-singular. 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 5-15. 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 the modes, and then compare a relative residual to `eig_tol`. If the residuals are large, the modes are not returned to the user. It can be helpful to use a larger value of `eig_tol` than the default, say 10^{-8}. In this situation, it is helpful to set `anverbosity` to a large value, say And then examine other diagnostic information, including the data written to the screen, to ascertain the accuracy of the modes.

Ceigen: The **ceigen** method uses methods in ARPACK, and solves the quadratic eigenvalue problem. It is the method of choice for challenging problems.

SA_EIGEN: The **sa_eigen** method solves a coupled structural acoustics problem by solving a linear, uncoupled eigenvalue problem 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 applicable to structural/acoustic systems. It is robust. Modal truncation can introduce significant errors. Some solutions can fail (or convergence may be 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 the eigenvalue problem,

$$Ku = \lambda Mu. \tag{5.19.2}$$

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

$$Ku + \lambda Cu + \lambda^2 Mu = 0 \tag{5.19.3}$$

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

Use the keyword `method` followed by the name of the method (Anasazi, ceigen, sa_eigen, Projection_eigen) to select a **QEVP** method. Below is a more detailed description of each **QEVP** method, their parameters, and examples of how to use them.

Problem	Ceigen	SA_eigen	Anasazi	Projection_eigen
Damped Systems	Acceptable	Acceptable	Fails	Acceptable
structural acoustics	Fails	Acceptable	Acceptable	Acceptable
Rotational systems	N/A	N/A	Acceptable	Acceptable
Damped structural/acoustics	Fails	Acceptable	Fails	Acceptable

Table 5-30. – A 2005 comparison of quadratic eigenvalue problem methods. Although Ceigen and Anasazi have changed substantially since 2005, this table has not been updated to reflect those changes.

5.19.2. Anasazi

The **Sierra/SD** interface to the Trilinos package Anasazi solves the quadratic eigenvalue problem defined as

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{5.19.4}$$

See Section 5.19.1 for a comparison of these methods for this problem. As currently implemented, the Anasazi method applies to systems with a non-singular mass and stiffness matrix. The damping matrix may be asymmetric. Options for input are described in Table 5-31. An example is given below.

Table 5-31. – Options for **QEVP** Anasazi Solutions.

Option	**Argument**	**Default**	**Comment**
nmodes	*Integer*	10	number of modes
shift	*Real*	0	ignored
reorthogonalize	Yes/No/Full	"Yes"	Reorthogonalize vectors
check_diag	Yes/No/Full	"Yes"	Check that vectors diagonalize linearized system
ANverbosity	*Integer*	17	Anasazi verbosity
ANblocksize	*Integer*	1	Anasazi Block Size

Although modal truncation methods are an industry standard, the results sometimes are inaccurate. A solution may not to change when the number of modes increases, and still be inaccurate. Verification is entirely up the user in every case with modal truncation methods.

In addition to the shift, users must select **eig_tol**. But discerning an effective value depends on the problem, especially on the shift and the linear solver accuracy. Too large a threshold ($> 1.e-4$) degrades solution accuracy. Too small a threshold ($< 1.e-13$) leads to divergence. However, with these methods, diagnostic information is provided (written to the standard output stream) to guide users with problem configuration and solution verification.

```
SOLUTION
  case eigenvalue
    QEVP
    method=Anasazi
    nmodes=14
    anverbosity=27
END
```

5.19.3. Damped Eigenvalue Problems

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,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{5.19.5}$$

Note that two other solution methods may also be used to evaluate the quadratic eigenvalue problem. Each method has its strengths and weaknesses. A comparison of these methods is provided in Section 5.19.1.

The following table gives the parameters needed for complex (non-Hermitian) eigenvalue problems. Additionally, the eigenvalue tolerance can be set with the **eig_tol** parameter in the parameters block (Section 4.4).

Parameter	**Type**	**Default**	**Description**
nmodes	*Integer*	100	Number of modes to compute, reported as complex conjugate pairs
viscofreq	*Real*	1e-6	Frequency at which to evaluate material damping

Table 5-32. – ceigen Solution Case Parameters.

The optional **viscofreq** keyword indicates the frequency at which the damping properties of viscoelastic materials will be computed. It must be non-negative. The **viscofreq** parameter can be confusing. In particular, viscoelastic 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 the viscoelastic 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 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 viscoelastic materials generate a more complex equation than that shown in equation 5.19.5. With a single term in the Prony series, the equation of motion for a damped viscoelastic structure can be written in the frequency domain.

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

Table 5-33. – Ceigen Tests.

Name	**Description**
ceigen	stiffness proportional damping
ceigen_visco	viscoelastic damping
ceigen_dash	dashpot damping
steel_in_foam	complex mixed materials

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

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

Eigendecomposition of damped models is more difficult, and much less mature than ordinary eigenvalue problems. 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. As usual, one must request displacement output in the `output` section (see 9.7.1). The output file contains 12 fields (six real and six imaginary). Few post-processing tools handle complex mode shapes. Also, see subsection Complex Eigen Analysis — Modal Analysis of Damped Structures section Solution Procedures of the Theory Manual.

Because of the challenges of solving complex eigenvalue problems, 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 5-33.

5.19.4. SA_eigen

The **QEVP** procedure with method SA_eigen provides a means of computing the modal response of very lightly coupled structural acoustic system, using a modal truncation basis. For the more tightly coupled aerospace vibration suppression problems only the CEigen and Anasazi methods have worked. The quadratic eigenvalue problem describing this system can be written as follows.

$$\left(\begin{bmatrix} K_s & 0 \\ 0 & K_a \end{bmatrix} + \lambda \begin{bmatrix} C_s & L \\ -\rho_a L^T & C_a \end{bmatrix} + \lambda^2 \begin{bmatrix} M_s & 0 \\ 0 & M_a \end{bmatrix}\right) \begin{bmatrix} \phi_s \\ \phi_a \end{bmatrix} = 0 \qquad (5.19.6)$$

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, $\phi_a = \nabla \dot{u}_a$. It helps to understand[45] the capabilities and limitations of this analysis.

Option	Args	Description
nmodes	*int*	Number of requested eigenvalues
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 5.8.
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:** Multicase requires 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 **full:** check all modes
check_diag	*string*	**no:** no check for orthogonalization **yes:** check redundant modes **full:** check all modes

Table 5-34. – SA_Eigen Options.

The **sa_eigen** method solves this system by solving for the uncoupled eigenvalues 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 5.3) to generate a dense system of equations that are solved and results propagated to the physical space. More details are available in the Theory Manual.

Options of the analysis are provided in Table 5-34, and an example is provided in input 5.10. 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 complex, as for the **ceigen** case (5.19.3).

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

Input 5.10. SA_Eigen Example

Limitations: This is a modal superposition method. The **Sierra/SD** interface to the Trilinos package Anasazi is a more complete but less robust method which does not rely on modal truncation. The SA_eigen method is accurate for many 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 5-35 summarizes the status of this procedure.

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

Analytic Reference	Verification Section	Tested	Parallel Test	User Test
32	46	Y	Y	some

Table 5-35. – Verification Summary for SA_Eigen.

Specialized Output: There are a few items that are output specifically for the *sa_eigen* procedures that can be 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 eigenvector computed on the reduced space. We divide ϕ into its structural and acoustic components. i.e.,

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

We compute,

$$F_{structure} = \frac{\phi_s^\dagger \cdot \phi_s}{\phi_s^\dagger \cdot \phi_s + \phi_a^\dagger \cdot \phi_a} \tag{5.19.7}$$

where $\phi^\dagger$ represents the complex conjugate transpose of ϕ. Note that these products are computed in the reduced space which has coordinates associated with each structural or acoustic eigenvalue. 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. 5.19.7) applied the acoustic domain. It represents the portion of the system level complex eigenvalue that is associated with the acoustic domain.

ErrorNorm We define a normalized modal energy residual.

$$E^n_{resid} = \frac{|\phi^\dagger(k + \lambda c + \lambda^2 M)\phi|}{\phi^\dagger K \phi} \tag{5.19.8}$$

Here 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.

5.19.5. Projection_eigen

Of all the solvers available for quadratic eigenvalue problems, only the *Projection_Eigen* method is guaranteed to always return a solution. On the other hand, no information is provided to the user to assess the accuracy of this approximation.

Options of the *Projection_Eigen* solver are provided in Table 5-36. These parameters are identical to those for the *sa_eigen* method.

Option	Args	Description
nmodes	*int*	Number of requested eigenvalues
shift	*Real*	Eigen shift used in computation of the subregion modes. See 5.8.
reorthogonalize	*string*	**no:** no reorthogonalization **yes:** reorthogonalize all modes **full:** check all modes
check_diag	*string*	**no:** no check for orthogonalization **yes:** check redundant modes **full:** check all modes
sort method	*string*	**magnitude:** by complex magnitude of λ **frequency:** by frequency and then damping. **damping:** by damping and then frequency. **truefreq:** by frequency, avoiding zero energy round off. **none:**

Table 5-36. – Projection_Eigen Options.

5.20. NlStatics Solution Case

Parameter	Type	Default	Description
tolerance	*Real*	1e-6	Controls completion of the Newton iteration.
max_newton_iterations	*Integer*	100	If the iteration count exceeds this value before reaching **tolerance**, the Newton loop is considered to have failed.
update_tangent	*Integer*	101	How often the tangent stiffness matrix is rebuilt during the Newton iterations.
num_newton_load_steps	*Integer*	1	Number of load steps used to incrementally step up to the final equilibrium position

Table 5-37. – NlStatics Solution Case Parameters.

If stiffness matrix K is a function of the displacement u in

$$K(u) = f,$$

then use **NlStatics** for nonlinear statics. Newton's method applied to the residual force equations to drive the residual $r = p - f$ to zero. The residual vector r is the difference between the internal force vector p and the external force vector f. 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** keyword provides control over the completion of the Newton iteration. Once the change in the L^2-norm of the *displacement* decreases below **tolerance**, the loop completes successfully.

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. 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 nonlinear (difficult) problems, this option may be optimal, but for most problems the extra cost of assembling a preconditioning the tangent stiffness matrix should be amortized over several solves. Note that for this option to improve Newton's method, the element types in the model must 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
```

5.21. NlTransient Solution Case

Parameter	Type	Default	Description
time_step	*Real*		Time step size
nsteps	*Integer*		Number of time steps to take
start_time	*Real*	0.0	Solution case start time
nskip	*Integer*	1	Results output frequency
rho	*Real*	1	Select time integrator
load	*Integer*		Load block to apply during solution case. See Section 4.5
write_files	*all| none| exodus| history*	all	Controls which result files are written during this solution.
tolerance	*Real*	1e-6	Controls completion of the Newton iteration.
max_newton_iterations	*Integer*	100	If the iteration count exceeds this value before reaching **tolerance**, the Newton loop is considered to have failed.
update_tangent	*Integer*	101	How often the tangent stiffness matrix is rebuilt during the Newton iterations.
flush	*integer*	50	Defines how often results are written to the exodus results file. See Section 4.6.1

Table 5-38. – NlTransient Solution Case Parameters.

The **NlTransient** solution method is used to perform a[28] direct implicit nonlinear transient analysis. A projector-corrector step is used. Note that for a linear system, the `NlTransient` analysis will require two solves per time step. Nonlinearity can be controlled using the parameter `nonlinear_default` 4.4

Stiffness proportional damping is silently ignored 6.7.1.

A successful nonlinear transient simulation sometimes requires fine tuning. The most important diagnostic parameter is to add NLresiduals to the input deck echo section. . The discussion of this topic resume below, in the discussion of `update_tangent`.

Also note that although some options for `NlTransient` have the same name as similar options for `NlStatics` (5.20), there are subtle differences in what the options do.

The `tolerance` applies to the Newton iteration at each step. Once the change in the L^2-norm of the *acceleration* decreases below **tolerance**, the loop completes successfully. Note the difference viz a viz **NlStatics** (5.20), where **tolerance** instead refers to the *displacement*.

If the iteration count in a given time step exceeds **max_newton_iterations**, the Newton loop is considered to have failed. The limit on the number of iterations per time step may be set using **max_newton_iterations**. The keyword has a different meaning in **NlStatics** (5.20), where **max_newton_iterations** refers to the total number of Newton iterations, instead of the number of iterations per time step.

In a **NlStatics** (5.20) 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 **NlTransient** analysis, load stepping makes no sense because the dynamic response of a structure subjected to a total load is different from the response 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 **NlTransient** problems, if Newton's method diverges, either the tangent stiffness matrix has to be updated more often (see **update_tangent**) or the time-step should be decreased.

Option **update_tangent** controls how often the dynamic tangent stiffness matrix is rebuilt during the Newton iterations, with default 101. The tangent matrix will not be updated at all by default unless a given Newton loop takes more than 101 iterations. 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 non-linear problems, some control of this option is recommended. Note, for this option to improve Newton's method, the element types in the model must have the dynamic tangent stiffness method implemented.

5.22. Random Vibration Solution Case

Parameter	**Type**	**Default**	**Description**

Table 5-39. – Random Vibration Solution Case Parameters.

See (5.16).

5.23. receive_sierra_data Solution Case

Parameter	Type	Default	Description
include_internal_force	*off|on*	on	Include or exclude computation of initial internal force
no_geom_stiff			Turn off the geometric stiffness term

Table 5-40. – receive_sierra_data Solution Case Parameters.

The solution case `receive_sierra_data` is used to input a deformed or preloaded model state. Calculations in SIERRA codes such as **Sierra/SM** may be input to **Sierra/SD**. The intended application is to compute large strain nonlinear responses in a separate code, followed by a supported solution case in **Sierra/SD**.

Primary example use cases are as follows.

(a) Preload from **Sierra/SM** results output, where displacements and stresses are provided, **Sierra/SD** reads those, adjusts the tangent stiffness matrix, and computes modes.

(b) Preload from **Sierra/SM**, where displacements and stresses are passed, **Sierra/SD** reads those, adjusts the tangent stiffness matrix and equilibration forces, and then computes a direct transient response to a user specified load.

(c) Implicit or explicit transient analysis in **Sierra/SM**, followed by a hand-off to an implicit direct transient analysis in **Sierra/SD**. By default, **Sierra/SD** starts at the end time of the **Sierra/SM** hand-off analysis.

In all cases, in addition to adjusting the stiffness matrix based upon the deformed model configuration from **Sierra/SM**, the mass matrix is recomputed. If the deformation of the solid is significant, as in the case of a highly-compressed foam, changes in the density of the material due to the deformation should be considered. The hand-off from **Sierra/SM** to **Sierra/SD** conserves mass as long as **Sierra/SM** outputs element-by-element deformed material density, and **Sierra/SD** inputs this density. Consult section 5.23.2 for details on variables for hand-off.

Solution method `receive_sierra_data` helps with the following.

1. Update the initial geometry from the previously computed displacements.
2. Update the element stiffness matrices due to preload stresses. This *geometric stiffness* correction is supported for volumetric, beam, shell, and membrane elements at this time. The geometric stiffness calculation may be disabled with the `no_geom_stiff` keyword or the blockwise parameter `geom_stiff` (table 6-5).

3. Compute and apply an initial force associated with the input stress state. Disable this by setting `include_internal_force=off` command.
4. Update tangent stiffness of materials based the preloaded material state.

`receive_sierra_data` solutions require a multi-case solution. An example follows. In this example preload data is received from the input **Exodus** file, that preload alters the stiffness matrix, and eigenvalues are computed using this updated stiffness matrix.

```
SOLUTION
  case transfer
    receive_sierra_data
  case eig
    eigen
    nmodes=40
    shift=-3e6
END
```

The `receive_sierra_data` solution is specifically designed to read initial conditions and tangent stiffness state computed in a **Sierra/SM** analysis. Note that connections between different parts of the model specified in the **Sierra/SM** input deck are not transferred, e.g., contact conditions, MPCs, and joints. These conditions may be equivalently specified in **Sierra/SD**. The following options are available to `receive_sierra_data`.

include_internal_force When set to the default value (`on`), the input stress state is integrated to generate an internal force body load. This load is included in the right-hand side of subsequent static or transient analysis. If the **Sierra/SM** preload analysis is in static equilibrium, the externally applied forces from boundary conditions will be in balance with the internal forces generated by the elements. By default, the internal forces will be computed again in **Sierra/SD**. Thus, if the **Sierra/SD** simulation includes the same forces that preloaded the structure in the **Sierra/SM** simulation, then the internal force should also be included to keep the model in static equilibrium. An alternative is to exclude the preloading boundary conditions and set `include_internal_force=off`. In this case, **Sierra/SD** will compute no initial internal force, and the initial state of the model will be treated as if it is in perfect equilibrium. See the Verification Manual chapter "**Sierra/SM** to **Sierra/SD** Coupling" for more detail on this topic.

no_geom_stiff This option can be used to include (`geom_stiff=yes`) or ignore (`geom_stiff=no`) the preloaded stress contribution to the geometric stiffness matrix when transferring data (section 5.23). See sections 5.23.1 and 6.6.1.9 for more details.

Option `start_time` is needed if a simulation does not restart the input. The `start_time` sets the initial time of a transient analysis. By default, a transient case begins where a previous transient solution case ended, or the time transferred from a `receive_sierra_data` or other preload solution case. Otherwise, the default start time is 0.0. To set the start time in the input deck, specify `start_time`.

5.23.1. Geometric stiffness

Generally, elements in tension have higher effective stiffness, while elements in compression have lower effective stiffness. At sufficiently large compression, element stiffness can become *negative*, which causes severe problems for solver stability. Turning off geometric stiffness can be used for debugging purposes, or to evaluate what effect the geometric stiffness term has on model response.

If stiffness properties are not transferred from **Sierra/SM** to **Sierra/SD**, then it is often more accurate for **Sierra/SD** to contribute a geometric stiffness to the input stiffnesses. The capability is a well-established default feature, though in some cases it is excluded using the `no_geom_stiff` (or blockwise `geom_stiff`) option.

It is possible to hand off the state of **Sierra/SM**, including the *material state* as specified by the Lamé material library.[34] Consult section 6.1.4 for details on specifying material properties of a Lamé material.

Sierra/SM incorporates both material and geometric stiffness due to its fully nonlinear solid mechanics formulation. Thus, the default value for element blocks associated with Lamé materials is `geom_stiff=no`. When a built-in **Sierra/SD** material model is used, the default is `geom_stiff=yes`.

Consult the Verification Manual chapter "**Sierra/SM** to **Sierra/SD** Coupling" for more detail on this topic.

5.23.2. Receiving SM User Defined Data

The purpose of `receive_sierra_data` is to input data from a previous **Sierra/SM** analysis. Data relevant to the load is mostly read in automatically based on expected naming conventions. The fields relevant to receive_sierra_data are given in tables 5-41 to 5-43.

Only accurately labeled data will be transferred from the **Exodus** file. An exact match is typically required, although some variables have multiple valid names. For example, both `stress_xx` and `stressxx` are allowed. The data, typically written from **Sierra/SM**, may require special output requests (in the **Sierra/SM** input file) for proper naming. See the Example Problems or Verification Manuals for examples.

Alternatively, **Sierra/SD** also supports user-defined setup of *some* input variables using `initialize variable name = <key>` in the `FILE` section, where the appropriate `<key>` for each variable (if applicable) is listed in tables 5-41 to 5-43. `read variable` and `variable type` lines may be used to read input variables from a non-default field name. See section 4.3 for more information.

Exodus Label	**Description**	**<key>** (for initialize `variable name`)	**Effect**

displ_x
displ_y
displ_z
	nodal displacement	displacement(x)
		displacement(y)
		displacement(z)
			Added to mesh coordinates, the initial mesh configuration.

vel_x, vel_y
vel_z
rv_x
rv_y
rv_z
	nodal velocity and rotation	velocity(x),velocity(y)
		velocity(z)
		rotational_velocity(x)
		rotational_velocity(y)
		rotational_velocity(z)
			Used to set initial conditions for transient analysis

force_internal_x
force_internal_y
force_internal_z
	force from stress state	force_internal(x)
		force_internal(y)
		force_internal(z)
			Used for equilibrium diagnostics

Table 5-41. – Nodal data used in `receive_sierra_data`.

Volumetric stress in Table 5-42 determines a geometric stiffness and an internal force. The options to skip the geometric stiffness contribution are `no_geom_stiff` or the blockwise parameter `geom_stiff` (table 6-5). The internal force contribution may be skipped by setting `include_internal_force=off`. The Neo-Hookean, "neo_hookean", and hyperfoam, "hyperfoam", Lamé models (only) are supported. And of these, only hyperfoam has a state. For hyperfoam, COMP is one of L11, L22, L33, L12, L23, L31, L44, L55 or L66.

Exodus Label	**Description**	**`<key>`** for `initialize variable name`	**Effect**

stress_xx
stress_yy
stress_zz
stress_xy
stress_yz
stress_zx
	volumetric stress components	stress(xx)
		stress(yy)
		stress(zz)
		stress(xy)
		stress(yz)
		stress(zx)
			Used to compute geometric stiffness and internal force

element_density	density	density	Used to update element mass

left_stretch_xx
left_stretch_yy
left_stretch_zz
left_stretch_xy
left_stretch_yz
left_stretch_zx
	Element Stretch	left_stretch(xx)
		left_stretch(yy)
		left_stretch(zz)
		left_stretch(xy)
		left_stretch(yz)
		left_stretch(zx)
			Used for material tangent stiffness calculations for Lamé material models

lame_state_`<model>` `<model>`_COMP	Material State	lame_state_ `<model>`(COMP)	and for Lamé material model tangent stiffnesses

Table 5-42. – Element data used in `receive_sierra_data.`

Exodus Label	**Description**	**<key>** (for `initialize variable name`)	**Effect**

memb_stress_xx
memb_stress_yy
memb_stress_zz
memb_stress_xy
memb_stress_yz
memb_stress_zx
	fiber membrane
	stress components
		memb_stress(xx)
		memb_stress(yy)
		memb_stress(zz)
		memb_stress(xy)
		memb_stress(yz)
		memb_stress(zx)
			Geometric stiffness and internal force

Q1z Q2z	stress components Note: Q1z and Q2z are defined instead of memb_stress. Q1z, Q2z, or both can be defined	Q1zterm Q2zterm	Geometric stiffness and internal force
fibermod	fiber modulus	fibermodulus	Tangent stiffness
fiberdensity	fiber density	fiberdens	Mass
fiberthickness	thickness	fiberthickness	Tangent stiffness
ax	primary fiber direction	fiberdir	Tangent stiffness
ay	secondary fiber direction	fiberdir2	Tangent stiffness
compbendingmod	composite modulus	compbendingmodulus	Tangent stiffness
corddiam	cord diameter	corddiameter	Tangent stiffness
matrixmod	matrix material modulus	matrixmodulus	Tangent stiffness
E1 E2 G12	material modulus	N/A	Tangent stiffness *
nu12	Poisson ratio	N/A	Poisson ratio *
density	material density	N/A	Mass *

Table 5-43. – Data transferred in `receive_sierra_data` specific to orthotropic_layer materials. Some fields require the `from_transfer` keyword in the material definition (denoted by $*$ in the **Effect** column). These aren't read in directly, but are dependent on other table inputs.

Coordinate Update

The `receive_sierra_data` method uses **Sierra/SM** displacement to update the initial coordinates of the mesh. Hand-off refers to using the output from **Sierra/SM** as input to **Sierra/SD**.

The initial coordinates for **Sierra/SD** are updated with the transferred displacements.

$$\vec{x} = \vec{x}_o + \vec{u}$$

Here x_o is the location of the undeformed coordinates. Stiffness matrices, forces, mass, and tied MPCs are then computed in the updated frame. Tied MPCs can be defined via tied surfaces or contact.

Conserving mass through hand-off Because the **Sierra/SD** mass matrix is computed in the deformed configuration, handing off element density from **Sierra/SM** is essential to conserve mass, especially in models where material deformation is significant.

In order to hand off element-by-element density information from **Sierra/SM**, special output options are needed; likewise, special material input syntax is needed to handle transferred density in **Sierra/SD**. In the **Sierra/SM** input file, it is required to compute element density as a user-defined output field. For example, an analytic function of mesh variables can be used,

```
BEGIN FUNCTION elem_density
  type = analytic
  expression variable: m = element element_mass
  expression variable: v = element volume
  evaluate expression = "m/v"
END
```

and corresponding outputs can be used to output the density to the Exodus mesh.

```
BEGIN USER OUTPUT
  block = block_1
  compute element element_density as function elem_density
END

BEGIN RESULTS OUTPUT out
  database name = output.e
  database type = exodusII
  at time 0.0, interval = 1
  nodal variables = displacement as disp
  element variables = stress
  element variables = left_stretch
  element variables = rotation
  element variables = lame_state_hyperfoam
  element variables = element_density
END
```

In **Sierra/SD**, the material density must then be specified as a mesh variable in the material definition.

```
MATERIAL FOAM
  // original density = 26.
  density exo_var scalar element_density
  begin-lame-material
    begin parameters for model hyperfoam
    ...
    end
  end-lame-material
END
```

Input 5.11. Example of reading density from Sierra/SM output.

A detailed example with syntax describing the hand-off between **Sierra/SD** and **Sierra/SM** is in Chapter 3 of the **Sierra/SD** Example Problems Manual.[44]

Compatibility of Elements Between SD and SM

See section 7.35.

5.24. statics Solution Case

Parameter	**Type**	**Default**	**Description**
load	*Integer*		Load block to apply during solution case. See Section 4.5
FilterRbmLoad	*String*	noFiltering	rigid body components of loads to filter options are "nofiltering", "allstructural", and "rotationonly"

Table 5-44. – statics Solution Case Parameters.

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. When running a static solution with a floating structure, it is important to designate the **num_rigid_mode** parameter in the **PARAMETERS** section so the solver converges, even when no body motion is expected.

```
SOLUTION
  title "Example of a statics solution"
  statics
END
```

Note: When applying a time-varying load to a statics solution case, the load value at time=0 (typically the first step) is used.

5.25. superposition Solution Case

Parameter	**Type**	**Default**	**Description**
reduced_file	*String*		Ignored?

Table 5-45. – superposition Solution Case Parameters.

superposition is currently BETA release.
Enable with the "`--beta`" command-line option.

The **superposition** provides superelement recovery capability. This recovers physical space solutions from generalized degrees of freedom.

A CB model generates a transformation matrix consisting of a combined set of fixed interface and constraint modes. See section 5.3. This modal basis may be stored in an exodus file.

A netcdf file containing the reduced order model is also created at this time. Subsequently, this reduced model is inserted into a residual model for superelement analysis, say a transient analysis. That analysis outputs the standard exodus results, and may also generate output on the netcdf file. These data may be post-process using linear superposition to determine output quantities on the original interior degrees of freedom of the superelement. This is illustrated in Figure 5-2.

The **superposition** method requires a single argument, the file name of the reduced order model containing the output results. In addition, the **geometry_file** specified in the FILE section must contain the modal basis for the Craig-Bampton reduction. See the example in input 5.12.

```
SOLUTION
  superposition
    reduced_file=rom-out.ncf
END

file
  geometry_file basis-out.exo
```

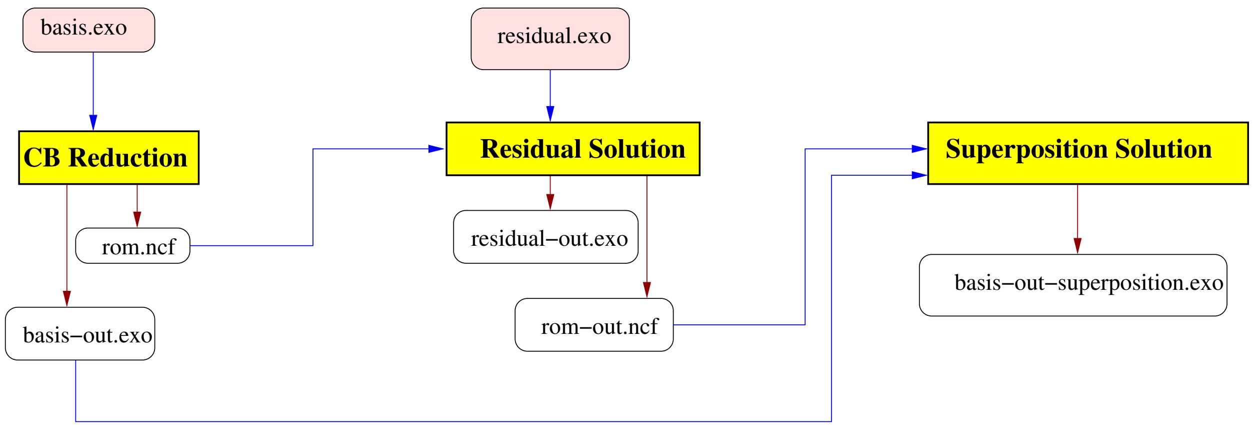


Figure 5-2. – Superposition Data Flow Diagram.

```
end
```

Input 5.12. Superposition Example. Output will be to "basis-out-superposition.exo"

Limitations

The superposition method is under development and contains the following limitations.

- Solutions are supported in serial at this time.
- Only displacement, velocity and acceleration may be output.
- Superposition is tested for Eigen and Transient solutions. There is no support for frequency domain solutions at this time.
- Data recovery is possible for a single superelement in each run.

5.26. tangent Solution Case

Parameter	Type	Default	Description

Table 5-46. – tangent Solution Case Parameters.

The **tangent** solution step requires a multicase solution (see paragraph 5.1.1). 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 computed again based on the current value of displacement.

During tangent the stiffness is computed again based on element configuration in the deformed state. Tangent also adds any stress stiffening effects from the preceding load case. Stress resultants from preloading are not added to stress resultants of subsequent cases after the tangent solution, but do contribute to increasing the effective stiffness computed in the tangent stiffness. For example, the stress stiffening that occurs from tension of a guitar string would be successfully captured using the tangent solution case. As another example the stiffness of a warped plate may be significantly different from the stiffness of a flat plate, tangent will take into account this change in stiffness.

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.

$$K_{tangent} = \frac{\partial f}{\partial u} \tag{5.26.1}$$

Preloading is incorporated by solving the eigenvalue problem for the tangent stiffness matrix.

5.27. transhock Solution Case

Parameter	**Type**	**Default**	**Description**
time_step	*Real*		Time step size
nsteps	*Integer*		Number of time steps to take
start_time	*Real*	0.0	Solution case start time
nskip	*Integer*	1	Results output frequency
rho	*Real*	1	Select time integrator
load	*Integer*		Load block to apply during solution case. See Section 4.5
write_files	*all*| *none*| *exodus*| *history*	all	Controls which result files are written during this solution.
SRS_damp	*Real*	0.03	Damping coefficient used for the shock response spectra calculation

Table 5-47. – transhock Solution Case Parameters.

The **transhock** solution method is used to perform a direct implicit transient analysis followed by computation of the shock response spectra (SRS) for the degrees of freedom in a specified node set. Frequencies (see Section 9.3) and nodesets for the SRS are selected in the required input deck `frequency` block. The options for configuring a **transient** solution case described in Section 5.28 are all applicable in the `transhock` solution case too. An example of a `transhock` solution case and `frequency` block are shown in input 5.13. Damping for the implicit transient portion of the simulation is defined in Section 6.7. The `srs_damp` is not used in the modal transient portion of the simulation. Examples are presented in inputs 5.8, 5.9 and 5.13.

A standard algorithm[60,61] is used to compute the shock response spectrum. MATLAB provides a nice, interactive environment for this analysis once the time integration has been performed in **Sierra/SD**. **Sierra/SD** performs identical calculations.

The shock spectrum procedure will compute acceleration results. The options specified in the `outputs` 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 outputs 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 shock spectra results will be for acceleration response for the nodes in the specified node set. *The calculated shock spectra are written to the frequency file (`frq`); they are not output to the Exodus results file.*

Transhock results differ in many subtle ways from transient results. Input deck requests for history output are ignored. The nskip parameter is interpreted differently . The time marching algorithm[16] formulation is different.

Note that the accuracy of the computed spectra is independent of the output resolution, `freq_step`.

```
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
```

Input 5.13. Transhock Example Input

5.28. transient Solution Case

Parameter	**Type**	**Default**	**Description**
time_step	*Real*		Time step size
nsteps	*Integer*		Number of time steps to take
start_time	*Real*	0.0	Solution case start time
nskip	*Integer*	1	Results output frequency
rho	*Real*	1	Select time integrator
load	*Integer*		Load block to apply during solution case. See Section 4.5
write_files	*all| none| exodus| history*	all	Controls which result files are written during this solution.
PredictorCorrector	*Integer*	-1	predictor-corrector implementation
ConstraintCorrectionFrequency	*Integer*	1	time-step frequency for constraint correction
ConstraintErrorDiagnostics	*yes/no*	no	prints constraint errors
flush	*integer*	50	Defines how often results are written to the exodus results file. See Section 4.6.1
nUpdateConstraints	*Integer*	0	Frequency to update constraints
FilterRbmLoad	*String*	noFiltering	rigid body components of loads to filter options are "nofiltering", "allstructural", and "rotationonly"

Table 5-48. – transient Solution Case Parameters.

The **transient** solution method performs a direct implicit transient analysis. The options are described here.

Time_step The `time_step` defines the time step size. The time step size is constant for a period. Multiple periods can be given as shown in Input 5.14.

Nsteps The `nsteps` defines the number of time steps to take. As shown in Input 5.14, multiple periods can be defined with different numbers of steps.

Start_time The `start_time` sets the initial time of a transient analysis. By default, a transient case begins where a previous transient solution case ended, or the time transferred from a `receive_sierra_data` or other preload solution case. Otherwise, the default start time is 0.0. To set the start time in the input deck specify `start_time`.

Nskip The `nskip` controls how many integration steps to take between outputting results. It defaults to 1, which is equivalent to outputting all time steps. Because transient analysis often takes little computational time per step, the overall runtime can be significantly reduced by choosing not to output the results at every step, i.e., setting `nskip` greater than 1. The `nskip` in the solution section can be overridden for history (Section 9.2) and linesample (Section 9.8.9) output by specifying `nskip` in those sections.

Load The **load** defines the name of the "load" block to apply during the solution case. See Section 4.5.1 for details.

Defining Multiple time Periods 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 tiny 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
  transient
    time_step      1.0e-5    1.0e-3
    nsteps         100       500
    nskip          10        1
END
```

Input 5.14. Solution Example

In this case, the user requested 100 time steps of $\Delta t = 10^{-5}$, followed by 500 time steps with $\Delta t = 10^{-3}$. There is no practical limit on the number of such regions that may be specified.

PredictorCorrector PredictorCorrector indicates whether predictor-corrector implementation is to be used (see Theory Manual). It defaults to 1. If it is 1, predictor-corrector is always used, while 0 indicates that it is never used. **ConstraintCorrectionFrequency** and **ConstraintErrorDiagnostics** are related to the constraint errors arising in the predictor-corrector implementation, with the first controlling how often the correction is applied, while the second can be used to examine the evolution of constraint errors of displacement, velocity and acceleration.

5.28.1. nUpdateConstraints

nUpdateConstraints is currently BETA release.
Enable with the "`--beta`" command-line option.
nUpdateConstraints defines the frequency to update constraints (e.g. MPCs and contact). When it is undefined or set equal to 0, the constraints will never be updated. A value of 1 results in the

constraints being updated every time-step. In addition to re-building the constraints, the nodal coordinates will also be updated at the corresponding time-steps based on the interpolation of any existing displacements on the mesh (if applicable).

The name of the displacement nodal variables can be defined by the `initialize variable name`, `read variable`, and `variable type` options in the `FILE` section, as shown in sections 4.3 and 5.23.

This option is related to the **nodesets_with_disp** option (section 4.4), which enables specifying displacements on a subset of all nodes via nodeset output. However, nodeset displacements do not currently support user-defined variable names as with nodal displacements.

5.28.2. FilterRbmLoad

Establishes a filter for rigid body components of the input load. The options are described in the table. It defaults to unfiltered. The parameter may need to be combined with the RbmTolerance and solver parameters. The **FilterRbmLoad** parameter is only supported for transient and static solution cases. For other solution cases this parameter will have no effect on the solution.

During rigid body mode filtering a net force in a rigid body mode will be counter-balanced by a force with a distribution defined by the rigid body mode shape times the mass matrix. For example to counter-balance a net force in X direction effectively a gravity load would be applied to the body where the sum of the forces of that gravity load is equal and opposite to the net force to be balanced. For rigid body filtering a well-defined mass matrix is required for both transient and static solution.

The rigid body load filtering should only be used on a model that has exactly the six standard rigid body modes (translation along three coordinate axes and rotation about those axes).

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

See Section 8.3.21 for more details about the use of this option.

Numerical damping

Two time integrator schemes are available for direct time integration. The method and the configuration of the integrator are selected using the keyword `rho`. If this keyword is not found, the time integrator defaults to a standard Newmark beta[17,28] integration scheme. With `rho` the Generalized Alpha method[21 16] is used, and the value of the numerical damping is controlled by `rho`.

Due to the inexactness of linear solvers, the Newmark beta integrator is conditionally unstable. Without damping, a solution may gradually diverge. Also, see subsection Linear transient analysis section Solution Procedures of the Theory Manual. *Either proportional damping or numerical damping is strongly recommended in all cases.*

The option `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 deck, the integrator defaults to the Newmark beta method.** Otherwise, the code uses the value of `rho` given by the user to configure 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 negative an error message is printed. `rho` determines **Newmark beta**, α_f, and α_m of the Generalized Alpha method. More detailed information on the implementation, and references can be found in the description of the method in the **Sierra/SD** Theory Manual.

The following conditions suffice to achieve second order accuracy and unconditional stability:

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

The parameters are determined to satisfy the conditions. 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 proportional damping, 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 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 $.5e+5$.

Initial Acceleration Determining the initial acceleration is necessary[29] for quadratic convergence. If force, velocity or displacement is specified, an initial solve may be required to determine a consistent acceleration. Remember that determination of the displacement at step $n + 1$ depends on values at the previous step. Specifically, the acceleration at the previous step is given by the solution to the equation,

$$A_n = M^{-1}(F_n + Kd_n + Cv_n)$$

where M, K, and C are the corresponding mass, stiffness and damping matrices.

If this mass solve is not performed, it is possible to introduce a spike in acceleration which can oscillate through time. Initialization can be a somewhat tricky process. An example set of use cases is provided in Appendix 8.5. By default, GDSW is used. Some combinations of MPCs can lead to a singular mass matrix that will cause solver errors. For these cases the initial mass solve is deactivated with the following command in the **parameters** block (see Section 4.4),

```
PARAMETERS
  DoInitialMassSolve = false
END
```

5.29. TSR_preload Solution Case

Parameter	**Type**	**Default**	**Description**
linedata_only	*True*|*False*	False	Indicates that no system matrices should be computed, but the linedata specified in the **linesample** file should be computed for verification of data transfer.

Table 5-49. – TSR_preload Solution Case Parameters.

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. Stress results are read from the geometry file, an equivalent internal force is computed, and that internal force is combined with the applied force throughout the transient run. If temperature data is also included in the file, it will be read and used to compute temperature dependent material properties. A `tsr_preload` requires a multicase solution, and it must be followed by a transient dynamics or statics solution (see paragraphs 5.1.1 and 5.28 respectively).

Note that since the stresses are converted into a force, and since there is no immediate deformation in transient dynamics, the elastic stresses output by **Sierra/SD** will be 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 9.8.9 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 8.3.7), may provide equivalent capability.

Data in the **Exodus** file from which TSR data is read must strictly match the following criteria. There must be one time step in the result. That time step must have some different element fields defined. These correspond to the six stress values of the stress tensor and the number of stress tensors defined per element must correspond exactly to the number of integration points for that element. For instance, a hex20 element requires exactly 27 stress tensor values per element. An error is produced if the number of data points read in does not match the number of integration points for that element. For instance, a fully integrated hex8 will produce an error if reading in Gauss point data produced by a fully integrated hex20 element. Shell and beam type elements are not supported in tsr_preload.

The labels for the stresses must be as shown in the table below where `SigXX` is interchangeable with `Sigma_XX` or `STRESS_XX`. Both sequential and ijk numbering are supported for integration point data. For sequential numbering, replace `%d` with an integer representing the integration point value (0 to 26). Do not zero pad. IJK labels are detailed in section 9.7.20.6.

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

Support for user-defined stress labels (and reading from an arbitrary input step) is available using the `initialize variable name` interface shown in section 4.3. Note that the labels will still need to follow the previously-outlined naming convention for integration-point data, i.e., the user-defined stress label defines the *root* of the stress component name.

An example of a transient simulation with TSR preload follows.

```
SOLUTION
  title 'Pure bending from initial stress'
  case TSR
    tsr_preload
    load 1
  case bend
    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-bend.exo`. The first of these has little useful information. The second contains the displacements (or other variables) from the transient analysis.

5.29.1. Line Sample

One additional feature for thermal structural response is the ability to do line sampling 9.8.9 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 for data stored on integration points using the variables names described above. Line sample is used for energy deposition (see the *Two Element Exponential Decay Variation Hex20* problem[46]). 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.

5.30. GeometricRigidBodyModes Solution Case

Parameter	**Type**	**Default**	**Description**

Table 5-50. – GeometricRigidBodyModes Solution Case Parameters.

Nominal rigid body modes may be determined from the coordinates. No attempt is made to account for boundary conditions. This solution method requires that the GDSW linear solver be used.

The intent of the examples is to first introduce rigid modes, next use the modes to solve an eigenvalue problem, and then demonstrate the Modal Transient capability 5.18. The third example uses the modes in a modal transient simulation to deflate out the rotations. This section depends on Section 8.3.21.

The geometry rigid body mode capability will always generate six modes and these modes are explicitly ordered +X, +Y, +Z, +RX, +RY, +RZ.

Rigid body modes are requested in the `Solution` block.

```
SOLUTION
  geometric_rigid_body_modes
END

PARAMETERS
  num_rigid_mode 6
END
```

The number of rigid body modes must also be specified. Only values of 1,6 or 7 are supported.

Rigid body modes can be incorporated into the modes computed in a modal analysis, and then used for other purposes. The resulting mode shapes are more accurate. Also, the rigid body modes themselves are ordered in a way that makes sense to humans. Without the GRBM case, the displacements and rotations are mixed together.

```
SOLUTION
  case rigid
  geometric_rigid_body_modes
  case flexible
  eigen
  nmodes 10
  shift -1e6
END

PARAMETERS
  num_rigid_mode 6
END
```

Rigid body modes are the 6 lowest frequency eigenvectors. In this case 4 more modes are computed, for 10.

In this example a modal transient simulation uses the geometric rigid body modes to deflate out the (infinitesimal) rotation, while retaining the translational rigid body modes. This is equivalent to use of the `FilterRbmLoad` for direct transient solutions (though accomplished differently).

```
SOLUTION
  case out
```

```
    geometric_rigid_body_modes
  case vibration
    eigen
    nmodes 10
  case filter
    modalfiltercase
    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
```

5.31. waterline Solution Case

Parameter	Type	Default	Description
max_iterations	*Integer*	100	Maximum number of solution iterations
tolerance_force	*Real*	1.0e-6	Target force balance accuracy
point_a	*Real*(3)		Coordinates of point on estimated water surface
point_b	*Real*(3)		Coordinates of point on estimated water surface
point_c	*Real*(3)		Coordinates of point on estimated water surface
VizOption	*none*|*Ensight*	none	Whether Ensight writes a file for visualizing the waterline plane

Table 5-51. – waterline Solution Case Parameters.

It can be advantageous to determine the waterline of a ship prior to commencing more complex analysis. The **waterline** capability solves the nonlinear geometric equations of equilibrium for a rigid ship in water. An example is shown in input 5.15.

```
SOLUTION
  case 'waterline'
    waterline
      max_iterations 100
      tolerance_force 1.0e-6 // absolute tolerance on force convergence
      point_a 0 0 0 // coords of point 'A' on estimated water surface
      point_b 1 0 0 // coords of point 'B' on estimated water surface
      point_c 1 1 0 // coords of point 'C' on estimated water surface
    load 1

  case 'transient'
    ...
END
```

Input 5.15. Waterline solution case

```
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
```

Input 5.16. Corresponding Waterline load

The arguments `point_a`, `point_b` and `point_c` indicate the Cartesian coordinates of three points A, B, C 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 shown in Figure 5-3. The Newton's method implementation then uses this plane as the initial guess, and begins iterations 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 moving the body may also be written.

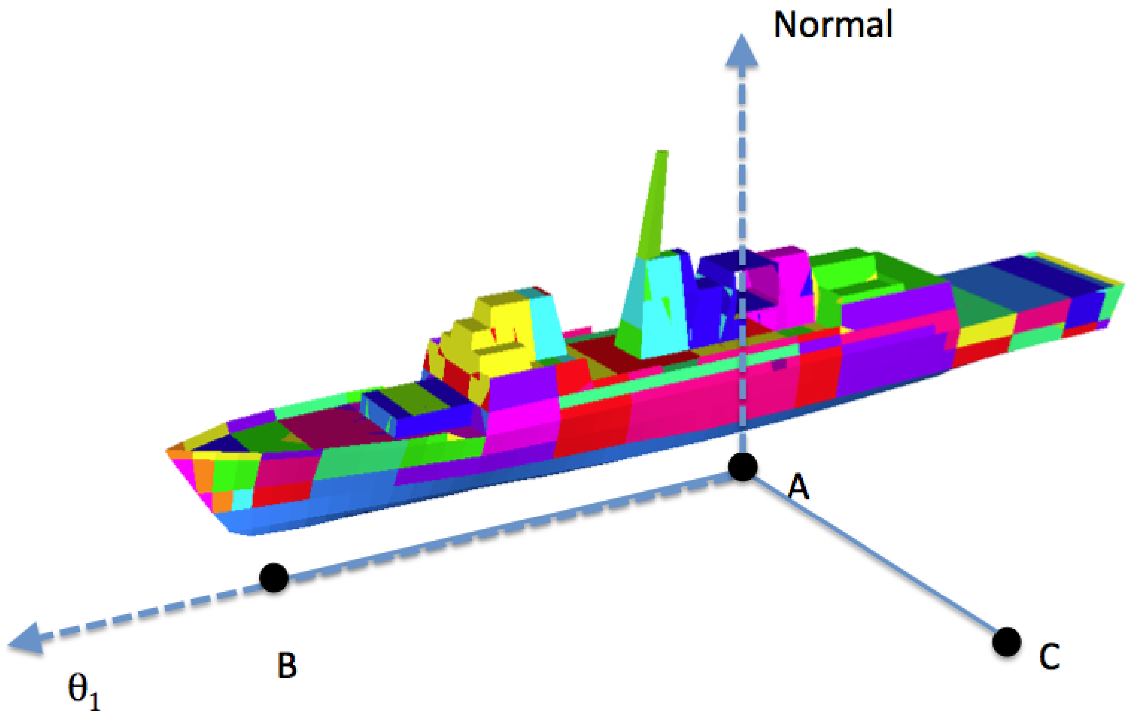


Figure 5-3. – Waterline Coordinate Definition. The plane of the surface is defined by three points: A,B, and C. The θ_1 rotation is about the line from A to B, while the normal is defined using the right-hand rule.

The optimization is configured as follows.

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.

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. Gravity is specified using the standard load keywords of a body load with a gravity vector. However, for the `waterline` solution, the magnitude of the gravity vector is relevant. The gravity direction is *always* directed opposite the normal to the surface for this solution type. The **function** defines the pressure as a function of depth. In the example of input 5.15, 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, 9, for details.

5.31.1. Limitations

There are some limitations to this method.

gradient-based optimization: These powerful algorithms are based on nonlinear gradient-based optimization and have subtle limitations. Limitations are listed below.

1. Singular tangent matrices are generated in various conditions, which cause the solution to terminate. A common condition causing a singular tangent matrix is a body completely submerged in a constant-density fluid. For simplicity, consider an unrotated cube of edge length S. These arguments are valid for any rigid body. The net force on the cube is,

$$
\begin{aligned}
F_{\text{net}} &= (A_{\text{bottom}}P_{\text{bottom}} - A_{\text{top}}P_{\text{top}}) - mg && (5.31.1)\\
&= \rho_f g S^2 (h_{\text{bottom}} - h_{\text{top}}) - mg && (5.31.2)\\
&= \rho_f g S^3 - \rho_s g S^3 && (5.31.3)
\end{aligned}
$$

   where ρ_f and ρ_s are the densities of the fluid and solid respectively. Significantly, the net force does not depend on the average depth. Thus,

$$K_t = \frac{\partial F_{net}}{\partial z} = 0.$$

   $K_t = 0$ for a ship that is completely out of the water too.

   Real seawater is not constant density. 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 5-4 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.

2. Gradient-based solution methods often have trouble with local minima. These can occur in the case of unstable systems, such as a 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.

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

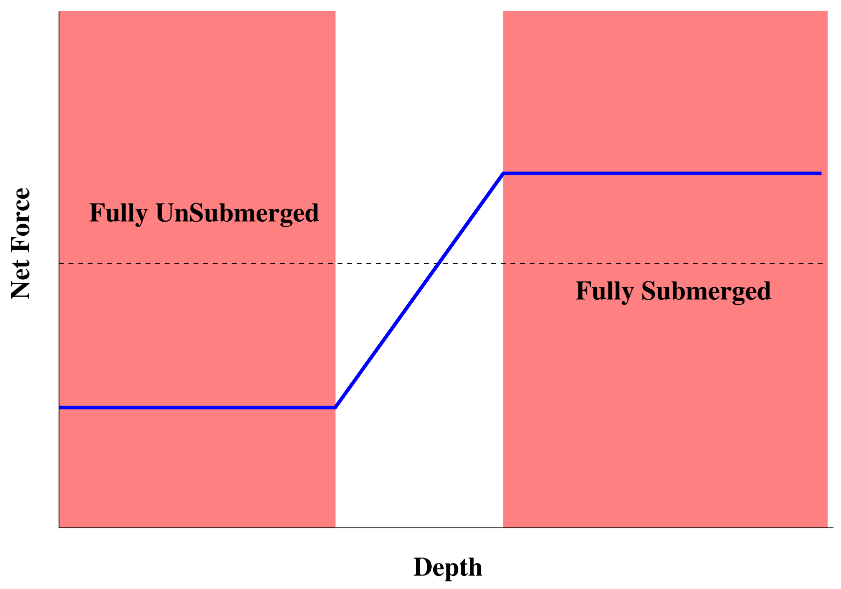


Figure 5-4. – Net Force vs depth for a Rigid Body. Only the unshaded region, where the body is partially submerged, has a non-singular tangent matrix.

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

One and only one sideset defines the wetted surface. Its outward direction should point into the water. There is no check for a reversal of the normal vectors 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.

5.32. Gap Removal Solution Case

Parameter	**Type**	**Default**	**Description**
ignore_gap_inversion	*true*|*false*	false	suppress fatal error, gap output behavior if true

Table 5-52. – Gap Removal Solution Case Parameters.

If two meshes are tied using either **tied data** or **contact definition**, then along the interface opposite elements may initially overlap of leave gaps. Gap removal, which is done by default, attempts to remove these gaps and overlaps. Gap removal is the same as **initial overlap** removal.

The gap_removal solution case is used to debug contact setup prior to submitting a full run. The gap_removal solution case runs quickly and uses low memory. This solution method enables visualization of the constraints created and the gap removed from tied data 10.1 and contact definition 10.2 blocks. The gap_removal solution method reads in the mesh, applies the contact search and gap removal algorithm and writes out the output mesh with gap removed. An example input is given below.

```
SOLUTION
  gap_removal
END

TIED DATA
  surface 1,2
  name "tied_1-2"
  search tolerance 1.0e-3
END
```

Removal of contact gaps is essential to maintaining rigid body invariance. This is illustrated in Appendix 10.3. However, the removal of gaps in tied surfaces can occasionally result in distorted elements which may make it difficult to impossible for the solver to converge. Thus, it can be advantageous to investigate the results of gap removal before committing to a full and expensive solution case.

Gap removal output includes the two element variables elementInversionFlag and elementQuality. Any detected inverted element is flagged (one) by the Boolean (zero or one) elementInversionFlag . The variable elementQuality is a condition number of the deformed geometry of each element (one is ideal and high is worse) The same element quality metric is used as described in Section 9.8.3.

If gap removal inverts any element the file name extension “-gap” will be appended to the output exodus file and a fatal error will be given. One of the parameters described in Section 4.4 influences the Gap Removal solution case. The parameter is ignore_gap_inversion. It is false by default. Set it to true to suppress both the fatal error and the “-gap” output behavior.

Gap removal for lofted surfaces is discussed in Section 10.3.

In addition to the element shape, information diagnostics regarding the contact constraints are available. The rslt file will list basic information on the number of constraints found. Detailed visual information on constraint locations is obtained by requesting **constraint_info** in the **outputs** block as described in Section 9.8.1. Furthermore, adding MPC to the **echo** block prints every contact created MPC as described in Section 9.9.3.

Coupled Electro-Mechanical Analysis Piezoelectricity is producing electrical charges on a surface by the imposition of mechanical stress. **Sierra/SD** supports coupled electro-mechanical physics to model piezoelectric materials subjected to electrical and mechanical forces. This support includes static, transient, eigen, and direct frequency response solution methods, piezoelectric and dielectric material models (6.5.9, 6.5.9), and voltage measurement based inverse methods such as material and source identification. Electrical boundary conditions such as charge-based Neumann (8.3.13) and voltage-based Dirichlet(8.1.3, 8.1.4) boundary conditions are also supported.[45]

6. MATERIALS

The material section has a (unique) material identifier (an integer or a string name). The material identifier is used in assigning material properties to element blocks. Material types and their parameters are summarized in Table 6-1.

Table 6-1. – Material Stiffness Parameters.

material type	parameters
isotropic	any two of K, G, E or nu
orthotropic	nine C_{ij} entries
orthotropic_prop	E1, E2, E3, nu23, nu13, nu12, G23, G13, G12
anisotropic	21 C_{ij} entries

For example,

```
MATERIAL steel
    isotropic
    E 3e7
    nu .3
END
```

A materials may be **isotropic**, **orthotropic**, **orthotropic_prop**, **anisotropic**, or **isotropic_viscoelastic**.

The Joint2G element 7.23 has material models for joints including an elastic-plastic model.

The default stress/strain ordering is xx, yy, zz, zy, zx, xy, or

$$\begin{bmatrix} 0 & 5 & 4 \\ 5 & 1 & 3 \\ 4 & 3 & 2 \end{bmatrix}. \tag{6.0.1}$$

6.1. Elastic materials

Elastic materials may be isotropic section 6.1.1, orthotropic section 6.1.2, or anisotropic section 6.1.3. Some material models from the Lamé library are available section 6.1.4.

6.1.1. Isotropic

Isotropic materials require specification of two of the following parameters. They can be defined directly as `parameter = <real>`, as functions of temperature (Section 6.5.6), or as spatially dependent properties (Section 6.5.7.)

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

$$E = 3K(1 - 2\nu), \quad G = \frac{3KE}{9K - E}.$$

Internally, **Sierra/SD** stores the values of `K` and `G`.

6.1.2. Orthotropic

Orthotropic material entry is similar 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, the stress/strain ordering (6.0.1) is the default.

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. As with isotropic materials, temperature-dependent parameters may be defined via a function as `parameter = function <string>` (see Section 6.5.6). Note that all elastic materials must satisfy requirements that the elasticity matrix is positive definite.

```
MATERIAL honeycomb
   orthotropic_prop
   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`).

The `receive_sierra_data` section 5.23 solution case transfers material parameters by file, also using the syntax `parameter = from_transfer`. Here is an example:

```
MATERIAL 13
  orthotropic_layer
  E1 = 508.7
  E2 = 7641.0
  nu12 = 1.293
  G12 = 115
  density=0.5
END
```

If sensitivity analysis is being performed (see Section 4.8), 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.

6.1.3. Anisotropic

Anisotropic materials require specification of a 21 element C_{ij} matrix corresponding to the upper triangle of the 6×6 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. The stress/strain ordering (6.0.1) is the default.

This is generally consistent with published Materials Science data. However, NASTRAN and Abaqus use a different convention. An input deck illustrating anisotropic material input is provided in Section 11.2.

If an element block uses a coordinate system the anisotropic material is defined in the $\hat{r}$, $\hat{s}$, $\hat{t}$ local frame (section 9.8.11). If an element block does not use a coordinate system the anisotropic material is defined in the X, Y, Z frame.

Note that anisotropic materials are **NOT** defined in the element orientation (section 9.8.4).

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.

6.1.4. Lamé Material

Lamé material is currently BETA release.
Enable with the "- -beta" command-line option.

Sierra/SD provides a limited capability to use linearized versions of the non-linear Lamé material models in **Sierra/SD**. This can be used in conjunction with the capability to **hand-off** a nonlinear preload to a linear **Sierra/SD** analysis. This hand-off is accomplished by reading in the element variables **stress**, **left_stretch**, and Lamé state (e.g. `lame_state_hyperfoam`) at the last time-step of your **Exodus** input file (see section 5.23). One example where this would be useful is computing the tangent stiffness of a compressed foam.

Lamé materials are defined in a material section. As in input 6.1. A Lamé material definition begins with `begin-lame-material` and ends with `end-lame-material`. Currently, only Neo-Hookean and Hyperfoam materials are supported.

```
MATERIAL 1
  begin-lame-material
    begin parameters for model Hyperfoam
      bulk modulus    = 1.e6
      Poissons ratio  = 0.1
      n = 3
      shear = 3.74e6, -3.17e6, 1.18e4
      alpha = 2.536, 2.090, -8.807
      Poisson = 0.5630, 0.5507 0.3662
    end
  end-lame-material
  density = 5.0
END
```

Input 6.1. Example material section for a Lamé Hyperfoam material model.

Note: unlike the rest of a **Sierra/SD** input file, the material model definition (emphasized text in input 6.1) must strictly follow syntax rules including newlines. Details of the allowed syntax for each material model are given below.

Lamé Neo-Hookean model

Acceptable syntax for Neo-Hookean models is given below.

```
BEGIN PARAMETERS FOR MODEL NEO_HOOKEAN
     #
     # Elastic constants
     #
     YOUNGS MODULUS = <real> E
     POISSONS RATIO = <real> ν
     SHEAR MODULUS  = <real> G
     BULK MODULUS   = <real> K
     LAMBDA         = <real> λ
     TWO MU         = <real> 2μ
   END [PARAMETERS FOR MODEL NEO_HOOKEAN]
```

A detailed discussion of the theory of the Neo-Hookean model can be found in the Sierra Solid Mechanics User Manual.[47]

Lamé Hyperfoam Model

Acceptable syntax for Hyperfoam models is given below.

```
BEGIN PARAMETERS FOR MODEL HYPERFOAM
     #
     # Elastic constants
     #
     YOUNGS MODULUS = <real> E
     POISSONS RATIO = <real> ν
     SHEAR MODULUS  = <real> G
     BULK MODULUS   = <real> K
     LAMBDA         = <real> λ
     TWO MU         = <real> 2μ
     #
     # Strain energy density
     #
     N = <integer> N
     SHEAR    = <real_list> μ_i
     ALPHA    = <real_list> α_i
     POISSON  = <real_list> ν_i
   END [PARAMETERS FOR HYPERFOAM]
```

As with Neo-Hookean models, a detailed discussion of the theory of the Hyperfoam model can be found in the Sierra Solid Mechanics User Manual.[47]

6.2. Acoustic

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 air
  acoustic
  density 1.293
  c0 332.0
  cavitating
  pvapor 0.0
END
```

Nonlinearity can be activated by the keyword `nonlinear`. 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.[26]

Cavitation can be activated by the keyword `cavitating`. This requires an additional parameter, `pvapor`, which is the vapor pressure with a default value equal to 0. Cavitating elements also require the definition of vectors for `hydrostatic_gravity` and `free_surface_point` in the **block** section as shown in input 6.2. `hydrostatic_gravity` and `free_surface_point` are vectors used to compute the external pressure at the block.

```
BLOCK 1
  material 1
  hydrostatic_gravity = 0 0 -32.2
  free_surface_point  = 0 0 34.0
END
```

Input 6.2. This is an example of a block section for a cavitating material.

For computational acoustics see Section 4.6.4.

6.3. Linear Viscoelasticity

The limiting moduli of viscoelastic materials are the glassy modulus occurring at $t = 0$ ($\omega = \infty$), and the rubbery modulus at the opposite extreme, $t = \infty$ ($\omega = 0$). Linear viscoelastic materials require the specification of the density, and the limiting moduli K_g, G_g and K_∞, G_∞. Additionally the Prony series for the viscoelastic materials is to be specified using keywords K_coeff, K_relax, G_coeff, and G_relax. Each parameter is required.

For the bulk modulus K, the Prony series parameters are defined by the following equation:

$$K(t) = K_\infty + (K_g - K_\infty) \sum_{1 \le i \le n} K_c^{(i)} e^{-t/K_r^{(i)}}. \tag{6.3.1}$$

Here K_c and K_r are shorthand for K_coeff and K_relax, and similarly for the shear modulus. The Prony series for the shear modulus may have a different number of terms, say m, than the n term bulk modulus series. This makes is possible to simulate a material with linear viscoelastic shear modulus G, and isotropic bulk modulus. Think about what happens at $t = 0$ and $t = +\infty$ in equation (6.3.1). For the material to be linear viscoelastic, it is necessary that

$$\sum_i K_c^i = 1 \qquad , \sum_i G_c^i = 1, \tag{6.3.2}$$

Note that the number of terms in K_coeff and K_relax must be the same, and the number of terms in G_coeff and G_relax must be the same.

E_g, E_inf, G_g, and G_inf may be constant or temperature dependent. Temperature functions can specify the value for the limiting moduli, for a given value of temperature. For example, if the limiting moduli depend linearly on temperature, a linear function can be specified for the values of E_g, E_inf, G_g, and G_inf. We refer to the example given below for the specifics on how to set this up.

Temperature-dependence

The current temperature must be defined in each block that is assigned a viscoelastic material with temperature-dependent behavior. A value of 0.0 is the default. See Section 8.3.7 for more details on temperature specification in **Sierra/SD**.

Optional reference (T_0) and glassy (T_g) temperatures may be specified for viscoelastic materials. The reference temperature T_0 is the temperature at which the input viscoelastic constants are defined. The reference temperature T_0 may differ from T_g.

If none of the moduli are specified as functions, the values specified for these parameters determine which model is used for temperature-dependent behavior. The T_g parameter may be given as a constant or as a spatially dependent property (Section 6.5.7.)

Two models are available:

(i) the Williams-Landel-Ferry (WLF) model,[67] and

(ii) the Hinnerich's model.

The WLF model[1,23] is used when shifting temperatures above T_g, while the Hinnerich's model is used when shifting temperatures below T_g. Both models incorporate the reference temperature T_0. The WLF and Hinnerich's model use model-specific constants. Units of temperature must be consistent with the values of the model-specific constants. The shift factors computed from the Hinnerich's or WLF equations are used to scale the coefficients in the Prony series.

In terms of the current temperature in the element, T_{elem}, the WLF model is

$$\log_{10}(a_T) = -\frac{C_1(T_{\text{elem}} - T_0)}{C_2 + T_{\text{elem}} - T_0}. \tag{6.3.3}$$

The material parameters C_1, and C_2 are determined experimentally. Typically, T_0 is the glass transition temperature of the material of interest. The shift factors computed from the WLF equation are a strong function of temperature.

The Hinnerich's model, provided by Terry Hinnerich's, accurately characterizes many viscoelastic materials below the glassy transition temperature. Its form is

$$\log_{10}(a_T) = a_{T1} * (1 - e^{a_{T2}*(T_{\text{elem}} - T_0)}), \tag{6.3.4}$$

where a_{T1} and a_{T2} are user-specified constants. This equation used to determine an approximate set of shift factors when experimental data for a particular material is not at hand.

Note if a model is shifted through T_g a composite shift is used. For example, if T_0 is less than T_g and T_{elem} is above T_g, first a Hinnerich's shift is used to shift parameters from T_0 to T_g, then a WLF shift is used to transition those constants from T_g to T_{elem}. These shifts are automatically computed given T_0, T_g, block temperature, C_1, C_2, a_{T1}, and a_{T2}. Note that if these shifting parameters are not specified in the input file, then no shifting will be done and the relaxation times as specified in the input deck will be used. If T_g is unspecified, then it defaults to the provided value of T_0.

Either specify each shifting coefficient, or specify none of them.

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,

$$K_{relax}[i] = a_T K_{relax}[i] \tag{6.3.5}$$
$$G_{relax}[i] = a_T G_{relax}[i]. \tag{6.3.6}$$

Example 6.3 demonstrates how Hinnerich's model is used to set a linear viscoelastic material.

```
MATERIAL foam
  isotropic_viscoelastic
  T_0= 10
  T_g = exo_var scalar t_g_input
  C_1=15.
  C_2=35.
  aT_1=6.
  aT_2=.0614
  K_g = function 1
  K_inf 1.e7
  G_g 1.e2
  G_inf 12.
  K_coeff .5 .5
  K_relax 3. 2
  G_coeff .5 .5
  G_relax 1 3
  density 0.288
END
```

Input 6.3. Hinnerich's viscoelastic material specification

As required in linear viscoelasticity, the coefficients of both K and G sum to 1.0. 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 element temperature T_{elem}. The T_g value is shown reading from an input mesh exodus field named 't_g_input'. T_g can also be specified as a constant like the other parameters.

The SM-style input syntax using the viscoelastic swanson model is also supported in SD. The parameters in that style input are used in a preprocessing step to convert into the Hinnerich parameters. An example of this associated SM input is shown in input 6.4.

```
MATERIAL foam
  viscoelastic_swanson
  density = 0.11330E-08
  bulk modulus = 1500
  a1 =    3.199653
  p1 =   -0.155565
  b1 =    0
  q1 =    0.5
  cut off strain = 0.01
  prony shear infinity =    0.021048
  prony shear  1 =      3.962759E-01
  prony shear  2 =      1.748129E-01
  prony shear  3 =      1.394151E-01
  prony shear  4 =      8.754736E-02
  prony shear  5 =      6.001257E-02
  prony shear  6 =      3.952786E-02
  prony shear  7 =      2.694688E-02
  prony shear  8 =      1.717840E-02
  prony shear  9 =      1.296047E-02
  prony shear  10 =     5.981649E-03
  prony shear  11 =     8.853162E-03
  prony shear  12 =     9.439881E-03
  shear relax time  1 =  1.00000E-04
```

```
   shear relax time  2 =  1.00000E-03
   shear relax time  3 =  1.00000E-02
   shear relax time  4 =  1.00000E-01
   shear relax time  5 =  1.00000E+00
   shear relax time  6 =  1.00000E+01
   shear relax time  7 =  1.00000E+02
   shear relax time  8 =  1.00000E+03
   shear relax time  9 =  1.00000E+04
   shear relax time  10 = 1.00000E+05
   shear relax time  11 = 1.00000E+06
   shear relax time  12 = 1.00000E+08
   wlf coef c1 =              5.94
   wlf coef c2 =            151.60
   wlf tref =                23.00
END
```

Input 6.4. SM viscoelastic swanson material specification

Limitations of Viscoelastic Use Linear viscoelastic materials are "linear" in the sense that (linear) transient dynamics accommodates them exactly. There are limitations for the use of these materials in **Sierra/SD**.

1. Viscoelastic materials in **Sierra/SD** only support a linear constitutive model and small deformation. Internal forces are handled differently in linear and nonlinear transient simulations. When using viscoelastic materials in a *nonlinear* transient simulation, in each element block with a viscoelastic material it is necessary to specify `nonlinear=no`.

2. A statics simulation models long-time or slow response material properties. The viscoelastic materials used are the G_∞ and K_∞.

3. Likewise, `Eigen` 5.8 solutions and the modal based solutions derived from them apply only the first terms of the Prony series for G_∞ and K_∞, which are used to define the elastic constants of an isotropic elastic material. This is because the real modal solution must use a constant mass and stiffness matrix, and has no damping contribution.

4. It is possible to evaluate the eigenvalues expanded about a given frequency, `viscofreq`. See the discussion of **ceigen** in Section 5.19.3. The damping matrix, C, is taken into account in the eigenvalue problem. Few modal solutions are adapted to use these complex modes.

6.4. Complex Viscoelastic

The isotropic viscoelastic complex material model is currently BETA release.
Enable with the "`--beta`" command-line option.

Complex isotropic viscoelastic materials may be defined for Direct FRF solution cases with the **isotropic_viscoelastic_complex** keyword. This material type has 4 required parameters, representing the real and imaginary shear and bulk moduli: `Greal`, `Gim`, `Kreal`, and `Kim`. (The real and imaginary components are commonly known as the storage and loss moduli, respectively.) Each complex viscoelastic parameter must be defined by a frequency-varying function, e.g., `parameter = function <string>`.

Example 6.5 demonstrates how a complex linear viscoelastic material is set.

```
MATERIAL 99
  isotropic_viscoelastic_complex
  Kreal = function 1
  Kim = function 2
  Greal = function 3
  Gim = function 4
  density 1.0
END
```

Input 6.5. Viscoelastic material specification

6.5. Properties

This section provides details on available keywords in **Sierra/SD** for input of material properties.

6.5.1. Density

For solutions requiring a mass matrix, all material specifications must define density. This can be set via the keyword **density** followed by a scalar value. Alternatively, the density can be defined as a temperature dependent property (Section 6.5.6, as a spatially dependent property (Section 6.5.7.), or be defined via a file transfer to `receive_sierra_data` solution case with (`density = from_transfer`).

6.5.2. High Cycle Fatigue

Material parameters for high cycle fatigue (Section 5.11) may be provided to define the statistical failure behavior of materials. These properties are summarized in Table 6-2.

Parameter	**Type**	**Default**	**Description**
Fatigue_A1	*Real*	0.0	S-N curve constant
Fatigue_A2	*Real*	-3.0	S-N curve slope
Fatigue_A3	*Real*	0.0	S-N curve translation, $S_{eq} = S(1 - R)^{A_3}$
Fatigue_A4	*Real*	0.0	S-N curve endurance limit, ignored
Stress_Ratio	*Real*	-1.0	R, the ratio of max/min stress. The default -1 indicates oscillation about a zero-mean stress state
Fatigue_A	*Real*	1.0	S-N curve constant
Fatigue_m	*Real*	3.0	S-N curve coefficient
Fatigue_Stress	*Real*	1.0	Stress unit conversion factor
std_err	*Real*	0.0	to shift S-N curve by material uncertainty
t_dist	*Real*	0.0	to shift S-N curve by material uncertainty

Table 6-2. – Material Section Parameters for Fatigue Parameters.

6.5.3. S-N curve Definitions

There are two competing S-N curve definitions in literature, which are equivalent for S-N curves that are linear in log-log space, and are both supported by **Sierra/SD**. The first is used by Wirsching, Paez, and Ortiz in their book *Random Vibrations Theory and Practice*,[69] and is supported with the Fatigue_A, and Fatigue_m parameters:

$$NS^m = A$$

The second is adaptable to a wider variety of materials, and is defined in Chapter 9 of MMPDS[20] (otherwise known as MIL-HDBK-5) as:

$$\log_{10}(N) = A_1 + A_2 \log_{10}(\ S\ (1 - R)^{A_3} - A_4\ )$$

For real materials, A_1 is always positive, and A_2 is always negative. A_2 gives the S-N curve its negative slope, and A_1 represents the crossing of the S-axis on the S-N curve. For the case of an

S-N curve which is linear in log-log space, $A_4 = 0$ and the analysis is assumed to operate at a constant stress ratio such that:

$$S_{eq} = S(1-R)^{A_3} = S\left(2^{A_3}\right)$$

and

$$\log_{10}(N) = A_1 + A_2 \log_{10}(S_{eq})$$

These functions are equivalent, and the material properties can be mapped to each other by:

$$A_1 = \log_{10}\left(\frac{A}{(1-R)^{A_2 * A_3}}\right), \quad A_2 = -m$$

Note that A_4 is equivalent to the endurance limit of the material, if it exists. Parameters $A_1,\ A_2,\ A_3$, and A_4 can all be found in MIL-HDBK-5[20] as empirically derived values for most metallic materials. See the example in Section 6.5.5. Since the Narrowband approach has an inherent assumption of zero mean stress, the default is R = -1. While the user can specify a different R-ratio, such usage would be inconsistent with the Narrowband and Wirsching methods.

6.5.4. *S-N Curve Units*

While **Sierra/SD** requires only a consistent set of units, the introduction of experimental data with their own units can confuse the solution. This is particularly challenging because the units for some parameters are mixed, and the data is gathered and presented in only a single unit system.

The stress scaling parameter helps reduce that problem. Consider the scaled equation for narrow band damage.

$$D_{NB} = \frac{\nu_o^+ \tau}{A}(\sqrt{2}\sigma_s F_{ss})^m \Gamma\left(\frac{m}{2}+1\right) \tag{6.5.1}$$

Here F_{ss} is the stress scaling parameter. This parameter lets the user convert from the unit system of the analysis to the unit system of the test data. Table 6-3 provides common conversions of these stresses.

Model Units	Experimental Units			
	PSI	Ksi	SI	CG
PSI (lbs/in^2)	1	0.001	6894.76	68947.6
Ksi (K lbs/in^2)	1000	1	6894757	68947573
SI (N/m^2)	.000145037738	1.4503774e-7	1	10
CG (dynes/cm^2)	.0000145037738	1.4503774e-8	0.1	1

Table 6-3. – Common Unit Scalings using **Fatigue_Stress_Scale**. The MIL-HNBK typically uses Ksi for experimental units.

6.5.5. *Typical Material Data for Fatigue*

Figure 6-1 shows typical fatigue data from MIL-HDBK-5.[20] For a range of stress ratios in this material, the number of cycles to failure is represented by the equation,

$$\log_{10} N_n = 9.65 - 2.85 \log_{10}(S_{eq} - 61.3)$$

In this range, and for this material, we have the definitions given in Table 6.5.5.

Parameter	**Value**	**Comment**
Fatigue_A1	9.65	offset in S-N curve. $A_1 = \log(A)$
Fatigue_A2	-2.85	slope of S-N curve, $-m$
Fatigue_A3		S-N curve translation, $S_{eq} = (1 - R)^{A_3}$
Fatigue_A4	61.3	endurance limit
Stress_Ratio	-0.60	R, determines equivalent stress, $S_{eq} = (1 - R)^{A_3}$
Fatigue_Stress_Scale	0.001	Stress unit conversion to Ksi

MIL-HDBK-5J
31 January 2003

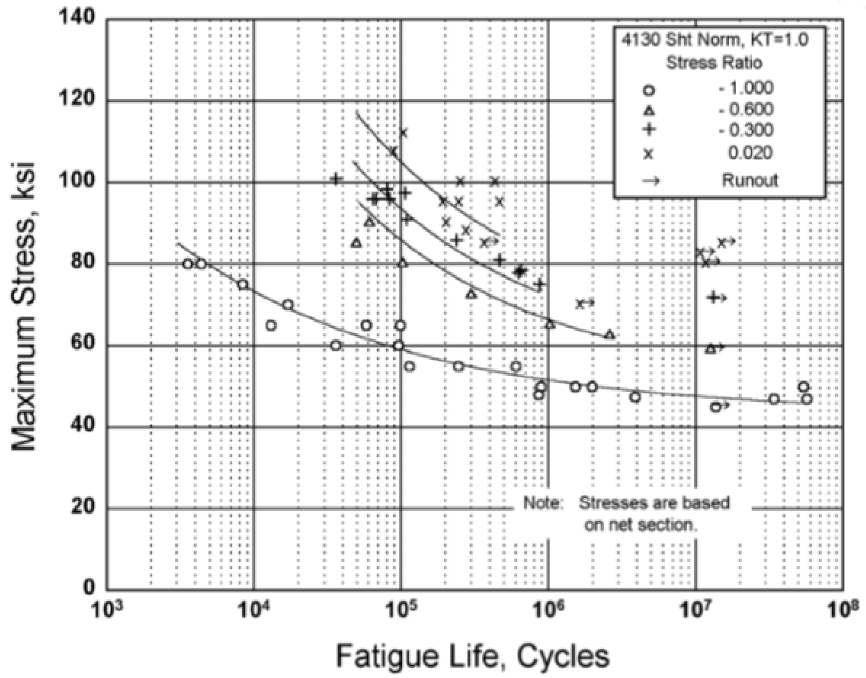


Figure 2.3.1.2.8(a). Best-fit S/N curves for unnotched 4130 alloy steel sheet, normalized, longitudinal direction.

<u>Correlative Information for Figure 2.3.1.2.8(a)</u>

<u>Product Form</u>: Sheet, 0.075 inch thick

<u>Properties</u>:

<u>TUS, ksi</u>	<u>TYS, ksi</u>	<u>Temp., °F</u>
117	99	RT

<u>Specimen Details</u>: Unnotched
2.88-3.00 inches gross width
0.80-1.00 inch net width
12.0 inch net section radius

<u>Surface Condition</u>: Electropolished

<u>References</u>: 3.2.3.1.8(a) and (f)

[Caution: The equivalent stress model may provide unrealistic life predictions for stress ratios beyond those represented above.]

<u>Test Parameters</u>:
Loading - Axial
Frequency - 1100-1800 cpm
Temperature - RT
Environment - Air

<u>No. of Heats/Lots</u>: Not specified

<u>Equivalent Stress Equations</u>:

<u>For stress ratios of -0.60 to +0.02</u>
$\text{Log } N_f = 9.65\text{-}2.85 \log (S_{eq} - 61.3)$
$S_{eq} = S_{max} (1\text{-}R)^{0.41}$
Std. Error of Estimate, Log (Life) = 0.21
Standard Deviation, Log (Life) = 0.45
$R^2 = 78\%$

<u>Sample Size</u> = 23

<u>For a stress ratio of -1.0</u>
$\text{Log } N_f = 9.27\text{-}3.57 \log (S_{max}\text{-}43.3)$

Figure 6-1. – S-N Curve for Steel Sheet.[20] Note that material parameters depend on the unit system.

6.5.6. *Thermal and temperature-dependent properties*

The two key parameters in a material definition that affect thermal loading are the reference temperature, T_{ref} = `Tref`, and the coefficient of thermal expansion (CTE), α_t = `alphat`. These parameters are involved in the calculation of thermal strains, that is,

$$\epsilon_{thermal} = \alpha_t \left(T_{\text{current}} - T_{\text{ref}}\right) . \tag{6.5.2}$$

A typical linear elastic material with constant CTE of 0.001 and a reference temperature of 300.0 can be defined by

```
MATERIAL 1
  E 10e6
  nu 0.3
  Tref 300.0
  alphat .001
  density 0.1
END
```

The default values for `Tref` and `alphat` are both 0.0. If `alphat` is not specified, the material will not undergo any thermal strain.

Currently, isotropic thermal strain is supported only for isotropic, isotropic viscoelastic, and anisotropic materials.

Shell and beam elements are not supported in thermal strains. If a material with a coefficient of thermal expansion is used in a shell, beam, or other unsupported element, **Sierra/SD** generates an error.

Material properties in **Sierra/SD** can also be specified to depend on temperature. Temperature-dependent material properties are supported when temperatures are

(i) read in from an **Exodus** file, or

(ii) specified on a block-by-block basis.

In the case of **Exodus** temperatures, material properties vary from element to element, and the values of the temperature-dependent material properties are calculated from an element average of the **Exodus** temperatures. When temperatures are specified block-by-block, the temperature dependence of the material properties can be specified explicitly in the input deck.

If temperatures are specified in the **Exodus** file *and* block-by-block in the input deck, the input deck values take precedence.

Currently, the temperature-dependent material properties only affect the computation for solid elements and hex shells.

For linear elastic materials, an example of specifying temperature dependent properties is given below.

```
MATERIAL 1
  E function=eTempFunc1
  alphat .001
  tref 100
  nu 0.0
  density 7700.0
END

MATERIAL 2
  E function=eTempFunc2
  alphat .001
  tref 100
  nu 0.0
  density 7700.0
END

MATERIAL 3
  E function=eTempFunc1
  alphat function = alphaFunc1
  tref 100
  nu 0.0
  density 7700.0
END

FUNCTION eTempFunc1
  type LINEAR
  data 0.0 4.0
  data 5.0e9  4.0
END

FUNCTION eTempFunc2
  type LINEAR
  data 0.0 3.0
  data 5.0e9  3.0
END

FUNCTION alphaFunc1
```

```
  type LINEAR
  data 100.0 0.001
  data 5.0e9 0.001
END
```

In this case, the elastic modulus of material 1 is specified by function eTempFunc1, the elastic modulus of material 2 is specified by function eTempFunc2, and in material 3, the elastic modulus is specified by function eTempFunc1, while the CTE is specified by alphaFunc1. The properties of each element will be determined from its temperature relative to the reference temperature, T_{ref}, and an interpolation using the function specified. In this example, the functions are trivial, and thus the moduli of materials 1 and 2 will be 4.0 and 3.0, respectively, while the CTE is 0.001. Note that the moduli, density, CTE, and any of the 4 elastic constants k, g, e, ν can be specified as temperature-dependent, and can be specified by different functions. In the above example, the Poisson's ratios are constant and only the elastic moduli and CTEs are temperature-dependent.

For viscoelastic 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 6.3 on viscoelastic materials for more details.

The **thermal_load** solution case keywords **thermal_time_step** and **nUpdateTemperature** can be used to control the time and frequency at which temperatures are updated from the **Exodus** file. In the case of temperature-dependent properties, **nUpdateDynamicMatrices** can also be used to update the material stiffness based on the temperature read from the **Exodus** file.

For discussion of these and many more important details about temperature input and thermal loads in **Sierra/SD**, consult Section 8.3.7.

If the **thermal_time_step** keyword is used, the temperature that affects material properties will be the temperature read from the provided time step.

Alternatively the **nUpdateDynamicMatrices** keyword can be given to update the material stiffness based on the last-read temperature. The last-read temperature is controlled by the **nUpdateTemperature** keyword. Updating the dynamic matrices is computationally expensive and should be done only when temperature has changed significantly.

6.5.7. *Spatially Variant Material Properties*

Multiple methods exist to define element-to-element spatial dependence in material properties.

Some material properties in **Sierra/SD** can be read on an element-by-element basis from the **Exodus** mesh file. The syntax for this is `parameter=exo_var scalar <string>`. Here `<string>` is the provided exodus field name. This must be a scalar field (one component) defined on each element using the material.

For linear elastic materials, an example of specifying **Exodus**-based properties is given below.

```
MATERIAL 1
  E = exo_var scalar e_input
  nu = exo_var scalar my_Poisson
  density = exo_var_scalar elem_density
END
```

In this case, the elastic modulus of material 1 is specified by the exodus mesh field ’e_input’, the Poisson’s ratio by ’my_Poisson’, and the density by ’elem_density’. The exodus based properties can be used to define complex spatial dependence of material properties as may be found in partially compressed foams for example. No time-variance of material properties is considered, if the input mesh exodus file has multiple time steps the material property fields should be constant over time to avoid confusion.

Additionally, it is possible to define material properties through the general `function` syntax. Syntax and requirements are detailed in table 4-24. Unless otherwise specified material property functions are evaluated as a function of temperature. The temperature can be defined via a block temperature in the input deck, an element-by-element temperature in the exodus input mesh, or nodal or Gauss point temperatures (which are then interpolated to the element centroid temperatures) in the exodus input mesh.

```
MATERIAL based_on_function
  E = function e_temperature_function
  nu = 0.3
  density = 1
END

FUNCTION e_temperature_function
  type linear
  data 0     1.0e+6
  data 400   1.0e+6
  data 500   0.9e+6
  data 900   0.3e+6
END
```

Alternatively specific input variables other than temperature can be explicitly defined in analytic functions. Nodal function responses are mapped to element material properties by evaluating the function at the nodes, and applying element shape functions to interpolate those values to the centroid.

```
MATERIAL reads_from_nodal
  E = function e_input_function
  nu = 0.3
  density = 1
```

```
END

FUNCTION e_input_function
  type analytic
  expression variable X = nodal some_mesh_var
  expression variable Y = nodal another_mesh_var
  evaluate expression "sqrt(X) + Y"
END
```

6.5.8. Specific Heat

Conversion of energy deposited in a structure to a change in temperature may be effected by a specific heat.

$$Q = \rho V\, C\, \Delta T. \tag{6.5.3}$$

Here 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 6.5.3. Once converted to temperatures, thermal stresses and temperature dependent material properties may be applied. A fatal error is encountered if the specific heat is not specified for each material containing an energy load. The keyword `defaultSpecificHeat` defined in the "parameters" section, can be used to specify a default specific heat for all materials.

```
MATERIAL 'Steel-SI'
  E=2e11              // Pa
  NU=0.28
  density=7850          // kg/m^3
  specific heat = 0.45  // J/(g K)
  tref = 300 // K
  alphat = 0.001
END
```

The reference temperature, T_{ref}, is used for temperature-dependent material properties, such as in viscoelastic materials:

$$\Delta T = \frac{Q}{\rho V C} \tag{6.5.4}$$

$$T_{\text{elem}} = T_{\text{ref}} + \Delta T \tag{6.5.5}$$

$$\epsilon_{thermal} = \alpha_T (T_{\text{elem}} - T_{\text{ref}}). \tag{6.5.6}$$

Energy loads use the energy per unit mass or specific energy,

$$\tilde{E} = \frac{Q}{\rho V},$$

as described in Section 8.3.8.

6.5.9. *Frequency dependence*

For the `CJdamp` solution method (see Section 5.2), a frequency dependent damping coefficient, $\eta(f)$, may be specified. [1] 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 4.10 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=1e7
  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 and amplitude pairs for η. The frequencies are in Hertz. The `CJdamp` solution process interpolates the function at the eigenvalues to determine the effective damping for any particular mode. **Piezoelectric Material Sierra/SD** supports two material models which possess voltage degrees of freedom in addition to displacements and rotations: dielectric and piezoelectric materials. The piezoelectric material is characterized by an electro-mechanical coupling in the stiffness matrix. A piezoelectric material is defined by three constitutive tensors: a rank four orthotropic elasticity tensor, a rank two anisotropic permittivity tensor, and a rank three piezoelectric coupling tensor. See theory manual for further details on the constitutive tensors. Piezoelectric material tensors may be specified in a material block by including the keyword **orthotropic_piezoelectric** followed by the required three material tensor specifications. The nine parameters defining an orthotropic elasticity tensor are given by the keyword **Cij** followed by the upper triangle of a six by six matrix. The piezoelectric coupling

[1] η is twice the normal modal damping coefficient. Thus, if eta=0.02 for all materials, the equivalent modal damping will be 1 percent.

tensor is given by the keyword `e_ij` followed by a six by three coupling matrix. **Sierra/SD** assumes the coupling matrix is in its stress-charge form (units $charge/Area$). The anisotropic permittivity tensor is provided by the keyword **permittivity_ij** followed by a three by three matrix. The permittivity matrix should be populated by absolute permittivity values (not normalized by the permittivity of free space).

Here is an example of a PZT5A piezoelectric material where *e0* is the permittivity of free space:

```
MATERIAL 1
  orthotropic_piezoelectriC
    Cij = 12.1e10  7.5e10   7.5e10
                   12.1e10  7.5e10
                            1.1e11
                            2.1e10
                            2.1e10
                            2.3e10
   permittivity_ij = 916 * e0    0        0
                       0      916 * e_0   0
                       0           0   830 * e0
   e_ij = 0     0  -5.4
          0     0  -5.4
          0     0  15.8
          0   12.3   0
        12.3   0     0
          0     0    0
   density = 7.75e3
END
```

Input 6.6. Piezoelectric Material

Dielectric Material A dielectric material, the second available material possessing a voltage degree of freedom, is used to model the electrostatic behavior of materials that do not exhibit electro-mechanical coupling (i.e., non-piezoelectric). Dielectrics can be generated with the keyword **dielectric**, and are defined with only its permittivity tensor. The following is an example of a dielectric input example.

```
MATERIAL 1
   DIELECTRIC
   permittivity_ij = 916 * e0    0        0
                       0      916 * e_0   0
                       0           0   830 * e0
END
```

Input 6.7. Dielectric Material

6.6. Block

```
BLOCK 32
  material 2
  tria3
  thickness 0.01
END
BLOCK block_3
  coordinate 1
  spring
  Kx=1e6
  Ky=0
  Kz=0
  BlkBeta=0.0031
END
```

Input 6.8. Numbered Blocks

```
BLOCK aft_cover
  material aluminium
END
MATERIAL aluminium
  E = 7e10
  nu = 0.35
  density = 2700
END
```

Input 6.9. Labeled Block

Each element block in the **Exodus** file must have a corresponding `block` section in the input file. The converse 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 7.23) 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. The behavior of **Sierra/SD** in this case is not defined. This section contains information about the properties of the elements within the block.

6.6.1. Block Parameters

Parameters are either for specific elements or generic. An element block must specify its material. The material reference is of the form, `material=material_id`, where `material_id` is a string representing the material identifier (see Section 6). The optional parameters for elements are coordinate frames, nonlinear behavior, block damping, and non-structural mass,

Finite element configuration beyond the coordinates is set manually in either the Exodus mesh file (element attributes) or in the input deck (element parameters). The types of parameters depend on the category of element. Table 6-4 summarizes the options for 5 categories of elements.

Explanations and definitions are described in the corresponding sections of chapter 7. Parameters apply to all the elements in the block. Attributes are specified for each element in the element block; in this sense attributes vary in space.

Currently, four groups of elements have parameters. Shells have membrane and bending factor factors 7.9.5. The Infinite Element 8.1.9 and Perfectly Matched Layer 8.1.10 are configured using several unique block parameters. And the Beam2 7.11 and Nbeam 7.12 both have several parameters.

Block labels can be provided in three forms. As shown in the example input 6.8 an integer number can be given; the number refers to the **Exodus** block id number in the mesh. Alternatively this number may be provided as 'Block_##' which is compatible with **Sierra/SM** syntax. The third form is a block name, such as 'aft_cover' in example input 6.9. This is the name of the block in the **Exodus** input mesh. Note that the material ID specified for Block 32 uses an index (material 2), whereas Block aft_cover uses a specified material ID string aluminium. These refer to materials defined by blocks Material 32 and Material aluminium respectively (see Sec. 6 for details).

Parameters that are generally applicable to almost all blocks are listed in Table 6-5. More detailed descriptions are available in the following paragraphs.

Table 6-4. – Element Parameters.

Element Type		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	I1	First bending moment
	3	I2	Second bending moment
	4	J	Torsion moment
	5,6,7	Orientation	orientation vector. For the orthogonal direction
	8,9,10	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	fiber orientation (theta)	fiber orientation
	3	offset	shell offset in normal direction
Quad	1	thickness	thickness
	2	fiber orientation (theta)	fiber orientation
	3	offset	shell offset in normal direction

Table 6-5. – Parameters for (almost) Any Block.

Keyword	Values	Description
nonlinear	yes/no	blockwise nonlinear behavior
material	*string*	material identifier
rotational_type	Eulerian or Lagrangian or none	blockwise behavior for rotational dynamics terms
coordinate	*string*	reference coordinate frame
blkalpha	*Real*	blockwise mass proportional damping
blkbeta	*Real*	blockwise stiffness proportional damping
nsm	*Real*	blockwise non-structural mass
density_scale_factor	*Real*	blockwise density scaling factor
stiffness_scale_factor	*Real*	blockwise stiffness scaling factor
T_current	*Real*	Temperature of every element in the block (default 0.0)
geom_stiff	yes/no	include/exclude("no") geometric stiffness See section 5.23

6.6.1.1. 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 (4.4). 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 33
  nonlinear=no
  material 2
  tria3
  thickness 0.01
END
```

Similarly, to turn on the nonlinear behavior for the block, we would have

```
BLOCK 34
  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 (4.4).

Some elements types are incapable of nonlinear behavior. This includes RBE3s, Rbars, and Rrods. By default, use of these elements in nonlinear analysis will generate a fatal error. Use of the command `nonlinear=no` in these element blocks will enable overriding this fatal error and use pure linear behavior for these elements in the nonlinear analysis. Care should be taken with this option, pure linear elements have some incompatibilities in nonlinear analysis. For example use of pure-linear elements in nonlinear analysis can artificially constraint large rotations.

> Linear element behavior in a nonlinear solution is limited to the linear range of the element. Nonlinear transient and statics simulations store rotations incrementally. Also, elements use Corotational formulations if available. The Corotational formulations accurately model large deformations as long as strains remain small. However, rotations must be less than 360°. For simulations involving large strains or rotations greater than 360°, **Sierra/SM** is recommended.

> Nonlinear statics and transient were barely documents experimental capabilities that recently became production ready.

6.6.1.2. 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.

6.6.1.3. 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. Further information on coordinate systems may be found in Section 4.9.

Alternatively, input/transfer element fields may be used to define the axes of a local coordinate system using the `from_transfer` keyword, as in the example below.

```
TRANSFER mesh_with_coords.exo
  destination blocks = all

  # x-axis (fields are "x-axis_x", "x-axis_y", "x-axis_z")
  initialize variable name = material_direction_1
    read variable = x-axis_
    variable type = element

  # y-axis (fields are "yx", "yy", "yz") -- (optional)
  initialize variable name = material_direction_2
    read variable = y
    variable type = element

  # z-axis (fields are "material_direction_3_x",
  #                    "material_direction_3_y",
  #                    "material_direction_3_z")
  initialize variable name = material_direction_3
    read variable = material_direction_3_
    variable type = element
END

block all
  material aluminum
  coordinate from_transfer
```

```
END
```

Finally, the intrinsic geometry may be used to define the local coordinate system using the `from_geometry` keyword. If this option is chosen, a default coordinate system will be created dependent on element type. For 1D elements, the x-axis is defined along the line connecting the 2 nodes, and the y-axis will default to the global y-axis or the `orientation` vector if it is defined. In the event that these are incompatible, an alternative, orthogonal y-axis will be determined and used. For zero-length elements, this will default to the global coordinate frame. An example input section is shown below.

```
block all
  material aluminum
  coordinate from_geometry
END
```

6.6.1.4. Block Specific Damping

In Section 6.7, 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.

$$D = \sum_{i}^{nblks} \alpha_i M_i + \beta_i K_i \tag{6.6.1}$$

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 level coefficients described in Section 6.7. 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 would couple modal based solutions. It is not currently available in modal solutions such as **modaltransient**. Also, see Section 6.5.9 for material modal like damping.

6.6.1.5. Non-Structural Mass

Non-structural mass (NSM) is specified per element block in the input deck. It is added to the internal mass of the element. One reason to add a NSM is to use a gravity load to simulate an external load. Another reason is to stabilize solutions with mass-less nodes. The units depend on the element type as defined in Table 6-6.

6.6.1.6. Non-Structural Mass Corner Cases

As mesh topology determines the dimension, a Hexshell element is considered three-dimensional. Layered shell elements add non-structural mass once per meshed element, not once per layer. The conditions shown in Table 6-7 cause non-structural mass to be silently ignored and trigger no warnings.

The following is an example of how to use non-structural mass in the input file:

```
//nsm specified in pounds per square inch
BLOCK 35
  material 2
  tria3
  thickness 0.01
  nsm 0.005
END

MATERIAL 2
  density 0.5
END
```

Table 6-6. – Non-Structural Mass Units.

Element Type	Units	Example
ConMass	Mass Per Element	lbs
Spring or Joint2G	Mass Per Element	lbs
Beam or Truss	mass/length	lbs / in
Two Dimensional	mass/area	lbs / sq-in
Three Dimensional	mass/volume	lbs / cu-in

Table 6-7. – Unhandled Corner Cases.

Element Type	State	Result
Beam or Truss	Area=0	NSM Silently Ignored
Two Dimensional	Thickness=0	NSM Silently Ignored

6.6.1.7. Blockwise Density Scaling

An element block may define a scale factor to be applied to the density of the material. This can be used to calibrate the exact mass in each block and account for discretization errors, or to reuse materials that only differ in density. The interaction of this feature with non-structural mass is documented in Table 6-8.

The following is an example of blockwise density scaling in the input file:

```
BLOCK 36
  material 2
  density_scale_factor = 1.0025
END

MATERIAL 2
  density 0.5
END
```

Table 6-8. – Combining NSM with Density_Scale_Factor.

Element Type	Mass Per Element
ConMass	NSM + Mass
Spring or Joint2G	NSM
Beam or Truss	NSM*Length + Density*Scale*Volume
Two Dimensional	NSM*Area + Density*Scale*Volume
Three Dimensional	NSM*Volume + Density*Scale*Volume

6.6.1.8. Blockwise Stiffness Scaling

An element block may define a scale factor to be applied to the linear stiffness of the material. This can be used to tune the precise stiffness of components without requiring separate material definitions, usually when adapting a model to match test results. The `stiffness_scale_factor` is applied to isotropic materials consistently, even if the Young's modulus is not defined explicitly by the user. Any valid combination of material constants will still be valid.

The `stiffness_scale_factor` command is limited to *isotropic material models only*.

- It is ignored on blocks without a material, such as spring elements.
- Attempting to use `stiffness_scale_factor` with any anisotropic material will result in an error.

The following is an example of blockwise stiffness scaling in the input file:

```
BLOCK 38
  material 2
  stiffness_scale_factor = 1.0025
END

MATERIAL 2
  isotropic
  E 1e7
  nu 0.3
  density 0.5
END
```

6.6.1.9. Geometric Stiffness Consideration

This option can be used to include (`geom_stiff=yes`) or ignore (`geom_stiff=no`) the preloaded stress contribution to the geometric stiffness matrix when transferring data (section 5.23). See section 5.23.1 and the **no_geom_stiff** keyword of the **receive_sierra_data** solution case (section 5.23) for more details.

Piezoelectric Material Damping Only stiffness (`blkbeta`) and mass (`blkalpha`) proportional damping can be applied to electro-mechanical materials, and damping models can only be specified at the block level. Global damping is prohibited on any model containing piezoelectric or dielectric materials. Voltage degrees of freedom do not couple with mass or damping (see Theory Manual). Hence, a piezoelectric element's damping matrix, defined by stiffness and(or) mass proportional damping, is zero at all voltage degrees of freedom. The effects of mechanical damping will only impact the voltage degree of freedom responses due to the electro-mechanical stiffness coupling.

6.7. Damping

The Damping section of an input deck is the interface to the two global viscous damping models. Either modal damping rates or stiffness and mass proportional damping may be applied. The options tabulated in 6-9 are described in the following sections.

Table 6-9. – DAMPING Section Options.

Parameter	Description
alpha	mass proportional damping parameter (real)
beta	stiffness proportional damping parameter (real)
gamma	uniform modal damping ratio (fraction of critical) applied to all modes (real)
mode	ADDITIONAL modal damping ratio applied to individual mode(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 6-9. 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.

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

$$\zeta_i = \alpha/(2\omega_i) + \beta \cdot \omega_i/2 + \Gamma + \text{mode}(i) + \text{ratiofun}(i),$$

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   // gamma+mode_1 = 1.5% of critical
```

[2]Use of block specific proportional damping is explained in Section 6.6.1.

```
  mode  2  0.005  // gamma+mode_2 = 1.0% of critical
  mode  3  0.015  // gamma+mode_3 = 2.0% of 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[3] from the previous modal solution. 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 4.10) 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 subsection Damping of Flexible Modes Only section Solution Procedures of the Theory Manual.

In order for this method of damping to work properly, the structure must have the conventional six rigid body modes of three translations and three rotations. A check of this condition is made

[3] i.e. non-modal based, but linear transient

[4] A previous modal solution case must have been specified to use modal damping, otherwise **Sierra/SD** will warn the user and abort.

inside of **Sierra/SD**, and a fatal error results if this condition is not satisfied. Specifically, the condition is met if

$$RatioFlexibleRbm = \frac{\|K\Phi_r\|_2}{\|K_d\|_\infty \|\Phi_r\|_2} \leq \epsilon \tag{6.7.1}$$

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
```

The FilterRbm option is compatible with the default Newmark-Beta time integration. If the generalized-alpha time integration is used, then the parameter rho should be set to 0.5. For additional details see the "Damping of Flexible Modes Only" section of the Sierra/SD Theory Manual.

6.7.1. *Nonlinear transient*

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 is not used. This reason is that the tangent matrix would be required to compute the damping terms at each iteration.

Nonlinear transient does not support 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 (6.7.2). Like modal damping, this is a system level damping model.

6.7.2. *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.[45] 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]

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 "eigen" methods.

The damping section is used to define the type of damping behavior. Currently, only two types of damping behavior are defined: a damper with optional cubic parameters, **damper**, and an Iwan model, **Iwan**.[45] 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 'eig'
    eigen
      nmodes 16
      shift -1e5
  case 'nonlinear'
    NlTransient
      nsteps = 200
      time_step = 5.0e-3
      rho = 0.8
END
```

Input 6.10. Modally Damped Nonlinear Transient

[2] Masing and Iwan models are used almost interchangeably in this document. Iwan models are a subset of more general Masing models.

```
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
```

Input 6.11. Nonlinear Damping

```
PROPERTY 1
  Mu = 0.001
  K = 0
END

PROPERTY 2
  Mu = 0.02
  K = 0
  Mu3 = 1e-5
  K3 = 1e-5
END

PROPERTY 3
  chi = -0.82
  phi_max = 1.0e-4
  R = 7.6e6
  S = 5.6e6
END

PROPERTY 4
  chi = -0.80
  phi_max = 1.1e-4
  R = 7.6e4
  S = 5.6e5
END
```

Input 6.12. Nonlinear Damping: Corresponding Properties

6.7.3. Frequency Band Damping

Traditional damping models such as mass and stiffness proportional damping apply different effective damping at different frequencies.. In contrast, frequency band damping enables a nearly-uniform damping coefficient across a frequency range of interest.

Frequency Band Damping in **Sierra/SD** follows the approach of Huang et. al.,[27] and may be enabled using the following syntax for the direct transient solution case. Frequency band damping uses a quasi viscous-material approach applied at the element level.

```
FREQUENCY BAND DAMPING
  low frequency = <real>
  high frequency = <real>
  gamma = <real, ge 0, lt 1>
  block = <list(block)>
  num filters = <int, ge 2>(5)
```

```
END
```

Frequency band damping is only enabled/tested for transient and modal-transient solutions.

The following commands are used by frequency band damping.

- **low frequency** and **high frequency** define the upper and lower frequency for application of the flat damping region. Flat and maximal damping is applied in this range. However substantial damping still occurs outside of the defined range. For example if the damping was 5% between a low and high frequnecy of 100 Hz to 1000 Hz there would also be ~2% damping out to 10Hz and 10,000Hz and ~0.5% damping out to 1 Hz and 100,000 Hz.
- **gamma** is the fraction of critical damping to apply. It has the same interpretation as the gamma coefficient in modal gamma damping.
- **block** keyword may be used to restrict damping to a subregion of the model. For example, `block = 1:100` would damp only blocks between 1 and 100, `block = foo,bar` would damp blocks "foo" and "bar", and `block = all remove 200:300` would damp all blocks *except* blocks in the 200-300 range.
- **num filters** sets the number of distinct viscous damping mechanisms. A larger number of mechanisms leads to a flatter damping over the range, but at a increased computational cost. For most problems the default value of 5 provides sufficient accuracy.

Multiple frequency band definitions may be used to target different frequency bands on different regions of the model.

Additional usage guidelines:

- Frequency band damping will introduce artificial reduction of natural frequencies. This effect will get higher at larger damping ratios. A 5% damping will introduce roughly a 5% artifical freqency shift.
- Application of excessively wide frequency bands can cause solver difficulties. In order to apply flat damping over the full range set a lower frequency somewhat below the first flexible mode and a high frequency somewhat above the maximum frequency of interest.
- Frequency band damping applies substantial damping outside of the low and high range. It will typically be unfeasible to damp the same regions of the model at different gammas at different frequency ranges.

Example:

```
low frequency = 100   # Low frequency for flat damping
high frequency = 1000 # High Frequency for flat damping
gamma = 0.02          # Apply 2% of critical damping
block = 1             # Apply damping just to block 1
```

This page intentionally left blank.

7. ELEMENT LIBRARY

Sandia Labs has adopted the **Exodus** format for finite element mesh storage. In **Exodus** format collections of elements are stored in element blocks. For this reason, generic information about finite elements is described in Section 6.6. This section reviews the specific features of the elements and pseudo-elements.

Element configuration through both the Exodus mesh file (element attributes) and in the input deck (element parameters) is supported. The input deck parameters take precedence over the element attributes. Recall that the parameters are summarized in Table 6-4. Parameters apply to all the elements in the block. Attributes are specified for each element in the element block; in this sense attributes vary in space.

Some elements have **Exodus** attributes. Such attributes may be specified either in the **Exodus** file per element or the input deck per block or both. Input deck values override **Exodus** file values. Attributes are described in this section. An attribute is either required or optional. The minimum element diameter is 10^{-10}. If an element has diameter less than the minimum, then the behavior Sierra is undefined.

7.1. Hex8

The `Hex8` is a standard 8 node hexahedron with three degrees of freedom per node. It has 8 integration points, and trilinear shape functions. Isotropic and anisotropic materials are supported.

There are three variations of `Hex8`. The default element is a bubble hex element. It is specified by **Hex8b**. From a user's perspective the Hex8b and the Hex8 are indistinguishable; both use 8 nodes with 3 degrees of freedom per node.

The `Hex8b` element uses bubble functions[64,30,35] to augment the standard element shape functions. It bends more accurately than the Hex8.

The **Hex8u** specifies a selective deviatoric hex. By default, this element uses full-integration of the deviatoric strain terms and single-point under-integration of the pressure term. The **sd_factor** can be specified for this element which controls how the deviatoric terms are integrated. A **sd_factor** value of 1.0 (the default) corresponds to full integration of deviatoric terms. A **sd_factor** value of 0.0 would make the element behaves like an uniform gradient hex with no hourglass control. Values between 0.0 and 1.0 are also allowed for the **sd_factor** which correspond to a mixture between these two states. More information is given in the selective Integration section **Sierra/SD** Elements of the Theory Manual.

```
BLOCK
  # Standard selective deviatoric formulation
  Hex8u
  material 1
END
```

```
BLOCK
  # Under-integrated formulation for both pressure and deviatoric terms
  Hex8u
  material 1
  sd_factor 0.0
END
```

The fully integrated Hex is specified by **Hex8F**. While it performs adequately when the element shape is nearly cubic, it performs poorly for larger aspect ratios. For most problems involving bending the Hex8b is recommended.

The only *required* parameter for these elements is the material specification. Any material may be applied.

For computational acoustics, see Section 4.6.4.

7.2. 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 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 9-15, and in subsubsection Shape Functions and Gauss Points subsection Quadratic Isoparametric Solid Elements section **Sierra/SD** Elements of the Theory Manual.

The stress may be output at the Gauss points as described in Section 9.7.20.6.

7.3. 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.

7.4. 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 element is recommended for use in most structural analyses.

A material specification is required, and any structural material may be used.

7.5. 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.

7.6. Wedge15

The Wedge15 element adds mid-side nodes to the Wedge6. Like the Hex20 and Tet10, it has quadratic shape functions, and is recommended. A material specification is required, and any structural material may be used.

7.7. Pyramid5

The Pyramid5 is a transition element between **hex8** and **tet4** blocks. They are generated automatically by meshing programs like cubit. It is not recommended that the entire mesh be built of **Pyramid5** elements. A material specification is required, and any structural material may be used.

Pyramid elements are not compatible with all **Sierra/SD** features. Known limitations are listed below.

- They do not implement a condition number (section 9.8.3). This is because our condition numbers are derived from Verdict,[62] which does not implement a pyramid condition number.
- They are not currently compatible with mesh transfer (section 4.3.2), although it could readily be extended to support them.

7.8. Pyramid13

The Pyramid13 element adds mid-side nodes to the Pyramid5, to transition between **hex20** and **tet10** blocks. Like the Hex20, Tet10, and Wedge15, it has quadratic shape functions, and is recommended. A material specification is required, and any structural material may be used.

7.9. Two-Dimensional Shell and Membrane Elements

Sierra/SD supports a variety of topologically 2D elements that capture shell or membrane behavior. A specific 2D element formulation can be selected by explicitly specifying the element type in the input block with keywords such as `QuadT`, `QuadM`, `Nquad`, etc. For three noded triangles if no specific element formulation is given then by default one of the Tria3 (isotropic materials) or TriaShell (orthotropic or layered materials) shell formulations will be used. For all other topologies (4-node quad, 8-node quad, and 6-node triangle) the default element is composed of sub-triangles using the Tria3 or TriaShell shell formulations as described in Section 7.9.1

7.9.1. QuadT, Quad8T, and Tria6

The 4-node quad `QuadT`, 8-node quad `Quad8T`, and 6-node triangle `Tria6` are all internally composed of sub-triangles. Each element is the **Sierra/SD** default formulation for its respective topology. These elements have both membrane and bending stiffness. The element stiffness and mass matrices are derived by composing internally generated triangle elements, as illustrated in Figure 7-1, 7-2, and 7-3. Output quantities such as stress are the average stress over the sub-triangles. Since these elements are made up of multiple triangles, pressure force application and stiffness across a single elements are not uniform. However, the stiffness and forces are balanced in such a way to produce expected displacement and acceleration behavior. Though not optimal, composing elements from low-order triangles is adequate for most applications.

The sub-triangles 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. More complex materials require use of the **TriaShell**. The underlying triangle formulation is determined automatically by **Sierra/SD**, and cannot be selected by the user. See the description of the `Tria3` and `TriaShell` for details of the formulations of the triangle elements that compose the `QuadT`, `Quad8T`, and `Tria6` elements.

Table 7-1 lists the supported inputs.

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
```

Figure 7-1. – QuadT Element.
The element is generated by internally combining two Triangle elements.

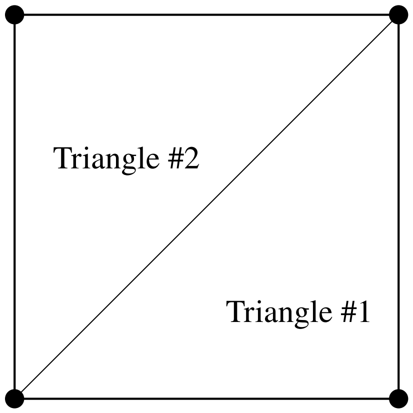


Figure 7-2. – Quad8T Element.

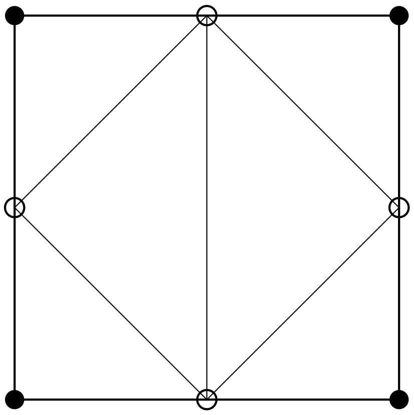
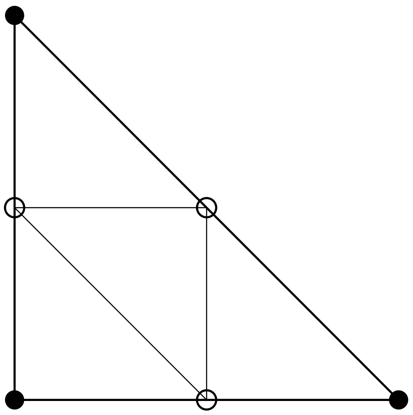


Figure 7-3. – Tria6 Element.



Keyword	Description
thickness	Thickness of the shell
offset	offset of the shell midplane: see Section 7.9.8
material	Linear elastic material used by element
layer	Layer properties: see Section 7.9.8
coordinate	Base global coordinate system: see Section 7.9.7
rotate about axis	Coordinate system rotation: see Section 7.9.7
rotate about normal	Coordinate system rotation: see Section 7.9.7
membrane_factor	Stiffness scale factor: see Section 7.9.6
bending_factor	Stiffness scale factor: see Section 7.9.6

Table 7-1. – QuadT, Quad8T, Tria6 Inputs.

7.9.2. QuadM

QuadM is a 4-node quadrilateral membrane element. It has membrane stiffness but no rotational degrees of freedom (DOFs). Membranes are well suited to structures with very low bending stiffness, such as fabric. Use of shell elements for such low-bending-stiffness structures can generate a problematic near-singularity.

In the input deck a `block` section indicating **QuadM** is required.

For two-dimensional problems, the **QuadM** 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. Preloading creates the out-of-plane stiffness. An unloaded element has no out-of-plane stiffness and may be singular. The out-of-plane behavior results from an additional stiffness term that is applied to the out-of-plane DOFs. The stiffness resembles the stiffness associated with Laplace's equation. This additional stiffness is derived in classical textbooks.[32]

Table 7-2 lists the supported inputs.

Keyword	Description
thickness	Thickness of the membrane, required
sd_factor	Selective deviatoric parameter used for numerical integration
material	Linear elastic material used by element, required
coordinate	Base global coordinate system: see Section 7.9.7
rotate about axis	Coordinate system rotation: see Section 7.9.7
rotate about normal	Coordinate system rotation: see Section 7.9.7
membrane_factor	Stiffness scale factor: see Section 7.9.6

Table 7-2. – QuadM inputs.

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, for full integrated membrane, one would specify

```
BLOCK 11
  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 12
  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 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 preloaded before the analysis of interest (e.g., a static preload followed by eigendecomposition), or even in cases where no preload is applied but the membranes are sufficiently constrained (such as a hex element with a layer of membrane elements on the surface).

The QuadM element can be used in coupled simulations. In these cases, the **Sierra/SM** simulation result is input to **Sierra/SD** for later analysis. These preloaded elements are non-singular.

The membrane element does not currently compute stress, strain, strain energy, or strain energy density outputs. All these outputs will be reported as zero for the element.

7.9.3. Nquad/Ntria

The `Nquad` and `Ntria` elements are isoparametric shells with membrane and bending stiffness. They are shear-deformable elements with six DOFs per node which support isotropic, orthotropic, and layered materials. 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[40] (but not in later editions). The layered shell stiffness matrix uses a composite laminate formulation.[37]

In the input deck a block definition indicating either Nquad or Ntria is required. The block definition must also have a material keyword referencing the isotropic material properties (Section 6) or orthotropic layer properties (Section 6.1.2) 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 14
  Nquad
  thickness 0.1
  material 2
END

BLOCK 15
  Ntria
  thickness 0.4
  material 4
END
```

Inputs are given in Table 7-3.

Keyword	Description
thickness	Thickness of a single layer shell
material	Linear elastic material used by single layer element
layer	Layer properties: see Section 7.9.8
Nquad_eps_max	Numerical integration parameter
coordinate	Base global coordinate system: see Section 7.9.7
rotate about axis	Coordinate system rotation: see Section 7.9.7
rotate about normal	Coordinate system rotation: see Section 7.9.7

Table 7-3. – Nquad/Ntri inputs.

The stabilization method from Belytschko[9] 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 that is a linear combination of the reduced integration and full integration shear stiffness matrices,

$$K_s = (1-\varepsilon)K_s^{[1x1]} + \varepsilon K_s^{[2x2]}.$$

The fraction, $\varepsilon = rt^2/A$ is a function of thickness and area. Here $r = 0.03$, t is the element thickness and A is the area of the shell. This automatic selection of ε works well for thin plates, but can be a problem for thicker elements; 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 16
  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 7-4 shows this function for Nquad_eps_max = 1.

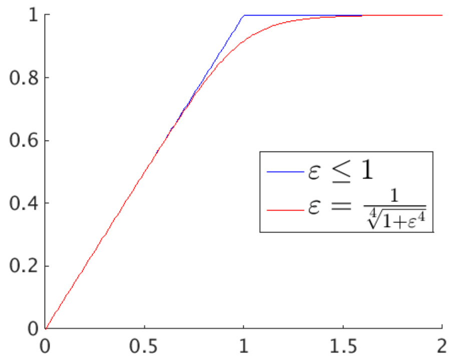


Figure 7-4. – Function for Nquad_eps_max.

The Ntria with orthotropic materials sets G_{23} and G_{13} to the input value of G_{12}.

7.9.4. *TriaShell*

The `TriaShell` is a 3-noded triangular element with 6 DOFs. The formulation of the `TriaShell` is generated by decoupling the membrane DOF and the bending DOF. Allman's Triangular (AT) element[2] models the membrane DOF, while the Discrete Kirchhoff Triangle[8] (DKT) models the bending DOF. These two elements are combined into the `TriaShell` element.

In general, the `Tria3` element is preferred to the `TriaShell` because it is less prone to shear locking behavior, and it is computationally efficient. The TriaShell element is required for orthotropic or layered materials.

- TriaShells support orthotropic or anisotropic materials.
- TriaShells support layered materials. Note however that mass lumping is not allowed with layered TriaShell elements.

Keyword	Description
thickness	Thickness of the shell
offset	Offset of the shell midplane: see Section 7.9.8
material	Linear elastic material used by single layer element
layer	Layer properties: see Section 7.9.8
coordinate	Base global coordinate system: see Section 7.9.7
rotate about axis	Coordinate system rotation: see Section 7.9.7
rotate about normal	Coordinate system rotation: see Section 7.9.7
membrane_factor	Stiffness scale factor: see Section 7.9.6
bending_factor	Stiffness scale factor: see Section 7.9.6

Table 7-4. – TriaShell input options.

7.9.5. Tria3

The `Tria3` is a three-dimensional triangular shell with membrane and bending stiffness. There are 6 DOFs per node. In most respects it is similar to the **TriaShell**. It is the default element for triangular meshes. The **Tria3** was provided by Carlos Felippa of CU Boulder. The element handles isotropic unlayered materials.

Keyword	Description
thickness	Thickness of the shell, required
offset	offset of the shell midplane see Section 7.9.8
material	must be isotropic, required
membrane_factor	Stiffness scale factor, see Section 7.9.6
bending_factor	Stiffness scale factor, see Section 7.9.6

An example element block is shown below.

```
BLOCK 17
  Tria3
  Thickness 0.01
  material 71
  membrane_factor=0  // turns off membrane stiffness
END
```

7.9.6. Stiffness Scaling

The stiffness of the 2D element bending and membrane responses are computed independently and can be independently scaled. Use **membrane_factor** and **bending_factor** to configure the element. Each of these parameters default to 1.0. Reasons for scaling the element stiffness could include accounting for damage to the structure or tuning of model response to match experimental data.

7.9.7. Shell Coordinate Systems

For orthotropic materials a coordinate system must be defined in the element to set the local material orientation. First, a user-defined global coordinate system is referenced with the **coordinate** keyword (Section 4.9.) The global coordinate system is evaluated at the element centroid to define a local $\hat{r}, \hat{s}, \hat{t}$ system for the element.

This $\hat{r}, \hat{s}, \hat{t}$ coordinate system is projected onto the surface of the shell as shown in Figure 7-5. This projection will rotate the coordinate system such that the $\hat{t}_p$ axis aligns with the shell normal and $\hat{r}_p$ and $\hat{s}_p$ are as close as possible to the original orientations of $\hat{r}$ and $\hat{s}$. It is recommended to use a global coordinate system in which $\hat{t}$ lies as close as possible to the shell normal vectors to minimize ambiguity in this coordinate system alignment step.

Two additional inputs are available in the shell block to alter the element local coordinate system. First a rotation can be applied to the $\hat{r}, \hat{s}, \hat{t}$ vectors prior to projection onto the shell plane with the command:

```
rotate <real> about axis <int>
```

The rotation angle is given in degrees. The axis integer 1, 2, or 3, represents the $\hat{r}$, $\hat{s}$, or $\hat{t}$ coordinate axes. After rotation of the coordinate system the $\hat{r}'$, $\hat{s}'$, $\hat{t}'$ system is aligned to the plane of the shell. See Figure 7-6 for an example.

A second input option can rotate the element local coordinate system after projection with the command:

```
rotate <real> about normal
```

The angle is given in degrees. This will rotate the projected system around the $\hat{t}_p$ vector as shown in Figure 7-7.

Once a shell-local system is defined, the $\hat{r}_p$ and $\hat{s}_p$ vectors will define the orientation of orthotropic materials. The **fiber_orientation** command can do one more local rotation of this orientation layer-by-layer as described in Section 7.9.8.

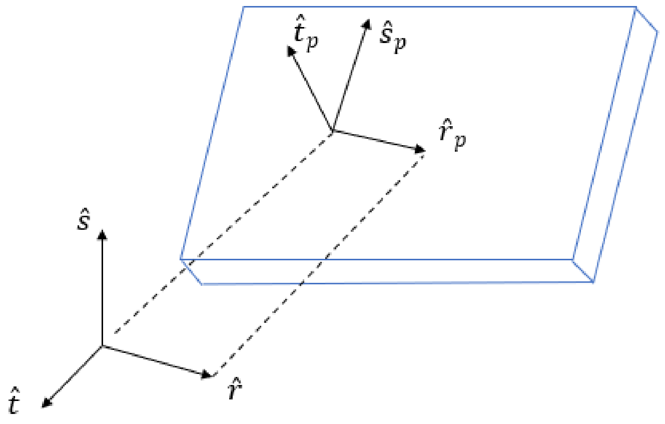


Figure 7-5. – Projection of global coordinate system to shell.

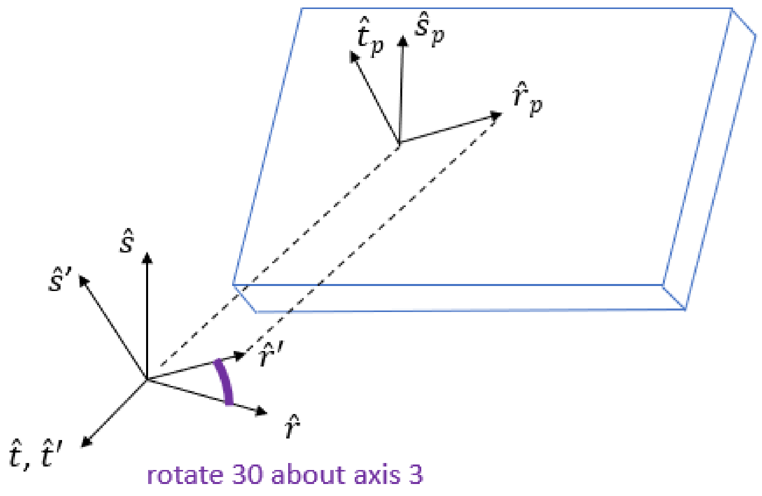


Figure 7-6. – Rotation of global coordinate system about axis prior to projection to shell.

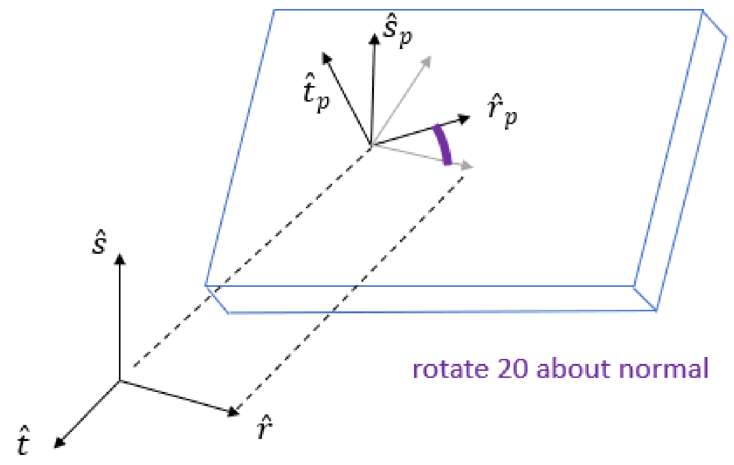


Figure 7-7. – Rotation of local system about normal after to projection to shell.

7.9.8. Layered Shells

Several of the shell element formulations allow composing the element via a stack of layers of different materials. This is used to model layered composites. When using layers, the available materials are *isotropic* and *orthotropic_layer*. Each layer must specify a material and thickness. A fiber orientation, which is a rotation of the layer material coordinate system with respect to the element coordinate system, may optionally be given. Thickness and fiber orientation for a multi-layer material must be specified layer by layer in the input deck. **Exodus** attributes may not be used.

Keyword **layer** defines a new layer for the current shell. Layers of the shell are stacked from the bottom to the top based on the order of the **layer** keyword in 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 7-8 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. [1]

An important parameter for the layered shells with orthotropic materials is the specification of a user-defined coordinate system with the **coordinate** option, see Section 7.9.7 In the example shown here, a coordinate system is defined for the shell block and orthotropic material properties are defined via a fiber orientation rotation with respect to that base element system.

[1]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 7.10.

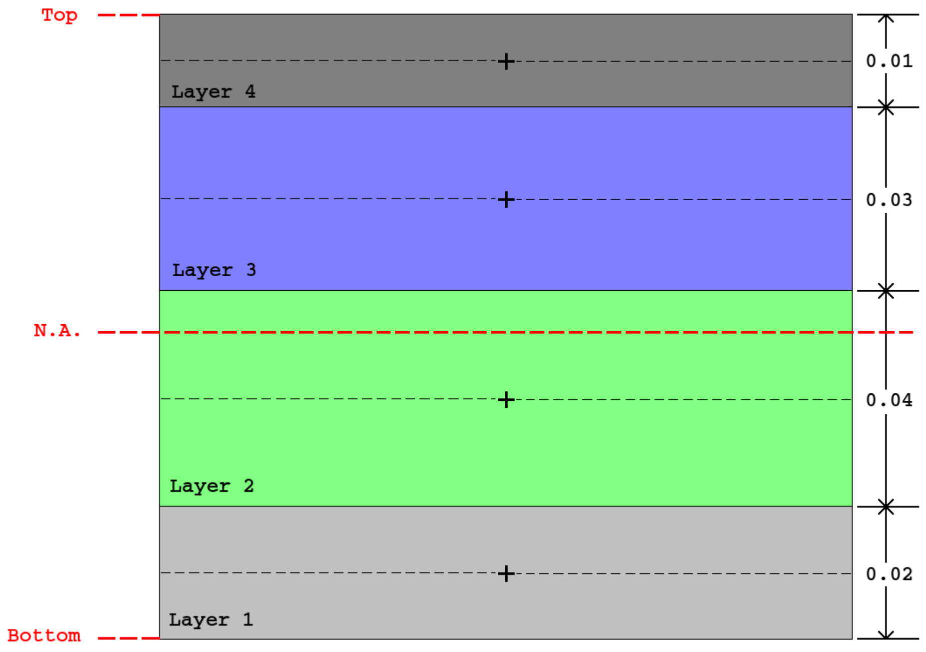


Figure 7-8. – Stacking arrangement for a multi layer shell element.

```
BLOCK 18
  TriaShell
  coordinate 1
  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

BEGIN RECTANGULAR COORDINATE SYSTEM 1
  origin 0.0 1.0 1.0
  z point 2.0 1.0 1.0
  xz point 0.0 1.0 10.0
END
```

Input 7.1. Layered Shell Example

Multi-layered shell stress can be written to the **Exodus** file. It is also written to the result file. if stress is selected in the echo section. The layer stresses will be computed only at the midpoint of each layer. Layer stresses at the top and bottom of each layer are not available.

The Navy Layered Shell with orthotropic materials sets G_{23} and G_{13} to the input value of G_{12}.

Offset Shells

By default, the meshed shell element lies at the midplane of the material volume represented by the shell. Shells may be offset from the midplane by specifying the offset input. The offset vector is the element unit normal vector scaled by the offset. An example is shown in Figure 7-9.

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 of offset shells is better than that of 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.

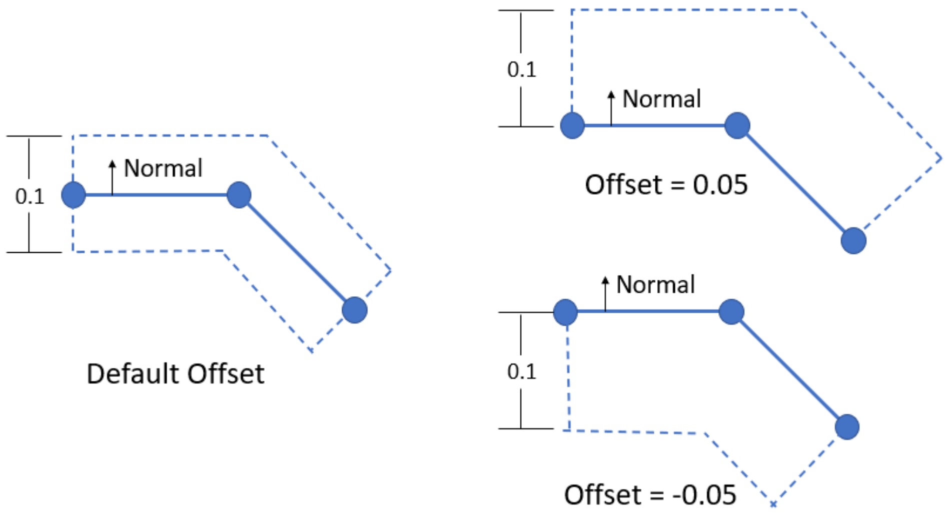


Figure 7-9. – Offset examples for shell of thickness 0.1.

In the `.inp` file the element offset is specified as,

```
offset=-3.14e-2
```

Offsets may also be specified in the **Exodus** file via attributes (see Section 7.9.9.) Some limitations of element offsets are described in Section 7.9.8.1.

7.9.8.1. Offset Shells and Lumped Mass

The elements more accurately model structures such as shells cladded on a volume. Offset elements necessarily couple the rotational and translational DOFs. 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 lumped (diagonalized) as described in Section 4.6.4.

Mass matrix lumping decouples the translational and rotational DOFs, which is inaccurate for offset shells. Specifically, while the total mass is conserved, the center of gravity and mass moments are not. The lumped 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. **Mass lumping with offset shell elements is discouraged**.

7.9.9. *Spatially Dependent Properties via* Exodus *Attributes*

Certain 2D element properties can be defined either in the input deck or via attributes on the input **Exodus** mesh. When a property is defined in the input deck it has a constant value for all elements of the block. The advantage of **Exodus** attributes is that a different value can be used in each element. This can be used to model properties such as variations in thickness in tapered shells.

The supported attributes are shown in Table 7-5.

Attribute Index	Keyword	Description
1	thickness	Thickness of the shell
2	fiber_orientation	Rotation of material with respect to local coordinate system
3	offset	Offset of the shell midplane with respect to the meshed surface: see Section 7.9.8

Table 7-5. – Shell parameters that can be set via attributes.

If an input deck value is given for a shell property, such as thickness, it will override any value given in the **Exodus** attributes. Furthermore, attributes cannot be used with multi-layer shells; all multi-layer shell element properties must be defined in input deck.

7.10. 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 thin. Details of the element formulation are available in a separate report.[22] An introduction to Hexshells is readily available in,[45] and the verification manual[46] discusses the results of the verification problems from[22] for **Sierra/SD**.

The Hexshell has a preferential thickness direction that must be set correctly. There are three ways to specify the thickness direction.

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.

Sierra/SD 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 9.8.4.

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 too. 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 its 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 other two methods do not depend on the decomposition.

Hexshell Parameters

Hexshell elements require 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 a string and integer argument. The first is the Name 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/name	sideset to specify thickness direction
2	tcoord	ID/name and direction	coordinate frame and coordinate direction
3	autolayers	# of layers and material	creates specified number of uniform layers of specified material

An example specification for a multi-layer Hexshell is shown in input 7.3.

```
BEGIN CYLINDRICAL COORDINATE SYSTEM thickness
  origin = 0 0 0
  Z point = 0 0 1
  XZ point = 1 0 0
END
```

Input 7.2. Hexshell: Coordinate Frame

```
BLOCK 86
  Hexshell
  sideset 88
  layer 1 material 1 coordinate 1 thickness .4
  layer 2 material 2 coordinate 2 thickness 0.6
END
```

Input 7.3. Multi-layer input with thickness direction determined using a sideset

```
BLOCK 89
  Hexshell
  material 1
  tcoord thickness 3  // azimuth
END

BLOCK 83    // the normal to
  Hexshell   //  sideset 1 will be
  material 1 // the thickness
  sideset 1  // direction for block 83
END
```

Input 7.4. Hexshell: Block 89 defines the thickness direction using a coordinate frame and the `tcoord` keyword.

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). [2]

There are two methods to specify multiple layers in a Hexshell. The first, illustrated in input 7.3, provides complete flexibility over the material specification, orientation and thickness of each layer. The `autolayers` feature provides are much more limited specification that is useful for models of a single material with temperature dependence across the thickness. It creates the specified number of layers, of uniform thickness, of a single specified material.

Stress output for HexShells is calculated for a standard Hex8 element, even when multiple layers are defined.

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.

[2] Layers for Hexshells must specify the *relative* thickness of the layer. This is in contrast to layered shells which specify the absolute thickness (Section 7.9.8.

```
BLOCK 70
  Hexshell
  sideset 1
  autolayers 4
  material steel
END

MATERIAL steel
  E function 1
  nu .3
  density 0.288
END

FUNCTION 1
  type Linear
  data 0 30e6
  data 1e6 20e6
END
```

An example of the Autolayers command is provided to the left. 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 with layers and with 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	46	Y	Y	some
multiple layers	no[†]	46	Y		

[†]Felippa’s report contains some verification. It has not been carried into **Sierra/SD**.

Table 7-6. – 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 definition use the same coordinate system.

```
BEGIN CYLINDRICAL COORDINATE SYSTEM 1000
  origin 0.0 0.0 0.0
  z point 1.0 0.0 0.0
  xz point 0.0 1.0 0.0
END

BLOCK 79
  Hexshell
  tcoord 1000 1
  material 8
  coordinate 1000
END
```

7.11. Beam2

The `Beam2` element formulation is described in Section 3.14 of.[45] 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 7.12.

The use of a `Beam2` element requires a `block` definition with a `beam2`. 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. Attributes may be defined in the input deck 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
```

The keywords are defined in following sections.

The definition of the bending moments, I_1 and I_2, is consistent with commercial codes including NASTRAN. The bending moments are the moments in their corresponding planes, not bending about their axes. If the moments are unequal, then an coordinate frame is required to apply the bending moments in the intended planes. If not specified, a warning will be output and the global coordinate frame will be used by default.

7.11.1. Beam Element Coordinate Frame

Figure 7-10 illustrates how the coordinate frame ($\boldsymbol{x}_{elem}, \boldsymbol{y}_{elem}, \boldsymbol{z}_{elem}$ depends on the element vertex coordinates and the orientation vector, $\boldsymbol{V}$. The $\boldsymbol{x}_{elem}$-axis is determined from the coordinates, $\boldsymbol{z}_{elem}$-axis is along $\boldsymbol{x}_{elem} \times \boldsymbol{V}$, and $\boldsymbol{y}_{elem}$ is parallel to $\boldsymbol{z}_{elem} \times x_{elem}$. If the provided $\boldsymbol{V}$ is not orthogonal to the $\boldsymbol{x}_{elem}$-axis, a warning will be output and an orthogonal axis will be chosen. A similar coordinate system is used for the Nbeam. 7.12

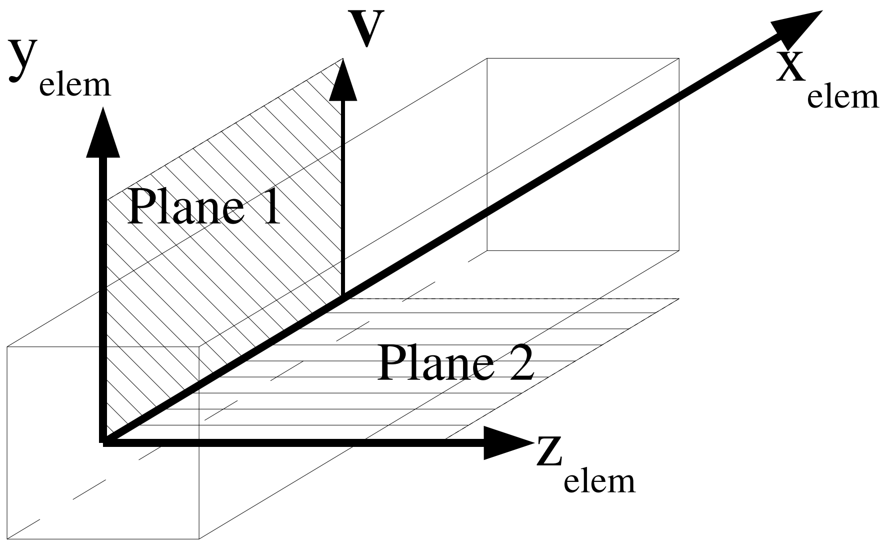


Figure 7-10. – Beam Orientation and Local Coordinate System.

In Figure 7-10 the $\boldsymbol{x}_{elem}$-axis and $\boldsymbol{z}_{elem}$-axis define plane 2 the $\boldsymbol{y}_{elem}$-axis, laying in the 1-plane, corresponds to a local 1-axis defined in a cross-sectional plane, a plane normal to the $\boldsymbol{x}_{elem}$-axis.

The z_{elem}-axis, which lies in the 2-plane, corresponds to a local 2-axis defined in the cross-sectional plane.

Four attributes are required. 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 z_{elem}-axis) is defined by the **I1** keyword. Bending moments in the 2-plane (or bending about the $\boldsymbol{y}_{elem}$-axis) is defined using the **I2** keyword. A polar moment of inertia, `polar_moment_inertia`, for torsion about the $\boldsymbol{x}_{elem}$-axis is required. The polar moment of inertia can be defined with the `J` keyword.

If the cross-section has the symmetry, $I_1 = I_2$, then the orientation vector $\boldsymbol{V}$ may be an optional attribute. Otherwise, $\boldsymbol{V}$ is a required attribute. It is necessary for the bending properties to have the correct global orientation. The components of the orientation vector can be specified with the values `x_orient`, `y_orient`, and `z_orient` using an `ORIENTATION` keyword or attributes.

The origin of the 1,2 coordinate system at the beam endpoints is the corresponding grid points by default. A user specified offset vector $\boldsymbol{V}_{off}$ translates the geometric location of the coordinate system origin. This offset vector is shown in Figure 7-11. For the `Beam2` element, one offset vector translates both ends of the beam. The `OFFSET` keyword is optional. The offset vectors move the beam neutral axis (the $\boldsymbol{x}_{elem}$-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 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.

For curved surfaces it is possible for the offset element to be inaccurate. The reason is that the radius changes, but the curvature does not change.

Refer to section 7.9.8.1 for limitations of element offsets.

Certain element attributes set in the mesh file are recognized in **Sierra/SD**. The element attributes corresponding to the input deck parameters are tabulated next.

Use the mesh file to specify attributes per element and the input block to specify attributes per element block. Input deck attributes override mesh file attributes. Attributes in the mesh file must be in the order specified in Table 7-8.

The **Beam2** element supports isotropic materials only.

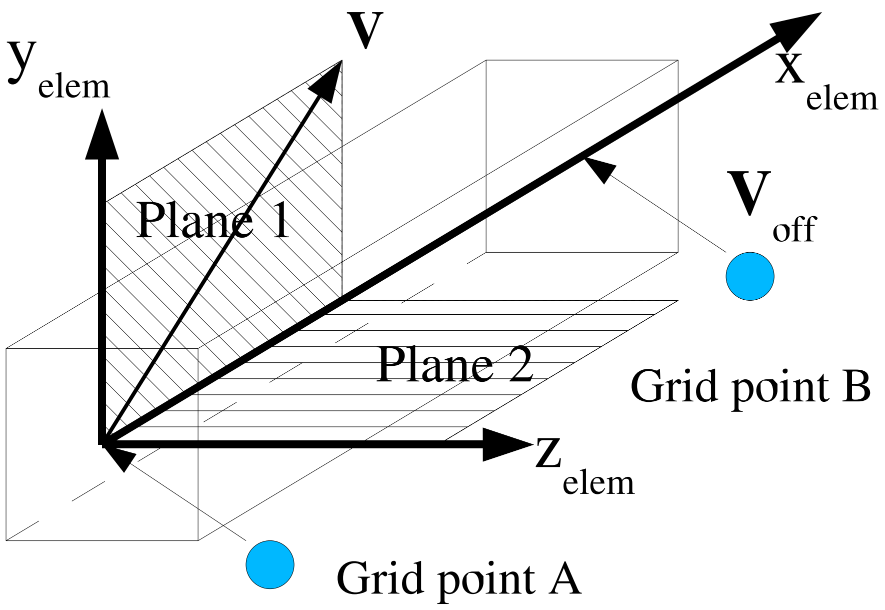


Figure 7-11. – Beam Offset and Local Coordinate System.

Table 7-7. – Element Attributes.

inertia_about_1	inertia_about_2	
polar_moment_inertia	area	
x_orient	y_orient	z_orient
x_offset	y_offset	z_offset

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 37
  Beam2
  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
```

Finally, the coordinate system for a beam can be defined using the intrinsic geometry. This can be done using the `coordinate from_geometry` option in the block definitions. See 6.6.1.3 for more details.

Table 7-8. – Attributes for Beam2.

	Keyword	Description
1	Area	Cross-section Area
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

7.12. Nbeam

The `Nbeam` element was developed from the COSMIC/NASTRAN 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 nonlinear 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_moment_inertia
  I12 = product_inertia_12
  Shear_factor_1 = sfactor1
  Shear_factor_2 = sfactor2
  orientation = x_orient y_orient z_orient
  offset = x1_offset y1_offset z1_offset x2_offset y2_offset z2_offset
END
```

The various keywords in the above `block` definition are described in following paragraphs.

Local coordinate frame The Nbeam uses essentially the same coordinates frame 7.11.1 as the other beam elements, with the difference being that the Nbeam supports independent offsets at either vertex. The $\boldsymbol{x}_{elem}$-axis lies along the length of the offset beam. In Figure 7-10 plane 1 contains the user specified orientation vector, $\boldsymbol{V}$ and the ($\boldsymbol{x}_{elem}, \boldsymbol{y}_{elem}$ axes. As is the case with all the one dimensional elements $\boldsymbol{x}_{elem} \times \boldsymbol{V}$ determines the z_{elem}-axis, and $z_{elem} \times x_{elem}$ gives the $\boldsymbol{y}_{elem}$-axis.

As the **Nbeam** supports arbitrary vector offsets at each end, the orientation of the *offset beam* may differ from the orientation of the geometry (see "offset" below) that is not offset.

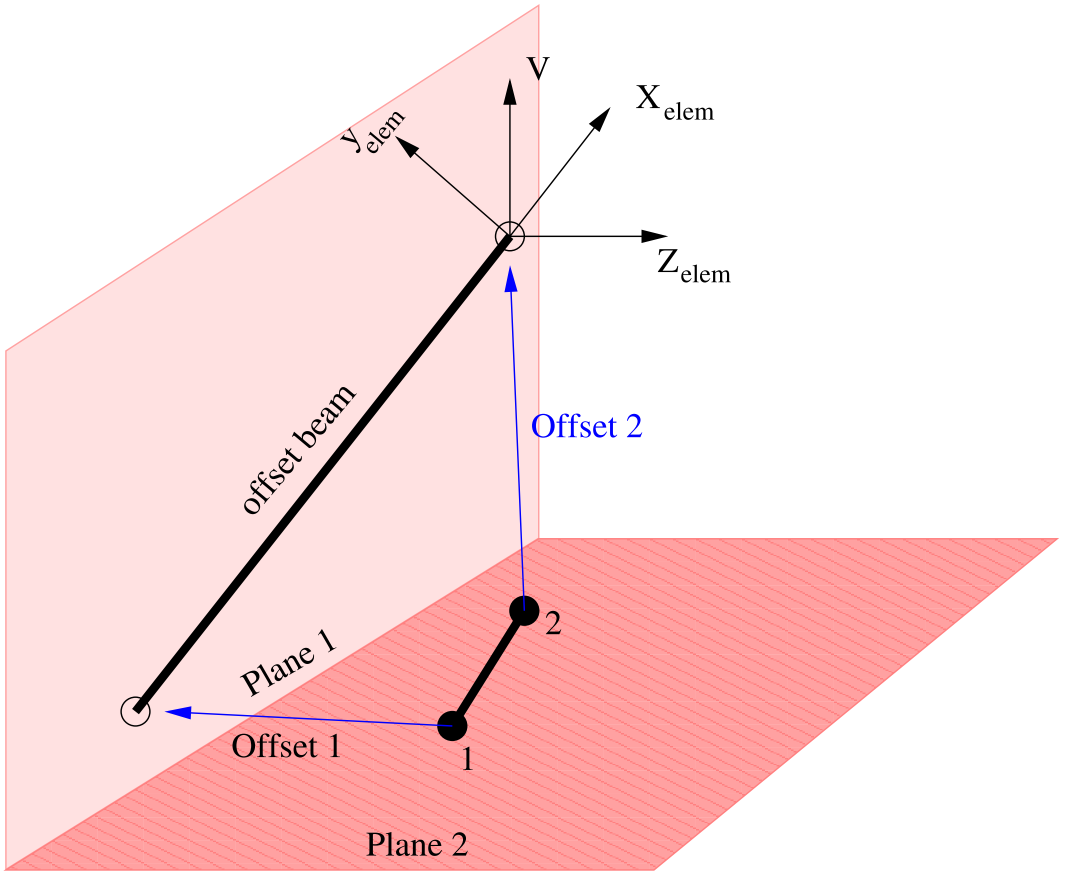


Figure 7-12. – Nbeam Orientation, Offset and Local Coordinate System. The coordinate system is in the plane of the *offset* beam. The plane is defined by the offset beam and the orientation vector, $\vec{V}$.

The $\boldsymbol{x}_{elem}$-axis and z_{elem}-axis define plane 2 in Figure 7-10. The $\boldsymbol{y}_{elem}$-axis, which lies in the 1-plane, corresponds to a local 1-axis defined in a cross-sectional plane, a plane normal to the $\boldsymbol{x}_{elem}$-axis. The 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 $\boldsymbol{y}_{elem}$-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 z_{elem}-axis), `inertia_about_2`, can be defined with the `I2` keyword. Finally, a polar moment of inertia, `polar_moment_inertia`, for torsion about the $\boldsymbol{x}_{elem}$-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.

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 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 varies over a cross-section. See Oden (Ref.[38]) 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 $\boldsymbol{V}$ 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, or mesh attributes. If no orientation or an orientation that is not orthogonal is specified, a warning will be output and an orthogonal orientation vector will be chosen.

Offset The `OFFSET` keyword is optional. The offset vectors move the beam neutral axis (the $\boldsymbol{x}_{elem}$-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 `x1_offset`, `y1_offset`, and `z1_offset`, `x2_offset`, `y2_offset`, and `z2_offset`. These values are associated with an `OFFSET` keyword, or attribute values.

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.

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 7.9.8.1 for limitations of element offsets.

Many of the parameters described can also be defined as attributes in the mesh file. Attributes in the mesh file must be in the order specified in Table 7-9. 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.

Table 7-9. – 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

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 47
  Nbeam
  material 7
  area 1.92
  I1 2.6
  I2 4.8
  J 0.026
  I12 -1.46
  shear_factor_1 0.44
  shear_factor_2 0.33
  orientation 1.0 0.0 0.0
  offset 0.5 0.5 0.5
END
```

7.13. TiBeam

This is a Timoshenko beam with consistent mass. For beams with an aspect ration of 100 there is little difference between the beam2 and the Tibeam. Beams with an aspect ratio of 10 differ significantly. The Tibeam uses the parameters of a **Beam2**. In addition the effective area in shear should be provided (the default is 2/3). For a beam with square cross section,

```
BLOCK 53
  Tibeam
  material 1
  // Shear_factor = 10(1+nu)/(12+11 nu)
  areay = .85
  areaz = .85
END
```

In this example, the area, I_1, I_2, J and the orientation are all set in the Exodus file. Recall that for a circular cross section, the approximate shear factor is given in terms of Poisson's ratio, ν, by $6(1+\nu)/(7+6\nu)$. At this time, although the parser warns that `areay` and `areaz` are ignored, they are parsed and applied.

7.14. Truss

This is the definition for a **Truss** element based on Cook (Ref.[17]). 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.

7.15. Ftruss

The **Ftruss** is a truss element whose stiffness is a function of the truss length.

Trusses have stiffness in only the axial direction. While they exist in a three-dimensional world, forces orthogonal to the axial direction result in no resistance, i.e. they are singular. Setting the axial force,

$$\vec{F}(\vec{L}_n, t) = -K(|\vec{L}_n|, t_n)\,\vec{L}_n \tag{7.15.1}$$

depends on the vector $\vec{L}_n$ 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.

Denote by K_o the stiffness of a standard truss, and by L_o the nominal truss length. $F = -K_o\,dx$ implies that $K = \frac{K_o dx}{L_o+dx}$. The definition in equation 7.15.1 was chosen so that a force may be applied when dx vanishes.

Table 7-10. – 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	*string*	*required*	function identifier (see Section 4.10)
-	Material	*string*	*optional*	If the material specification is provided, it must point to a valid material (sec. 6), and an area must also be provided.

If a standard (non-user-defined) 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 7-10, and a demonstration example is provided below. [3]

```
BLOCK 49
  Ftruss
  function 88
  scale 1.0
  material 17    //optional material
  area 0.01      //area required if 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. *scale* may be set either in the input deck, or in the **Exodus** file as the second element attribute.

7.16. ConMass

Concentrated masses are used to apply a known amount of mass at a point location. The **Exodus** file element topology is a sphere. Support for concentrated masses as two noded elements in exodus has been deprecated.

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.

[3] 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.

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 13
  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.

7.17. 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 (translational spring stiffness). Rotational parameters are supported using the **RSpring** element described in Section 7.18. 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 also be defined in user defined coordinate systems.

Attribute	Keyword	Description
1	Kx	translational spring constant in X
2	Ky	translational spring constant in Y
3	Kz	translational spring constant in Z

Spring stiffness may be defined as a constant value, calculated from the output of a function, or extracted from the **Exodus** element attributes (if the corresponding parameter is left blank). Input 7.5 includes an example of each approach.

Note that when defined via a function, the input is the current displacement. For a `nltransient` solution, the function `cubic_Ky` from input 7.5 is equivalent to a spring3 with `Kx1 = 1000`, `Kx2 = 2000`, and `Kx3 = 3000`. For `transient` solutions, the spring stiffness will not be updated unless the `nUpdateDynamicMatrices` parameter is used. In that case, the input displacement to the spring function will be the displacement at the *previous* time step.

```
FUNCTION cubic_Ky
  type = analytic
    evaluate expression "1000 + d*2000 + d*d*3000"
END

BLOCK 54
  spring
    coordinate 7

    Kx = 1e6
    Ky = cubic_Ky
    # Kz (from element attributes)
END
```

Input 7.5. Spring definition

7.17.1. 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.

While setting nonzero spring stiffness helps to avoid solver problems, underlying domain decomposition problems may still exist for parallel calculations. Domain decomposition tools employ heuristics for connection of springs to solids; the models are not compatible. Finite length springs often result in constraints on rigid body modes. [4] Springs fill an important analysis need, but analysts may find that in many cases 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 mitigated.

7.18. 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 (rotational spring stiffness). It is strongly recommended that all three components have some stiffness. This is particularly important when doing parallel analysis (see the discussion in Section 7.17.1). Translational stiffness require the use of the **Spring** element described in Section 7.17. 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
```

[4]This is not specific to parallel solutions. Most often, finite length springs introduce strain for a model rotation.

```
  Kry = 1.11E7
  Krz 0.1
END
```

7.19. 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 nine required parameters are translational spring stiffness. 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. Cubic springs may be defined in user defined coordinate system.

Each component of applied force is a cubic polynomial in the corresponding coordinate direction,

$$F_x = K_1^x u_x + K_2^x u_x^2 + K_3^x u_x^3 \tag{7.19.1}$$

Linear analyses use the first K_1 term only.

#	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

Here is 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
```

7.20. 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 eigendecomposition. 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.

$$K \cdot u + c \cdot \dot{u} + M \cdot \ddot{u} = f(t) \tag{7.20.1}$$

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 61
  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 7.17 for a discussion.

7.21. 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 7-11.

#	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 7-11. – SpringDashpot Parameters.

As shown in the table, all the elements of the matrices may be entered for this element. An example follows.

```
BLOCK 48
  SpringDashpot
    Kxx = 1e4
    Kyy = 1e4
    Kzz = 1e4
    Kxy = -1e4
    Kyz = -1e4
    Byz = 3.2
END
```

7.22. 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 7-13.

#	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 55
  Hys
  Kmax 4.5e+7
  Kmin 3.0e6
  fmax 5.9
  dmax 1.0e-6
END
```

7.23. Joint2G

The `Joint2G` element models adherence between surfaces. It is represented in the mesh by a beam or bar element. To preserve rotational invariance 3.8.5, typically beam length vanishes. The `Joint2G` element has the generic element properties 6.6 : materials, coordinate frames, optional nonlinearity (Iwan elements), damping, and non-structural mass 6-6. In addition a `Joint2G` has a shear axis, that will also be explained later.

Each `Joint2G` element connects a pair of nodes (or *grids*, hence the "G" in Joint2G). The constitutive behavior of each of the degrees of freedom connecting its node pair may be specified independently. A `Joing2G` elements is typically used with a rigid set 7.36

Relative displacement 9.7.10 may be output. The element responds with generalized scalar forces corresponding generalized displacements. The EForce output option 9.7.8 includes the generalized forces in the output.

The `Joint2G` element is used in the Tied Joint 7.38 pseudo-element. To make Sierra easier to use, the beam element of a Tied Joint is added implicitly by Sierra. These Virtual elements are included in the output mesh. This enables visualization of the Tied Joints. All references to Virtual elements apply only to the Virtual elements added for the Joint2G pseudo-elements in Tied Joints.

Joint deformation under cycling loading is complex. The Iwan element can accurately model bolted joints, but depends on parameters that must be calibrated (using experimental data). The `Joint2G` element was added with the Iwan models.

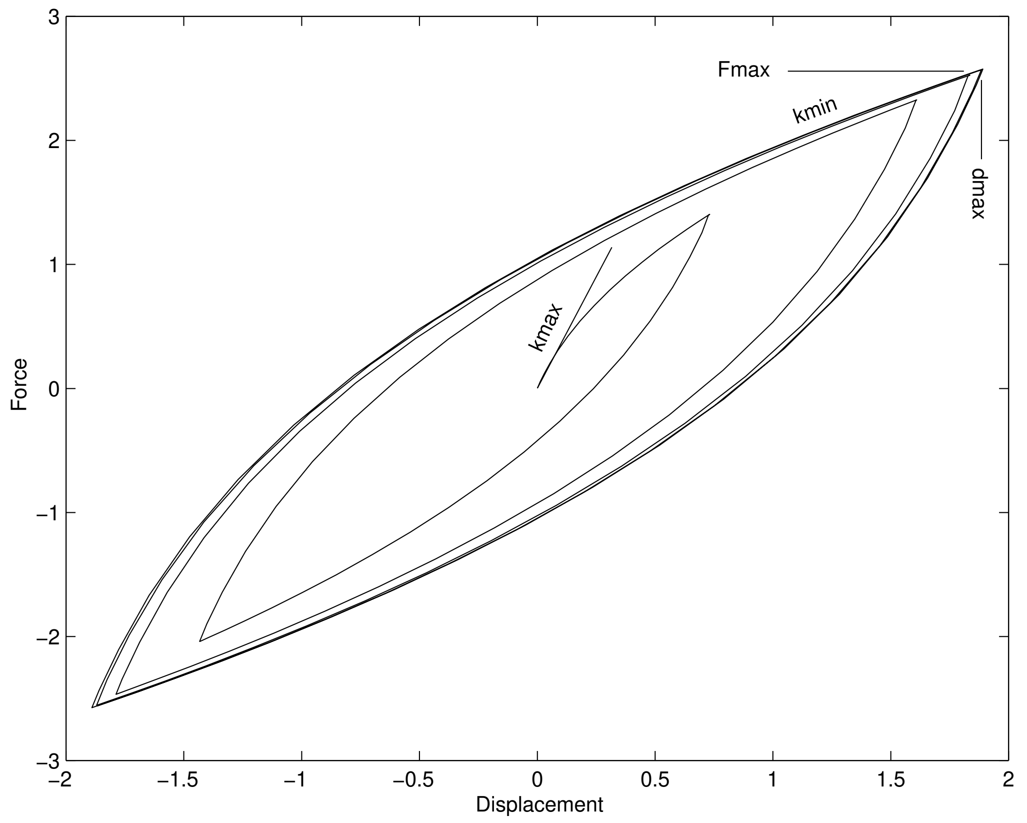


Figure 7-13. – `Hys` element parameters.

7.23.1. Specification

A `Joint2G` pseudo-element is represented in the input **Exodus** mesh file by either a `Beam` or a `Bar` element. Any element attributes are ignored. The `Joint2G` is configured in its input deck block section and an optional Property section. In the example below, properties are assigned to element block 1.

```
BLOCK  1
  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 1 to be of type `Joint2G`. It also declares the constitutive response in the *x* to be the 4 parameter Iwan model.[49] The model parameters are specified in

"Property 1" defined below. In this case, the four parameters chosen are χ, ϕ_max, R, and S. The Iwan properties can be specified alternatively by the parameter set chi, phi_max, F_S, and beta.

```
PROPERTY  1
  chi     =  -0.82
  phi_max =   1.0e-4
  R       =   7.6e6
  S       =   5.6e6
END
```

Another Iwan element, the RIwan is described in Section 7.23.3. However, the elements are not interchangeable. Although the Iwan and RIwan elements use the same terminology, some of the terms, such as

$$\phi_{\text{max}},$$

The constitutive behavior in the y and z directions is elastic with stiffness specified by the third argument - 1.0 x 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 not activated or *touched* by this element. For technical reasons, we recommended that if any rotational DOF is active (not *Null*), then they all should be active.

The directions x, y, and z are associated with the coordinate system declared for the block, e.g., coordinate system 5. The global coordinate system is the default.

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 7-27 and Section 7.38.

7.23.2. Constitutive Behavior

Elastic

Undamped, linear elastic behavior is defined by the `elastic` followed by the value of the parameter. No `property` section is required.

Damper

Linear, damped behavior is obtained using a `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.

```
PROPERTY 102
  K=1e6
  MU=.2
END
```

Input 7.6. Damper property card

```
BLOCK 59
  Joint2G
  kx=damper 1
  ky=damper 1
  kz=damper 1
  krx=null
  kry=null
  krz=null
END
```

Input 7.7. Joint2G as damper

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). Joint modeling with Iwan elements is described elsewhere.[50,54] Descriptions of the relationship of the Iwan element to other joint elements are also available.[51]

The schematic of the Iwan model is shown in Figure 7-14. Parameters for the behavior may be specified using either an older definition (Table 7-12), or a newer set (Table 7-13). The newer parameters are summarized below.

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,

$$\text{Dissipation} \approx (\text{Amplitude})^{\chi+3}$$

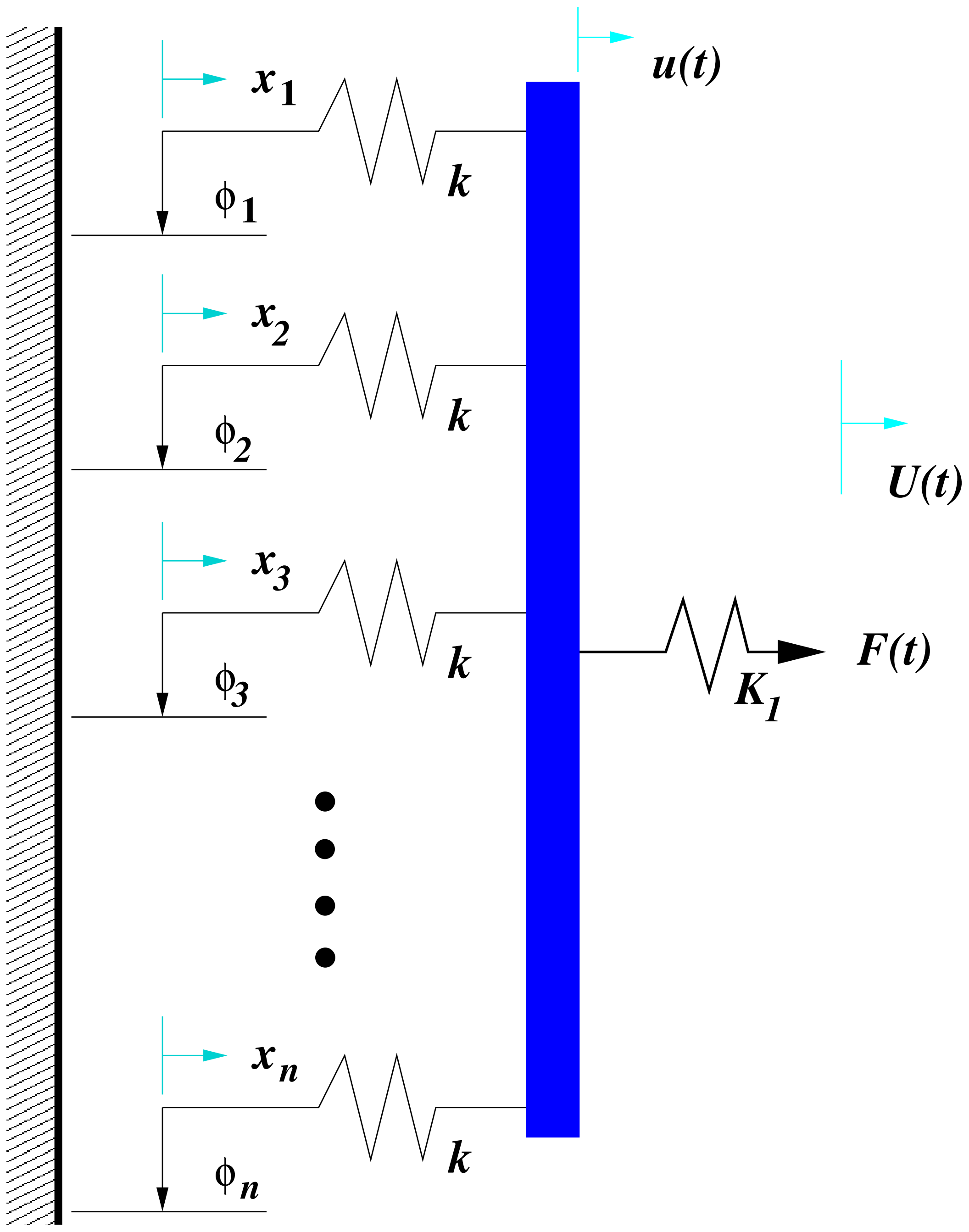


Figure 7-14. – **Iwan** Constitutive Model.

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at 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 7-12. – Older Iwan 4-parameter model.

#	Keyword	Description
1	chi	Exponent, χ, describing slope of force-dissipation curve at small amplitudes
2	beta	shape parameter of force/dissipation curve
3	K_T	Tangent stiffness at 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 7-13. – Revised Iwan 4-parameter model.

beta: determines the shape of the dissipation-force curve. Beta affects both the shape of the hysteresis curve within microslip (Figure 7-15), and the abruptness of the transition from microslip to macroslip as shown in Figure 7-16. $0 \leq \beta < \infty$.

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 into macroslip. The Iwan element is a statistical distribution of spring/slider elements. This is a point on that distribution.

The Reduced Iwan improves on the Iwan by requiring less calibration. Mechanisms to capture nonlinearity and dissipation are provided. Accuracy however depends on the fit with the experimental data under varying loading conditions.

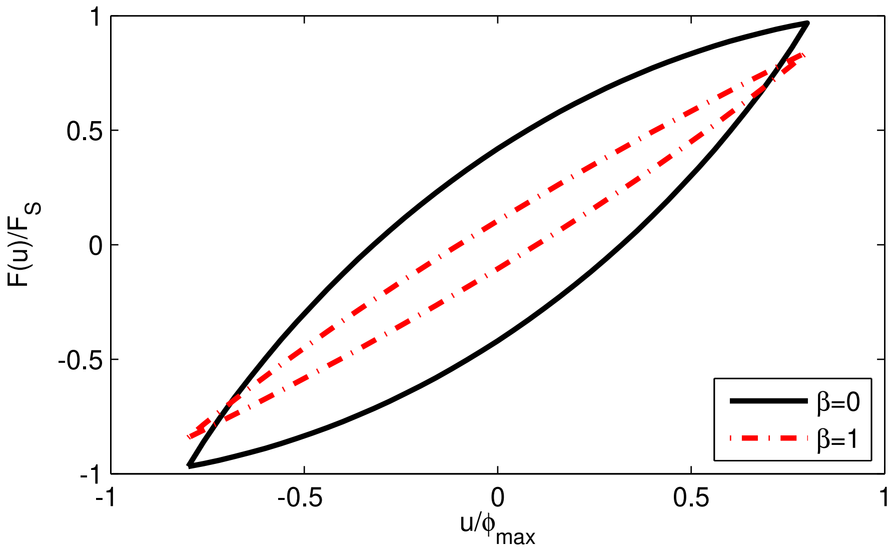


Figure 7-15. – Dimensionless hysteresis curves for the four-parameter Iwan model with $\chi = -1/2$ and two values of β.

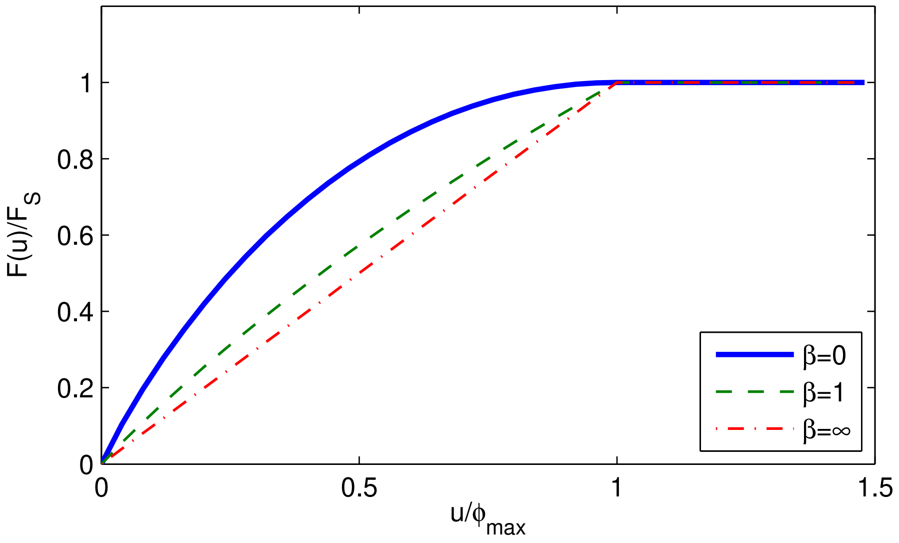


Figure 7-16. – Dimensionless static loading curves for the four-parameter Iwan model with $\chi = -1/2$ and three values of β, as the model goes into macroslip.

7.23.3. Reduced Iwan Plus Pinning (RIwan)

Reduced Iwan model, described later in the section, can be used as a constitutive model for a `Joint2G` definition using the `riwan`. This can be used for any desired direction, using the following format that requires an associated property block:

```
BLOCK 67
  Joint2G
  kx=damper 1
  ky=riwan 2
  kz=damper 3
  krx=null
  kry=null
  krz=null
END
```

Input 7.8. Joint2G Element Block

Although the Iwan and RIwan elements use the same terminology, some of the terms, such as

$$\phi_{\max},$$

```
PROPERTY 103
  FS=4e3
  Kt=1.5e7
  chi=-0.5
  beta=0.005
  Kp=2e7
  dp =2e-3
END
```

Input 7.9. Property text for the Joint2G

Description: An RIwan element[12] is a modified and revised 4 parameter Iwan model. The modifications include (a) simplification of the Iwan force by assuming that, upon load reversal, the distribution of friction elements resembles scaled version of the original distribution, and (b) incorporation of pinning forces through approximation of Hertzian contact. The monotonic load-displacement curve for this element is shown in Figure 7-17, where the initial nonlinear curve corresponds to microslip, the plateau corresponds to macroslip, and the steep linear curve represents pinning. In summary,

$$F_{RIwan} = F_{Pin} + F_{Sliding}. \tag{7.23.1}$$

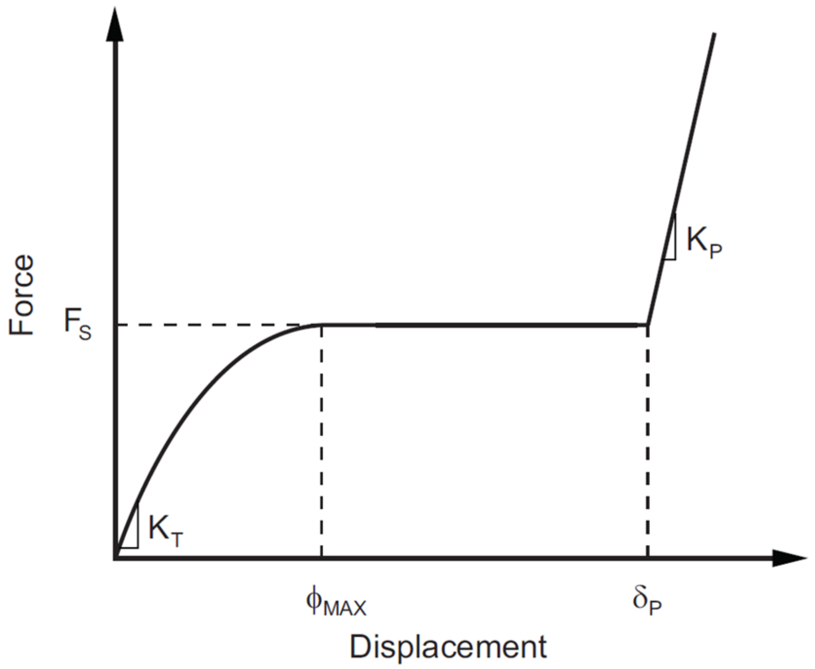


Figure 7-17. – **Reduced Iwan** Load Displacement Curve.

where the pinning force is given by the linear relationship,

$$F_{Pin} = \begin{cases} K_p(u - \delta_p), & u > \delta_p \\ 0, & \delta_p \le u \le \delta_p \\ K_p(u + \delta_p), & u < -\delta_p \end{cases} \tag{7.23.2}$$

where δ_p is the pinning displacement shown in 7-17. The sliding force is given by

$$F_{Sliding} = \begin{cases} F_0 + \frac{F_S - F_0}{F_S} F_{Iwan}\left(u \frac{F_S}{F_S - F_0}\right) & \text{loading} \\ F_0 - \frac{-F_S - F_0}{-F_S} F_{Iwan}\left(-u \frac{-F_S}{-F_S - F_0}\right) & \text{reverse loading} \end{cases} \tag{7.23.3}$$

where F_0 is the shifted central force with $-F_S < F_0 < F_S$. Table 7-13 and Figure 7-17 define $F_S, K_T, \chi, \beta, K_P, \delta_P$. We use the 4 parameter model Iwan force. $F_{Iwan} =$

$$\begin{gathered} \frac{F_S(\chi+1)}{\phi_{MAX}^{\chi+2}\left(\beta + \frac{\chi+1}{\chi+2}\right)} \left(\left(\frac{1}{\chi+2} - \frac{1}{\chi+1} \right) u^{\chi+2} + \frac{\phi_{MAX}^{\chi+1}}{\chi+1} u \right) \\ + \frac{F_S}{\phi_{MAX}} \frac{\beta}{\beta + \frac{\chi+1}{\chi+2}} min(u, \phi_{MAX}) \end{gathered} \tag{7.23.4}$$

One Dimensional Gap Model (gap)

The **Gap element** 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 and the axial behavior of the stand alone version of the `Gap element` implemented in **Sierra/SD** (Section 7.25).

#	Keyword	Description
1	Ku	Unloaded Stiffness
2	Kl	Loaded Stiffness
3	U0	Initial Gap Opening
4	F0	Preload (force at U0)

```
PROPERTY 104
  ku   =  1e5
  kl   =  1e6
  U0   =  0.01
  F0   =  200
END
```

Elastic Plastic Hardening Model (eplas)

`eplas` element is an elastic-plastic one-dimensional element with linear isotropic hardening. Both the plastic strain and the hardening variable are initialized to zero. The parameters are illustrated in Figure 7-18.

#	Keyword	Description
1	k	Linear Stiffness
2	kp	Hardening Stiffness
3	fyield	Force at Yield

```
PROPERTY 105
  k        =  1e6
  kp       =  1e5
  fyield   =  1e4
END
```

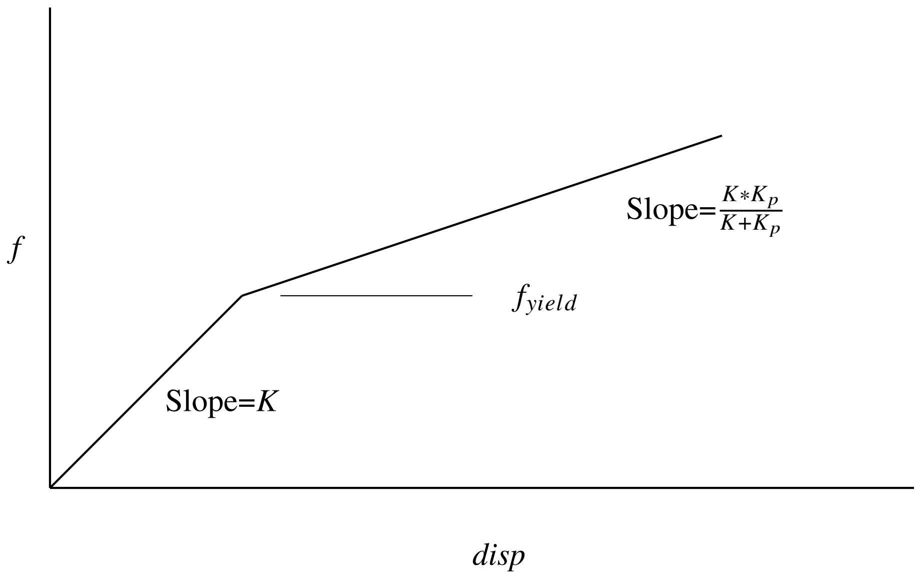


Figure 7-18. – Eplas Model.

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 modal 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.

$$K \cdot u + \mu \cdot \dot{u} + M \cdot \ddot{u} + K_3 \cdot u^3 + \mu_3 \cdot \dot{u}^3 = f(t) \tag{7.23.5}$$

#	Keyword	Description
1	K	stiffness
2	Mu	viscous damper coefficient
3	K3	*Optional* cubic stiffness coefficient
4	Mu3	*Optional* cubic damping coefficient

```
PROPERTY 106
  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

7.24. Line Weld

Line welds offer a way to join two shells by a collection of virtual Joint2G elements (section 7.23). The position of the line weld is defined by a collection of beams which are meshed contiguously with one of the shells. Using a contiguously-meshed beam instead of the shell directly to define the line weld position enables more general weld definitions (e.g. skip welds).

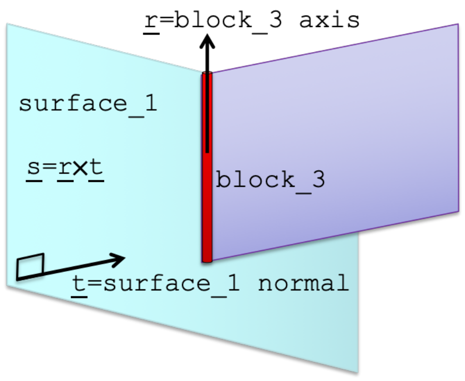


Figure 7-19. – Line weld definitions for attaching the purple and cyan shells. The line weld follows the red beam, which is contiguously meshed with the purple shell.

In figure 7-19, the line weld will be used to join block 3 (in red) to surface 1 on the cyan shell. This is accomplished in three steps, each illustrated by figure 7-20:

- Create a new virtual node (in green) coincident with every existing block node
- Tie each virtual node to the nodes of the shell on which it lies
- Create a zero-length, virtual Joint2G between every block node and its coincident virtual node

Currently, line welds in **Sierra/SD** are only available for the elastic Joint2G axial constitutive model. These can be thought of as a general translational/rotational spring, with axes as shown in figure 7-19. Input 7.10 gives an example input for the line weld shown in figure 7-19. Note that rotational stiffness is only supported along the direction of the beam (the “r-axis”). Any rotational components along the other two axes will be ignored.

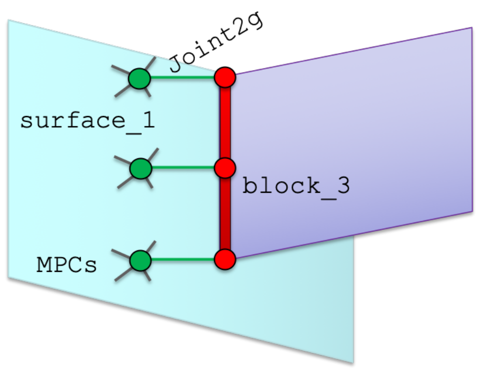


Figure 7-20. – Line weld Joint2G connections. The Joint2G elements are initially zero-length, and the green and red nodes coincident, but they are depicted with a finite deformation to delineate the individual components.

In keeping with Sierra/SM syntax, line welds may be specified using a force vs. deflection function as shown in Input 7.11. The current line weld capability is purely linear, and the line weld stiffness k is determined by estimating the derivative with a forward finite-difference stencil, i.e.,

$$k \approx \lim_{h \to 0^+} \frac{f(h) - f(0)}{h}.$$

A warning is issued if the corresponding backward stencil indicates that the derivative is not well-defined. If

$$\lim_{h \to 0^+} \frac{f(h) - f(0)}{h} \neq \lim_{h \to 0^-} \frac{f(h) - f(0)}{h}.$$

A fixed step size is used, so although the code will detect the case where the limits are not equal, it will not know whether the one-sided limits are well-defined. To aid the user in assessing the validity of these operations, the estimated stiffness values and other critical information is reported in the `rslt` file.

```
BEGIN LINE WELD
  surface = surface_1
  block = block_3
  search tolerance = 1e-4
  r displacement elastic = 1.0e6
  s displacement elastic = 1.0e6
  t displacement elastic = 1.0e6
  r rotation elastic = 1.0e6
  gap removal = on
END
```

Input 7.10. Line weld input corresponding to figure 7-19

```
BEGIN LINE WELD
  surface = surface_1
  block = block_3
  search tolerance = 1e-4
  r displacement function = R_force_function
  r displacement scale factor = 1.0
  s displacement function = S_force_function
  s displacement scale factor = 1.0
  t displacement function = T_force_function
  t displacement scale factor = 1.0
  r rotation function = Rrot_force_function
  r rotation scale factor = 1.0
  gap removal = on
END
```

Input 7.11. Force function syntax for line welds

Gap removal is enabled by default, and can be disabled using the `gap removal` option. The gap removal solution method 5.32 can make debug easier. Disabling gap removal can cause artificial grounding of rigid body modes, but can avoid collapsing elements when the search tolerance is large or when element quality is low. As with contact, the output mesh represents the model after gap removal.

Adding `line_weld` to an **outputs** or history block selects line weld output. To get the line weld output in the history file, the beam block from which the line welds were created must be included in the history section using **block**. The keyword `line_weld` adds the additional outputs shown in table 7-14 to the beam elements from which the line weld was created. The new virtual line weld elements created in the mesh have the usual output associated with a Joint2G element, such as **eforce**.

Line welds output force per unit length in the weld local coordinate system as `line_weld_force_rst` and `line_weld_moment_rst` .

Name	type	Description
line_weld_weld_active	int	A value of one marks beams within search tolerance to the surfaces that have active welds
line_weld_initial_weld_length	real	Length of the line weld beam element
line_weld_force	vector	Force per unit length produced by the line weld in the global XYZ system
line_weld_moment	vector	Moment per unit length produced by the line weld in the global XYZ system
line_weld_force_rst	vector	Force per unit length produced by the line weld in the element local RST system
line_weld_moment_rst	vector	Moment per unit length produced by the line weld in the element local RST system

Table 7-14. – Line weld output: note that these are intended to exactly match the equivalent outputs in **Sierra/SM**.

7.25. Gap element

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 7-21.

#	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	Coordinate frame.

If the gap is open, then the element stiffness is the unloaded stiffness, K_U, which must be positive. A gap is closed if $U_A - U_B > U_0$. If the gap is closed (as shown in the figure), then the element stiffness is the loaded stiffness, K_L.

The initial gap opening and preload define the corner point in the force/deflection curve as shown in Figure 7-21. 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 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 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 5.20 (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 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 'nlresiduals' option in the echo section (see 9.9) which will put convergence information into the results file.

An example is shown below.

```
BLOCK 42
  Gap
  KL 4.5e+7
  KU 3.0e6
  KT=1e6
  f0 5.92
  u0=0.98e-6
  coordinate 5
END
```

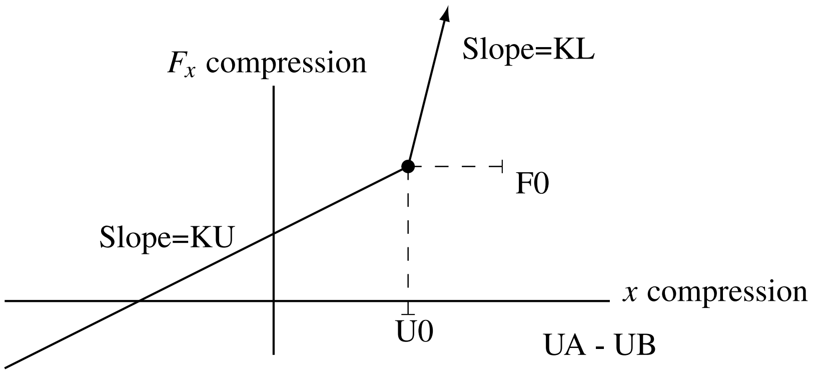


Figure 7-21. – Gap element Force-Deflection Curve.

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 3.8.5. 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 easy to get this direction reversed.

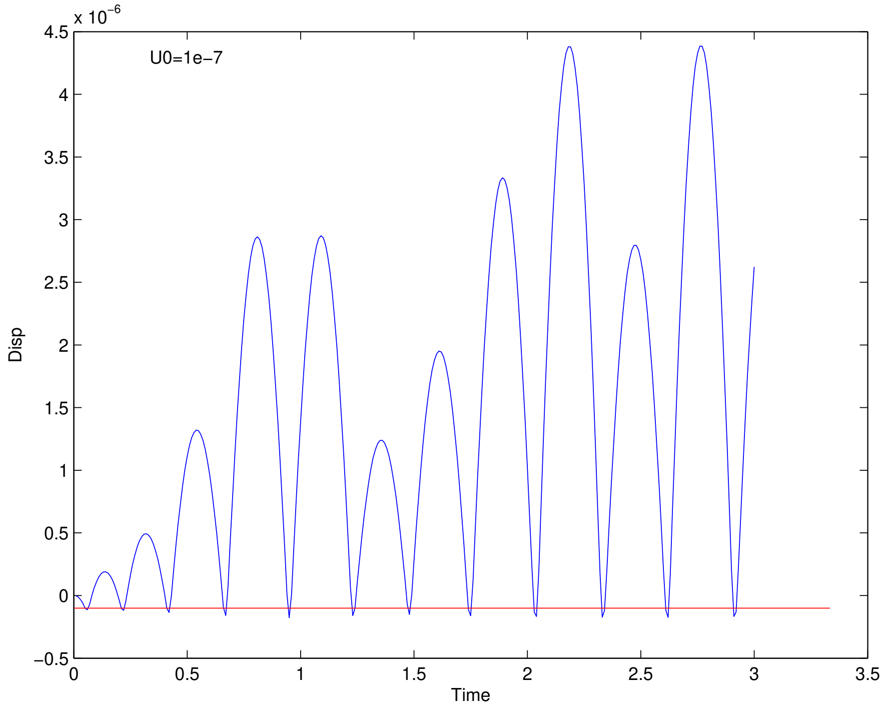


Figure 7-22. – Mass bouncing off a Gap. With this large time step the model is not conserving energy. Reducing the time step is required to correct the problem.

- 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 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 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 7-22, 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.

7.26. Gap2D

The **Gap element** of the previous section provides a useful construct for planar type interactions. A common modeling issue is a bolt hole that is too large. To model this interaction an ellipsoidal Gap element (or **Gap2D**) may be required.

The **Gap2D** element operates 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.

$$\left(\frac{u_x}{\mathrm{U0X}}\right)^2 + \left(\frac{u_y}{\mathrm{U0Y}}\right)^2 < 1 \qquad (7.26.1)$$

The major and minor axes of the ellipse are defined in the x and y direction of the coordinate frame.

Parameters of the **Gap2D** element are listed below.

While the gap geometry is defined as an ellipse, stiffness is 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 from 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 7-23, 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 $f_y^{(1)} = \mathrm{KL} \cdot y_1$. In path 2, we follow the straight line through (x_b, y_b). The associated force is $f_y^{(2)} = \mathrm{KU} \cdot y_b + \mathrm{KL}(y_1 - y_b)$. For this element, we always choose the shortest line path (path 2). This ensures that the force is not history dependent.

#	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	Coordinate frame.

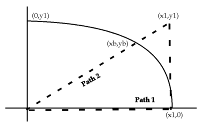


Figure 7-23. – Gap2D force diagram.

7.27. GasDmp

GasDmp is currently BETA release.
Enable with the "`--beta`" command-line option.

A **GasDmp** element is a nonlinear, beam-like element that simulates the damping forces on MEMS devices due to gas pressure as MEMS beams vibrate. It has no stiffness, but has damping roughly proportional to velocity/L^3, where L is the distance from the beam to the substrate. It is experimental. Contact Troy Skousen of *Sandia National Labs* or Professor Burak Ozdoganlar at Carnegie Mellon. The parameters are implemented through the input file and not through the **Exodus** file. 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

7.28. 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. One interface allows the user to provide an arbitrary subroutine to evaluate the constitutive equation.

An example of the user interface is shown in input 7.12. Mount orthogonal directions must be provided either as attributes in the **Exodus** file, or using the "Orientation" keyword in the "block" section. The relation of the orientation vector to internal element coordinates is shown in Figure 7.28.1. 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. A list of built in Nmount types is listed in Table 7-15.

```
BLOCK 41
  Nmount
  mount type = 99
    parameters = 1.4 3.1 2.7
    orientation = 0 0.7 1
END
```

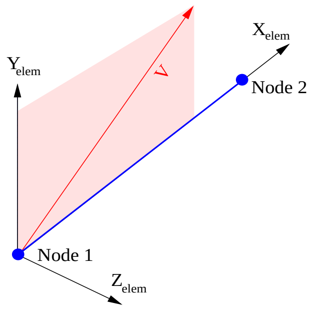
Input 7.12. Mount Interface

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 fewer parameters are provided than are anticipated for the mount, the last parameters are set to zero, a warning is printed, and the analysis continues.

Mount type 7 is a special user subroutine mount. It uses Fortran or C functions that are compiled at runtime. For this option **user subroutine file** must be defined in the file section. The user subroutines are incorporated into a custom build of the **Sierra/SD** executable with a command line like

```
sierra --make salinas -i my\_input.inp
```

Descriptions of the mount user subroutine formats and requirements are available.[47]



$$\begin{aligned} Z_{elem} &= (\hat{X}_{elem} \times \vec{V})/(|\hat{X}_{elem} \times \vec{V}|) \\ Y_{elem} &= \hat{Z}_{elem} \times \hat{X}_{elem} \end{aligned} \tag{7.28.1}$$

Nmount orientation is explain in equation (7.28.1). In the figure, X_{elem} normalized vector from node 1 to node 2, and changes as the structure deforms. The user provides the orientation vector $\vec{V}$.

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.

Name	Index	Comment/Parameters
fail_truss	1	1 = Axial stiffness, Kr 2 = torsional stiffness 3 = critical value of tensile strain 4 = critical value of rotational strain
SpringDashpot	2	1 = Axial stiffness, Kr 2 = Stiffness in S frame, Ks 3 = Stiffness in T frame, Kt 4 = Axial damping, Cr 5 = Damping coefficient in S frame, Cs 6 = Damping coefficient in T frame, Ct
Snubber	3	1 = Model Direction (1=axial, 2=radial) 2 = Model Type (1=10k, 2=20k) 3 = Snubber gap 4 = offset 5 = input weight on compression 6 = input angle of radial action
TorsionalSpring	4	1 = Torsional spring constant, K_θ $M = K_\theta(\theta_2 - \theta_1)$
Test-spring	5	1 = K_x 2 = K_y 3 = K_z 4 = K_{θ_x} 5 = K_{θ_y} 6 = K_{θ_z} $F_j = K_j x_j$ and $M_j = K_{\theta_j}\theta_j$
NLSpring	6	1 = K_x 2 = K_y 3 = K_z 4 = W_x 5 = W_y 6 = W_z $\vec{F} = K\,x + W\,x^2$
subroutine	7	Mount behavior defined by mount force subroutine = <string> mount init subroutine = <string> mount size subroutine = <string>

Table 7-15. – Nmount Models and Attributes.

7.29. Rrod

An Rrod is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for a 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** input deck. No material is required. A block may contain Any number of connected or disconnected Rrod elements. The following is an example of the input file specification for an Rrod if the **Exodus** database contains beams in block id=27.

```
BLOCK 27
  Rrod
END
```

7.30. 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.

Table 7-16 summarizes the corresponding parameters.

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 elements may be placed in a block. The following is an example of the input file specification for Rbar elements if the **Exodus** database contains beams in block id=29.

```
BLOCK 29
  Rbar
END
```

Rbar elements can be reordered so that the number of them connected to a single node is minimized. Having a large number connected to the same node results in a 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 4.4).

The Rbar attributes are listed in Table 7-16, and are described below. These attributes may be set in the input deck in which case the attribute is used for every Rbar of the block, or may be set as an element-by-element specific attribute in the input mesh file.

Attribute	default	description
RB_ID	-	translation identifier
CID_FLAG_INDEP	123456	independent coordinate flag
CID_FLAG_DEPEND	123456	dependent coordinate flag

Table 7-16. – Rbar Exodus Attributes.

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. [5]

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. As an example if an Rbar had a "CID_FLAG_INDEP" value of "13456" the rbar would introduce only 5 rather than 6 constraints and the rbar could shear in the y direction.

Interaction of Rbars If two Rbars or sets of Rbars share a node then they are effectively merged into a single rigid set. E.g., if an Rbar connects nodes 1 and 2, and another Rbar connects nodes 2 and 3 then the effect is the set of nodes 1, 2, and 3 act as a single rigid set of nodes. The same rule applies to intersection of rigid sets 7.36 or the intersection of Rbars and rigid sets as the rigid set capability is effectively a means to automatically generate Rbars.

RBE2

Sierra/SD has no support for the NASTRAN **RBE2** element. However, in most cases the **RBE2** element is not that different from a collection of `Rbar` rigid elements.

[5] It is an unusual descriptor, but it was designed to somewhat mimic the NASTRAN cid flag.

7.31. RBE3

The RBE3 pseudo-element's behavior is taken from the NASTRAN RBE3. More detail is found in the Theory Manual section **Sierra/SD** Elements subsection Rigid Elements subsubsection RBE3.

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 reference 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 elements, 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.

Keyword	**value**	Description
refc	*string*	reference node coordinates
WT	6 *reals*	relative weight of coordinates

refc The REFC parameter sets the degrees of freedom to activate on the reference node. For instance REFC='12' activates 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 the 6 structural degrees of freedom on the reference node.

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 reference 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 that 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 formulation of the RBE3 is based directly on the published method from MSC/NASTRAN. Details of the method are described in the Theory Manual section **Sierra/SD** Elements subsection Rigid Elements subsubsection RBE3.

Cautions in using RBE3

Albeit convenient, the **RBE3** is not a true element. It can introduce complexity in the solution.

- A RBE3 may connect a large portion of the model. This degrades linear solver efficiency. As a consequence, convergence may be slow.
- A RBE3 connected to many nodes requires a lot of memory. This memory is stored on a single processor.
- No two MPCs should be linked together. Linear solvers may fail in this case.
- Accelerations (see Section 8.1.4) cannot be prescribed 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 its 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, every link in the RBE3 must be connected to the reference node.
- A Joint2G with side averaging uses two RBE3 elements.
- Many user issues are caused by 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=31.

```
BLOCK 31
  RBE3
  refc=123456
  wt=1 1 1 0 0 0
END
```

7.32. Superelement

Keyword	**value**	default	Description
file	*string*		input file containing matrices
format	*string*	netcdf	format of Superelement input file: netcdf or DMIG
savememory	*yes/no*	no	controls storage of matrices in memory
diagnostic	*integer*	1	0 - run no diagnostics, 1 - compute `Kr * RBM`, 2 - compute `eig(Kr,Mr)`
map	*integer*		table of node/cid pairs
map	*string*		"ascending_id" or "sorted" or "locations"
mode	*integer/real*		additional modal damping coefficients applied to individual modes of superelement
skip_output	*yes/no*	no	option to disable netcdf output
sensitivity_param	*integer/real*		parameter index and value of sensitivity parameter may be used multiple times to specify different parameters

A Superelement is an abstract concept with different realizations in commercial codes. **Sierra/SD** does not have a fully automatic Superelement capability, however, the **Sierra/SD** CBR solution case 5.3 reduces an entire model to a reduced-order model that can be used in subsequent runs. All **Sierra/SD** solutions support use of models with Superelements.

Superelements are used to decrease the total number of DOFs in a structure that consists of substructures. One of the substructures is extracted into its own mesh (with **GREPOS** for example) and the CBR solution reduces the substructure to a Superelement.

To use in subsequent analysis, a new structure needs to be created by removing the substructure captured in the Superelement. Then, the Superelemnt is appended to the new structure to replace the removed substructure. The **Mksuper** application handles some of the associated book keeping in the Exodus mesh file for inserting Superelements. This process is documented in the Example Problems Manual[44] in the Superelement Insertion section.

A natural case is that a substructure of a structure is meshed much more finely than is needed for the current simulation. Among other things, replacing that substructure with a Superelement reduces the amount of computation needed for these simulations.

Evaluating the suitability of a Superelement approximation is left to the user; tutorials are available in the Example Problems Manual.[44]

Limitations

- The Superelement must be small enough (have a sufficiently small number of degrees of freedom) to fit (in the virtual memory) a single MPI rank. 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. The Superelement modal degrees of freedom will be stored in the **Exodus** and MATLAB output on the X-degree of freedom of newly created virtual nodes.
- 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. For a Superelement formed from a free-free system this will preserve the total translational mass of the Superelement. However, the rotational inertia and center of gravity will not be exactly preserved in the Superelement mass properties. If the Superelement has an internal constraint or if for other reasons the Superelement cannot exactly reproduce rigid body modes then the translational mass properties will not necessarily be preserved by the Superelement.
- **wtmass** is not going to be applied to the mass matrix. It is assumed that the units are already integrated as a result of using the **wtmass** option in the **cbr** solution
- 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

The following input is provided by the user. If `format=DMIG`, the connectivity information is read from the **DMIG** file, and cannot be specified in any other way.

connectivity (Exodus): Note that codes such as NASTRAN input Superelements by connecting to the nodes directly. Like any element with **Sierra/SD** the Superelement must be mapped to a single processor. The Superelement must be in the finite element database used to partition the elements. To provide the geometric connectivity to the model, the connectivity must be added to the **Exodus** file in one way or another.

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. **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. The connectivity map cannot have more nodes than the element.

The **Mksuper** utility function will add a Superelement to an **Exodus** database. See.[44]

connectivity map (Exodus): 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 (DOFs) such as fixed interface modes. 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 negative, 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. 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. [6] 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 among 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. Each Superelement interface node will be mapped to the closest finite element model node. No search tolerance is needed as the closest node is always found, even if it is far away. However, when using this option care should be taken that Superelement interface nodes and finite element nodes match in space. If a Superelement interface degree of freedom is mapped onto a finite element node with significantly different coordinates the Superelement behavior may be substantially degraded and the Superelement may no longer be able to represent rigid body modes. A warning is emitted by **Sierra/SD** if a node match cannot be found within a distance of one one-millionth of the characteristic model size.

system matrices: The system matrices may be provided in a **netcdf** or **DMIG** file. These matrices are available as output of the CBR reduction process (Section 5.3) 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. Its dimension must match that of the stiffness matrix.

`Cr`. An optional reduced damping matrix may be used. It 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.

An accurate 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 non-singular. Failure to meet these requirements can confuse the entire solution procedure, and lead to erroneous solutions.

transfer matrices (Exodus): Output of results on interior points in the Superelement are facilitated using optional output transfer matrices (OTM). These are described in the section on Craig-Bampton reduction (5.3). These matrices are written to the output **Exodus** file *only* if Superelement output is requested in the **Craig-Bampton reduction** output specification. The following matrices apply.

OTM Nodal output transfer matrix.

[6] 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.

OTME Element output transfer matrix.

OutMap An optional node map for the OTM.

OutElemMap An optional element number map for OTME.

mode: For nonlinear transient, additional damping parameters can be specified on a mode by mode basis for each fixed-interface mode. When a mode is not specified, the properties from the input system matrices will be used. When a mode is specified, the specified properties will override the properties from the input system matrices. Currently, only three types of damping behavior are defined: a damper with optional cubic parameters, **damper**, an Iwan model, **Iwan** and a reduced Iwan model, **riwan**.[45].

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 Superelement block would be writing to the same file location, resulting in corrupted data.

output specifications: In the input deck Superelement output is selected from either the **outputs** or **echo** sections by adding the word `Superelement`.

If requested in the **outputs** section, then a new **Exodus** file will be generated from 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 OTM. The number of elements is the number of elements in the OTME. All elements will be placed in a single element block.

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 impedes visualization, 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: If the **Craig-Bampton reduction** that generated the Superelement included a sensitivity analysis, then 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 as needed. 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 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.

The above parameters are entered in the **block** section of the input file. For example,

```
BLOCK 97
  Superelement
  file='example.netcdf'
  //  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
  mode 1 damper 1          // Only supported in nltransient
  mode 3 riwan 2           // Only supported in nltransient
END
```

If using the **mode** option, **property** sections must also be defined. It is important to recognize that the values of these parameters are in the *modal space*, so they need to be scaled appropriately. See the Verification Manual[46] for more details. Nonlinear parameters are also supported in nonlinear transient (**nltransient**) solutions.

```
PROPERTY 1
  K = 909.094116210938
  Mu = 1.8181818281549
  K3 = 82.64521119e8 //optional, default=0
  Mu3 = 82.64521119e-3 //optional, default=0
END

PROPERTY 2
  Fs=1.5e4
  Dp=0.03
  Kt=1.5e6
  Kp=3.0e7
  Chi=-0.5
  Beta = 1e10
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. Note that the format is assumed to be **netcdf** because the keyword `format` is not specified.

DMIG *Input Files*

Matrix	Acceptable Names
Stiffness (`Kr`)	`K2GG,KAAX`
Mass (`Mr`)	`M2GG,MAAX`
Damping (`Cr`)	`C2GG,BAAX`

Table 7-17. – Acceptable names of matrices within **DMIG** input files.

- **Sierra/SD** does not check the file extension of **DMIG** input files. Files generated by **Sierra/SD** use the `.dmig` extension, whereas direct output from NASTRAN uses `.pch`. Both are acceptable as input.
- Allowable matrix names are listed in Table 7-17.
- Information required by **Sierra/SD** in parsing **DMIG** files comes from the matrices themselves: all comments are ignored, including e.g., those providing information about the dof map, number of interface modes, etc.
- The stiffness damping matrix `K4AXX` is ignored.

Finally, we show an example input deck for **DMIG** format:

```
BLOCK 99
   Superelement
   format = DMIG
   file=rom4.dmig
END
```

7.33. Dead

Dead is now DEPRECATED.
Please consider using "omit block" (section 7.34) instead.

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 also no parameters to be specified for `dead` elements.

Special care must be taken when dead blocks are used/referenced elsewhere in your analysis. A few commonly-used cases are documented below.

Note that some loads and other boundary conditions on dead block regions will still be partially applied. Namely, any nodes that are shared with the remainder of the model will be included. This behavior is consistent for dead blocks and “body” regions, as well as nodeset or sidesets within an dead block.

In contrast, sideset regions of dead blocks will be skipped (with a warning) when used with tied data.

Sidesets on internal boundaries between dead blocks and active blocks can cause unexpected behavior. Deactivation of the dead block may deactivate the boundary condition on that internal sideset *if the sides are associated with the dead elements*. This may be diagnosed by visualizing the sideset normal, which points outward from its attached element. It is recommended to **avoid using internal sidesets** due to the ambiguity they can cause.

Extreme care must be used when combining dead element blocks with sidesets in contact. Generally, dead blocks specified in contact definitions or tied data will be ignored, and **Sierra/SD** will issue a warning message. However, *sidesets* of dead blocks will only be ignored in contact surface definitions if the surface *touches only dead blocks*. Any sidesets that span both dead and active element blocks will *not* be filtered from contact.

7.34. Omit Block

Blocks may be omitted from your analysis by adding a **omit block** line(s) to your **FILE** section. Note: an omit block line may only contain comma or space-separated lists of valid block ids and names; assemblies of blocks, ranges of block ids, and the **remove** keyword are **not** allowed. Element blocks that are omitted from the input are also omitted from the output.

Note that while not required, including a corresponding BLOCK definition is recommended for omitted blocks for for accurate syntax checking. This is in contrast to removing the block from the mesh as a preprocessing step (e.g. with grepos), which would require the block definition to be removed.

```
FILE
  geometry_file = mesh.exo
  omit block 12,wing tail
  omit block block_123
END
```

Input 7.13. Omit Block Example

> Special care must be taken when omitted blocks are used/referenced elsewhere in your analysis. A few commonly-used cases are documented below.

Nodes that are only found in an omitted block (i.e. not shared) will be omitted as well. Attempting to reference omitted nodes will result in a fatal error, for example when omitted nodes are found in MPC definitions.

Sideset regions of omitted blocks will be skipped when used with tied data. Likewise, omitted blocks and sidesets/nodesets of omitted blocks will be skipped when used in a contact definition.

Loads or boundary conditions applied to omitted blocks are also ignored. Likewise for sidesets spanning a dead block. For nodesets and “body” regions, any nodes that are shared with the remainder of the model will still be included.

7.35. Compatibility of SD/SM Elements

Some default **Sierra/SM** element formulations differ significantly from the corresponding **Sierra/SD** formulations. This means that on a coarse mesh these elements may produce different results for a nominally equivalent problem. Additional inconsistencies may result from the hand-off of **Sierra/SM** state via the receive_sierra_data section 5.23 solution case. For example an equilibrium state in SM may not be a SD equilibrium state.

On the other hand, some **Sierra/SM** element formulations types are identical to the corresponding SD formulations, including fully integrated formulations such as the Tet4, Hex20, and fully integrated Hex8 element (Hex8f in SD, fully_integrated in SM). A fully integrated Tet10 behaves slightly differently in SM than SD due to the SM default volume averaging behavior. The selective deviatoric element are identical: SD Hex8u and SM selective_deviatoric. A ’Nquad’ shell element in **Sierra/SM** is identical to a ’Nquad’ in **Sierra/SD**, but all other shell elements differ. One SM shell element formulations nearly matches the ’Nquad’ formulation, the ’BL_SHELL.’

Note, even when using equivalent element formulations full equivalency between SD and SM only holds at very small strain and small deformation. At larger deformations SD and SM results will diverge due to geometric non-linearities, differences in strain measures, and other linear vs. non-linear effects.

See the **Sierra/SD** Verification Manual[46] for additional information on these topics.

7.36. Rigid set

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 2:5
  nodeset 88-90, wing
  block 102
END
```

The above definition would establish a single set that is tied together. For purposes of error reporting only, the *optional* name “set1” is associated with this example set. If multiple *independent* sets are required, then multiple rigid set definitions may be made.

Table 7-18 shows the parameters. Any number of `Rigidset` sections may be introduced. Each acts independently. Exodus sideset, nodeset, or block information may be included in the definition. The rules for defining multiple nodesets, sidesets, or blocks at once are the same as the history output Section 9.2.

Table 7-18. – Rigid set parameters.

Parameter	type	description
sideset	int/name/list	sideset(s)
nodeset (*not recommended*)	int/name/list	nodeset(s)
block	int/name/list	block(s)
CenterNode tiedto node	integer	see below

Tied Node

`tied node` or `center node` A rigid set can be attached to a Tied Joint 7.38 or Joint2G 7.23. In this case, a “reference” node may be generated and tied to another block or element. This is accomplished with the keywords below.

```
CenterNode tiedto node XX block YY
```

The `CenterNode` command will create a bar element with two nodes, and associate it with block `YY`. One end of the bar element is node `XX`. Node `XX` must exist in the input mesh. The other end node of the bar element will be created on the fly at the centroid of the rigid set. Note this capability only works for blocks with a single element. There are examples in.[44] Figure 7-24 illustrates the concept.

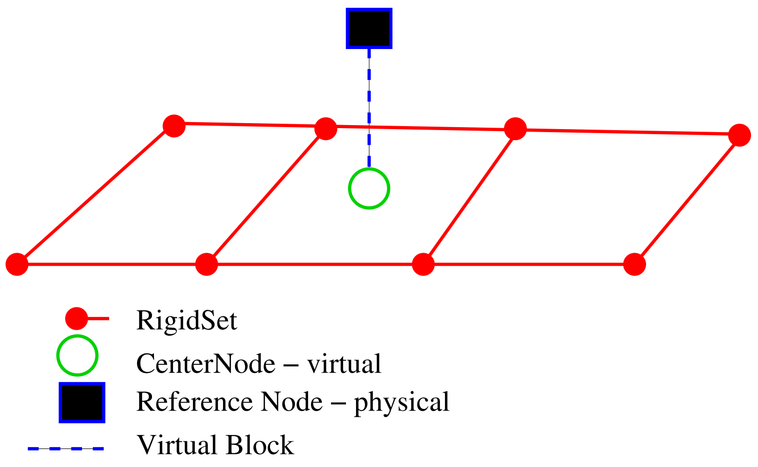


Figure 7-24. – The model illustrates the connection of a physical rigid set to a physical reference node via a virtual center node and virtual connection block.

7.36.1. Voltage Rigid Sets

For many models using piezoelectric materials, a free surface of the piezoelectric tile may be plated with a purely conductive material such as copper. The conductive layer results in an equipotential surface. To simplify modeling an equipotential surface, **Sierra/SD** enables voltage rigid sets, which enforce a spatially constant voltage on the nodes associated with specified nodeset or sideset. Specifying voltage rigid sets are done in the rigid set block as shown in the following.

```
RIGIDSET unique_identifier
  sideset 1
  voltage
END
```

Input 7.14. Voltage Rigid Set

7.36.2. Limitations

The rigid set meets an important need to tie together many nodes Generally they are much more robust than generating collections of `Rbar` rigid elements or other rigid elements. However, it is easy to generate redundant constraints through this input. Redundant constraints cause most linear solvers to fail, and **Sierra/SD** may not always provide diagnostics. Generally,

1. Rigid sets should be completely disjoint, i.e. should share no common nodes. If two rigid sets do share any node they are effectively merged as a single larger rigid set.

2. If a Rbar is connected to any node of a rigid set then effectively the combination of Rigidset and connected Rbars behave as a single rigid system. Similarly, connecting two rigid sets via a Rbar effectively merges the two sets into a single larger rigid set.
3. None of the nodes in the rigid set should be constrained (as through a boundary condition).
4. While nodesets can be used to define rigid sets, 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 sidesets or blocks to define the rigid set.

7.37. Rrodset

Rrodset is currently BETA release.
Enable with the "- -beta" command-line option. Like the `Rigidset` of Section 7.36, 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. Additionally, only sidesets may be used to define a Rrodset. The `Rrodset` acts much like a fiber textile: 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 7-25. An example is shown below.

```
RRODSET
  sideset 5
END
```

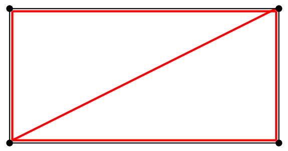


Figure 7-25. – Rrodset Constraints. The black lines indicate the edge of the element. Red lines are corresponding linear constraints.

Like the `Rigidset`, the `Rrodset` may be used to connect a "reference" node to a block.

7.38. Tied Joint

The *Tied Joint* models a joint structure. At the heart of the *Tied Joint* is an *Iwan* model. The *Tied Joint* supports flexibly mixing many models. An Iwan element may be used to represent the shear response, and multipoint constraints may be used to represent the normal response. Energy loss of the joint can be approximated by the *Iwan* element, and normal surfaces alignment can be preserved by the constraints.

Input Specification Refer to Figure 7-26 for reference to the model definition. An example input is shown in input 7.15. There are several sections to the model definitions. Parameters of the input are summarized in Table 7-19. Details are below.

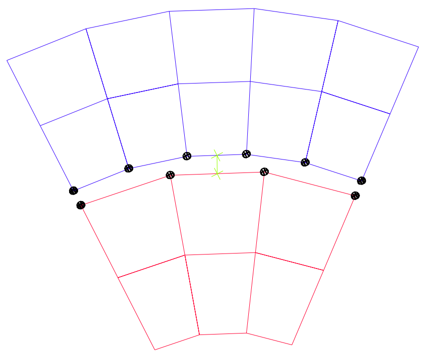


Figure 7-26. – Tied Joint Geometry. The two side set surfaces are shown separated for clarity. A virtual element is created which connects only the shear components of the joint. Normal components are interconnected using Tied MPCs.

```
TIED JOINT
  Normal Definition = slip
  surface 3,5
  connect to Block 59 // Joint2G block
END

// definitions for the referenced Joint2G
BLOCK 59
  Joint2G
  Kx=Iwan 1
  Ky=Elastic 1e6
  coordinate=5 // for anisotropic shear parameters
END
```

Input 7.15. Tied Joint Example

Name: Optional name of this joint. Useful primarily in diagnosing error messages.

Surface: Exactly two sidesets should be provided, these two sidesets will be connected via the joint. For node-face MPCs involved in normal direction constraints the nodes of the second surface (node-surface) are constrained to the faces of the first surface (face-surface)

surface	2 sidesets connected by the joints, in slip constraints the first sideset is the face-surface and the second the node-surface as in tied data
tied nodes	a sideset connected by the joint, the face-surface in slip constraints as in tied data
tied faces	a sideset connected by the joint, the node-surface in slip constraints as in tied data
normal	
slip|none	if `slip`, normal-only node-face constraints
search tolerance	if `slip`, defines node-face MPCs
edge tolerance	if `slip`, defines node-face MPCs
shear	
connect to block	reference block for whole joint
side	`average`, `rigid` or `Rrod`

Table 7-19. – Tied Joint Parameters.

Normal Definition: In the *Tied Joint*, the joint behavior in the normal direction can be governed by the joint element or by distributed node-face MPCs on the sidesets.

Slip implies the surfaces will remain in tied contact, and shear effects are managed by the "shear definition". Tied node-face constraints will be created similar to the Tied Data 10.1 command. These constraints only tie the normal deformation of the sides together. The shear behavior of the joint will be managed by the single joint element. In this case, the joint element will be located at the nearest node to the centroid of the node-surface.

None implies that no specific node-face normal constraints will be generated. Surfaces may separate or interfere and the joint normal behavior will be controlled only by the whole-joint element. In this case, the joint will be located at the centroid of the node-surface.

Connect to block: A reference to a block containing parameters for the whole-joint element, a single element that connects the two sides of the joint. Usually a Joint2G element is used. If the normal definition is `none`, then the whole-joint element must specify behavior in all six dofs. Otherwise, if the normal definition is `slip`, then tied constraints are used to constrain the normal motion of the joint and only the three dofs associated with plane motion in the whole-joint element are used.

For non-isotropic shear behavior, the block may include a coordinate command. The frame may be curvilinear (e.g. cylindrical), in which case whole joint quantities are evaluated at the centroid of the surfaces (see coordinates, 4.9). To reference the basic (or default) frame, use coordinate frame "0". 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 $\tilde{X}$ and $\tilde{Y}$ axes of the user-defined coordinate system are projected to the plane, with the new third axis ($\tilde{Z}'$) in the normal direction.

In the case when the whole-joint element is a `Joint2G` element, the `shear_axis` can be used in the block definition to define the coordinate direction used for the first in plane constitutive component. We refer to Figure 7-27 for a description of the local coordinate system used to specify the constitutive behavior of the `Joint2G` element. The surface normal, n, is obtained 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 $\tilde{Z}'$ 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 $\tilde{X}'$ is defined as the part of the $\tilde{X}$ axis from the user-defined coordinate system that is orthogonal to $\tilde{Z}' = n$. If the $\tilde{Z}$ axis of the user specified coordinate system lines up exactly with the normal at the node, then the shear direction will be in exactly the same direction as the $\tilde{X}$ 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. Once $\tilde{Z}' = n$ and $\tilde{X}'$ are defined, the remaining component of the coordinate system $\tilde{Y}'$ can be obtained by a cross product.

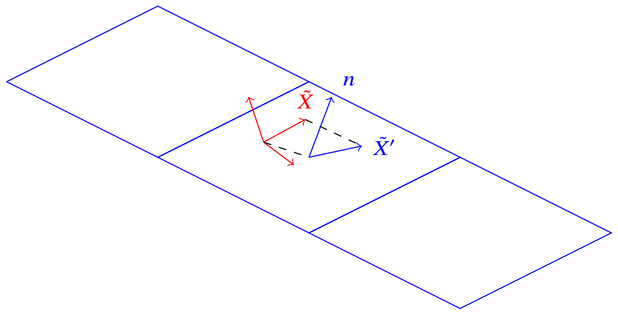


Figure 7-27. – The surface normal, n, is defined by the normal of the surface at the centroid point. The shear axis direction (in this example $\tilde{X}$) is projected onto the surface as $\tilde{X}'$. Together, $\tilde{X}'$ and surface normal provide the basis for the generated coordinate frame.

Side: The `side` defines additional constraints on the surfaces and how the tied joint Joint2G element spreads load to the surfaces.

A tied joint is between two surfaces A and B. To create the tied joint first a point in space is selected near the centroid of the joint. Next two new collocated nodes A_c and B_c are created at this point. Node A_c will track the average displacement of side A. Node B_c will track the average displacement of side B. The whole joint element, a Joint2G, is connected between nodes A_c and B_c and controls the stiffness and damping of the joint.

average See Figure 7-28. A_c and B_c are connected to side A and B via RBE3 constraints. One RBE constrains node A_c to have the average displacement of side A and another RBEs constrains node B_c to have the average displacement of side B. The RBE3 constraints are added as bar elements to the output mesh file. Side average is the default for “normal definition” of “slip”.

rigid provides a means of constraining all the nodes on each surface to move together as a rigid set (see users 7.36). See Figure 7-29. Side A is turned into a rigid set and side B is turned into a rigid set. Node A_c is added to rigid set A and node B_c is added to rigid set B. An RBE3 is not required in this case as the rigid set already guarantees that the displacement at A_c and B_c track the average displacement of the sides. Side rigid is default for "normal definition" of "none".

Rrod provides a means of constraining all the nodes on the surface to move together as a Rrodset (see users 7.37). See Figure 7-30. Side A is turned into a Rrodset and side B is turned into a Rrodset. As with side=average RBE3 constraints are needed to constrain nodes A_c and B_c to the average displacement of the two sides. The RBE3 constraints are added as bar elements in the output mesh file.

The side **rigid** option will create the stiffest overall joint behavior, **average** the softest, and **Rrod** something in between. Using option **rigid** to avoiding a complex `RBE3` constraint can also avoid numerical issues that sometimes come with the **average** or **Rrod** options.

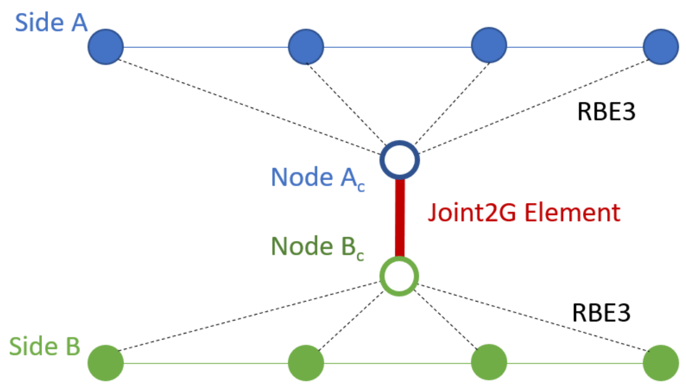


Figure 7-28. – Construction of tied joint with side=average.

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 7-20 summarizes some dependencies between these two parameters.

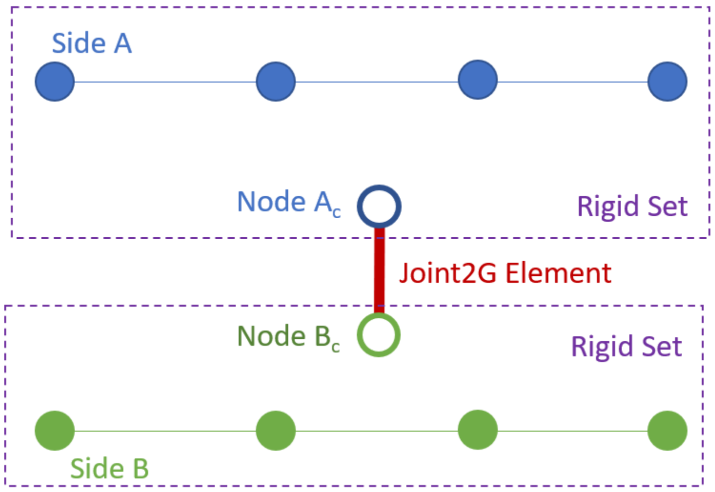


Figure 7-29. – Construction of tied joint with side=rigid.

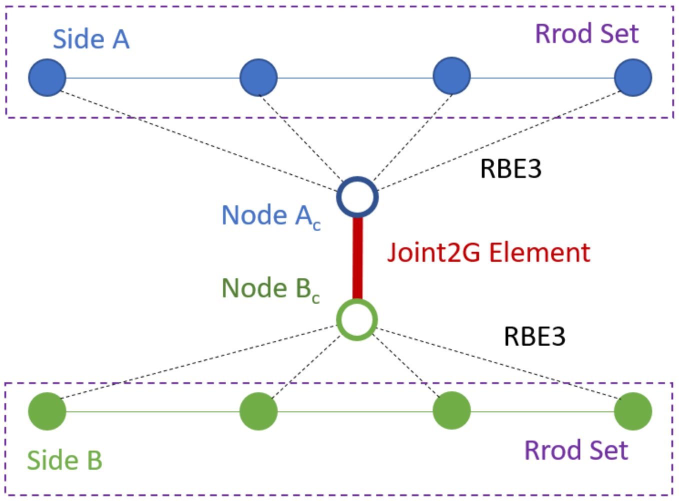


Figure 7-30. – Construction of tied joint with side=Rrod.

Normal Definition	Side	Status
none	average	RBE3s constrain joint end nodes to have the average displacement of each surface.
	Rrod	RBE3s constrain joint end nodes to have the average displacement of each surface. Additionally, each surface is turned into an Rrod-set.
	rigid	Each surface is turned into an rigid set which also contain the joint end nodes. An RBE3 is not needed in this case as the rigid set already constraints the joint end nodes to have the average displacement of the surfaces.
slip	average	Node-face normal only constraints are added to the surfaces. RBE3s constrain joint end nodes to have the average displacement of each surface. The shear deformation of the whole-joint model is based on the relative lateral motion of each side.
	Rrod	Node-face normal only constraints are added to the surfaces. RBE3s constrain joint end nodes to have the average displacement of each surface. Rrod constraints stiffen the surface. The shear deformation of the whole-joint model is based on the relative lateral motion of each side.
	rigid	Invalid. Overly constrained joint. Fatal error.

Table 7-20. – Tied Joint, *Normal* and *Side* dependencies.

7.38.1. *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.

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_summary” in the “ECHO” section (see 9.9). This includes the following.

1. The type of the normal enforcement.
2. Surface information.
3. Centroid of the surface pairs (if applicable).
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 input 7.15, 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.

```
BOUNDARY
  sideset 5                    // for nodes in sideset 5
    absorbing                  // use an absorbing bc

  sideset 2                    // for the acoustic sideset 2
    p = 0                      // fixed acoustic pressure
                               // (pressure release condition)

  sideset acoustic_surface     // for sideset "acoustic_surface"
    pdot = 1.0                 // constrain the time derivative
                               // of acoustic pressure for
                               // enforced accelerations
    function = 2               // varying in time with function 2
    p0=1.0                     // and initial condition p0 = 1.0

  sideset 6                    // for sideset 6
    impedance_pressure=0.5     // use a pressure impedance bc
    impedance_shear = 0.5      // and a shear impedance bc

  sideset 7                    // for sideset 7
    slosh = 0.6                // use a slosh bc

  sideset 8                    // for sideset 8
    infinite_element           // use infinite elements
    use block my_block         // where the infinite element
                               // parameters are in the
                               // user-defined block "my_block"

  sideset piezoelectric_side // for "piezoelectric_side"
    V = 0                      // fixed voltage
                               // (electrical ground)
END
```

Input 8.1. Sideset

An input deck has a **boundary** section for specifying boundary conditions. Node sets, side sets, blocks, or node lists may be used to specify boundary conditions. The rules for defining multiple nodesets, sidesets, or blocks at once are the same as the history output section, 9.2, and also follows the rules for integer lists detailed in Section 4.1. Specialized coordinate systems that can be used are described in section 4.9. The example in input 8.1 illustrates the method.

```
BOUNDARY
  nodeset end_nodes            // for nodes in nodeset "end_nodes"
    x = 0                      // constrain x=0
    coordinate 1               // for x/y/z in coord system 1

  nodeset 1                    // for nodes in nodeset 1
    x = 0.1                    // constrain x=0.1
    y = 0                      // and y=0
    RotZ = 0                   // and the rotational dof about Z

  nodeset 13:15, widget        // for nodesets 13-15 and "widget"
    accelx = 0.3               // constrain the x-acceleration
    function=1                 // varying in time with function 1
    disp0 = 0.1                // and initial condition
    vel0  = 0.2                // disp0 = 0.1*0.3 and vel0  = 0.2*0.3
END
```

Input 8.2. Nodeset

```
BOUNDARY
  block fixed_block            // for nodes in block "fixed_block"
    fixed                      // constrain all dofs

  node_list_file='nodes.txt' // for nodes in the file "nodes.txt"
    fixed                      // constrain all dofs
END
```

Input 8.3. Block or Node List

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 8-1. An optional equals sign separates each descriptor from the prescribed value. The value `fixed` implies a prescribed value of zero for all degrees of freedom.

Note however that the syntax checking in the boundary block does not check for duplicate boundary conditions, and silent failures are possible. For example in the example shown in input 8.4, part of the input block is silently ignored.

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
V	Voltage
	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
AccelV	second derivative of voltage
disp0	initial displacement
vel0	initial velocity
Pdot	derivative of acoustic pressure
P0	initial acoustic pressure
	prescribed displacement keywords (Direct FRF-only)
DispX	scaling factor on X component of motion
DispY	scaling factor on Y component of motion
DispZ	scaling factor on Z component of motion
RotDispX	scaling of rotational motion about X axis
RotDispY	scaling of rotational motion about Y axis
RotDispZ	scaling of rotational motion about Z axis
FreqV	scaling factor on voltage
FreqP	scaling factor on acoustic pressure

Table 8-1. – Dirichlet Boundary Enforcement Keywords.

```
BOUNDARY
  nodeset 10   // This
    rotx = 0   // is
    roty = 0   // parsed
    rotz = 0   // as
  nodeset 10   // expected
    accelx=370.
    function=1
    accely=380. // Parsing
    function=2  // silently
    accelz=390. // ignores
    function=3  // these!
END
```

Input 8.4. Example Silent Failure of Parsing

To apply functions 2 and 3, it is necessary to provide a nodeset for each function as in input 8.5.

```
BOUNDARY
  nodeset 10 rotx = 0 roty = 0 rotz = 0
  nodeset 10 accelx=370.0 function=1
  nodeset 10 accely=380.0 function=2
  nodeset 10 accelz=390.0 function=3
END
```

Input 8.5. Corrected syntax

The way that the parser works for the boundary block is that the text is divided up into consecutive chunks by the keywords `nodeset`, `sideset`, `block`, `node_list_file`, and `end`. This is the pattern followed in input 8.1. Within each chunk, any number of Dirichlet boundary enforcement keywords (see Table 8-1) may be provided. Surprisingly, only the first function in a chunk is parsed; any others are ignored. There is no warning for ignored text in the boundary block at this time.

8.1. Boundary conditions

This section provides more information about more complicated boundary conditions that can be applied using the `boundary` section of an input deck.

Node_List_File To make it easier to apply boundary conditions, a *node_list_file* option is provided. In this option, the user provides an additional text file that contains a list of global node ids separated by white space. No comments, or other characters are allowed in the file, as shown in input 8.1. The remainder of the boundary condition specifications are unchanged.

There are several limitations placed on collections of nodes specified in this manner:

1. This is an 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 mapped **Exodus** global numbers, which is the arbitrary node numbering provided by the analyst. Earlier versions of **Sierra/SD** used "1 to N" ordering, where N is the maximum number of nodes in the model. Recent versions always use the mapped ordering for referring to global nodes.
5. There is no requirement that the nodes be sorted in the list, **but repeating a node in the list can have undefined results**.

8.1.1. Prescribed Displacements and Pressures

In linear statics, one may prescribe a nonzero displacement by entering a value following the coordinate direction, as shown in input 8.6.

```
BOUNDARY
  nodeset 1
    x = 3
    y = 0
    z = 0
    rotx = 0
    roty = 0
    rotz = 0
END
```

Input 8.6. Prescribed Displacement for Statics

For acoustics, pressures may be fixed by specifying $p = 0$, as in Table 8-1 on sideset 2. This corresponds to a pressure release condition.

For linear statics, there must be no **function** entry following the entry. 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.

Prescribed displacements have the same limitations as prescribed accelerations, described in Section 8.1.4.

8.1.2. Prescribed Displacement in Transient

Prescribed Displacement in Transient is currently BETA release.
Enable with the "- -beta" command-line option.

Similar to the transient acceleration capability 8.1.4, in direct transient analyses time-history displacement boundary conditions may be specified with the keywords `DispX`, `DispY`, `DispZ`, `RotDispX`, `RotDispY`, or `RotDispZ` paired with a function.

A voltage time history may be directly defined by keyword `transV` and a time history function .

8.1.3. Prescribed Voltage

For electro-mechanical coupled physics problems, constant voltage boundary conditions may be specified on nodesets or sidesets using the keyword `V` in the Boundary block. In the following example, electrical grounds were set at sideset 1 and nodeset 1, and a non-zero constant voltage boundary condition set at sideset 2.

```
BOUNDARY
  sideset 1
    V = 0
  nodeset 1
    V = 0
  sideset 2
    V = 2
END
```

Input 8.7. Prescribed Voltage

8.1.4. *Prescribed Accelerations*

In transient dynamics, the acceleration on a portion of the model may be prescribed as a function of time. As shown in Table 8-1. acceleration are specified using `accelX`, `accelY`, `accelZ`, `RotaccelX`, `RotaccelY`, `RotaccelZ`, `disp0`, `vel0`, `Pdot` and `accelV`

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 nodesets 13 through 15 and "widget" will be prescribed as $0.3 * f(t)$, where $f(t)$ is defined in function 1. The $accelx$ factor also scales the initial displacement and velocity. Thus, initial displacement is given as $0.1 * 0.3$ and the initial velocity is $0.2 * 0.3$.

Prescribed accelerations are ultimately enforced in the code by integrating to produce a prescribed displacement as

$$u(t) = \text{scale factor} * \left[\int_{t_{\text{start}}}^{t} \left(\int_{t_{\text{start}}}^{t} f(t)dt \right) dt + (t - t_{\text{start}}) * v_0 + u_0 \right]. \tag{8.1.1}$$

The start time of the function, t_{start}, is *not* the start time of the analysis. Accounting for this is important in hand-off analyses. A function is required; not listing a function will generate an error message. 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 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 for `p0` which specifies the initial acoustic pressure, since only the first time derivative of acoustic pressure is specified.

In the case of prescribed voltage acceleration, the descriptors `disp0` and `vel0` are used to define, respectively, the initial voltage and initial time derivative of voltage. By default `disp0`, `vel0`, and `p0` vanish.

In the case of a prescribed voltage time history, the prescribed value is the second time derivative of the voltage, i.e., voltage acceleration (`accelV`). However, since the first and second time derivatives of voltage do not contribute to the equations of motion (see the Theory Manual for details) a prescribed voltage 'displacement' option should typically be used (See 8.1.2).

When prescribed accelerations are used, they induce a load on the structure. Thus, a **loads** section, even if it is empty, must be present to use the prescribed acceleration capability. An error message will be generated if the input file has no **loads** section.

- Prescribed accelerations do not work with multicase solutions.
- They can only be applied in the standard Cartesian coordinate system.
- The nodes involved in prescribed accelerations cannot coincide with nodes that are involved with MPCs.
- Prescribed accelerations can be used in meshes that have nonlinear or viscoelastic elements only if the prescribed accelerations are *not applied directly to the nonlinear or viscoelastic elements*.

8.1.5. *Prescribed Frequency-Varying Displacements*

For the direct frequency response solution method, a portion of the model may be prescribed displacements as a function of frequency. As shown in Table 8-1 the descriptors for prescribed frequency dependent displacements are `DispX`, `DispY`, `DispZ`, `RotDispX`, `RotDispY`, `RotDispZ`, `FreqP`, `FreqV`. A function must be used to apply the frequency dependent boundary condition.

8.1.6. *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.

Only pressure waves need to be absorbed for acoustic elements, and the absorbing boundary could represent an infinite fluid surrounding a structure. For elastic waves both pressure and shear waves need to be absorbed, and the absorbing boundary 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 large radius (the radius is 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.

8.1.7. 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 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 impedance 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 the multiplier on the characteristic impedance ρc. Similarly, for the elasticity element the pressure and shear impedance 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 large number, the impedance boundary condition reduces to a rigid-wall condition for acoustics, and a fixed condition for elasticity elements.

8.1.8. Slosh

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 contributes to the mass matrix, resulting 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.

8.1.9. Infinite Elements

In this section, we describe how to use infinite elements for acoustics. These elements serve as both high-order absorbing boundary conditions, and far-field calculators that 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 an ellipsoidal surface or part of an ellipsoidal surface. Thus, a full spherical surface, hemispherical surface, or a quarter of a sphere would all be acceptable. Infinite element accuracy will degrade if the element surfaces on the boundary do not adequately represent the ellipsoidal surface. The finite element surfaces will be faceted, but enough elements on the boundary are needed to represent the ellipsoidal curvature.

Once a sideset is identified for the infinite element surface, the `boundary` section in the input deck would be modified as follows.

Parameter	Description	Options	default
radial_poly	the type of polynomial for radial expansion	Legendre Lagrange Jacobi	Legendre
order	the order of the radial basis	0-19	0
source_origin	the origin of the ellipsoid	3 reals	0 0 0
ellipsoid_dimensions	radial dimensions of ellipsoid axes	3 reals	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
use plane line	intersect method		

Table 8-2. – Available parameters for the infinite element section.

```
BOUNDARY
  sideset 1
    infinite_element
    use block 57
END

BLOCK 57
  infinite_element
  radial_poly = Legendre
  order = 5
  source_origin = 0 0 0
  ellipsoid_dimensions 15 15 30
  neglect_mass = yes
END
```

where block 57 contains the infinite element parameters. The number 57 is arbitrary; the user can pick any number (or name) that is not assigned to a block in the input mesh (Exodus) file. The parameters are summarized in Table 8-2. Currently, only Legendre polynomials are available for the radial basis. 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 ellipsoid that the infinite elements emanate from.

The ellipsoid_dimensions parameters indicate the axial dimensions of the ellipsoid in the global coordinate system. They are specified as ellipsoid radii instead of ellipsoid diameters. In the case of a sphere, all 3 parameters are equal and 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. By default, **neglect_mass** is yes. 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. Neglecting the mass, `yes`, is recommended in most cases.

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.

The infinite element formulation in **Sierra/SD** uses a Petrov-Galerkin formulation, instead of 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 and frequency domains (i.e. **Direct FRF**). 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.

8.1.9.1. Far-Field Postprocessing

The infinite element formulation allows the analyst to compute the response outside of the acoustic mesh as a post-processing step. The response can be computed at any point outside the mesh, and for any time interval. Currently, the **linesample** capability is used to write out the far-field data (see Section 9.8.9). 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 `displacement` .

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 Manual 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,:))
```

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.

8.1.10. Perfectly Matched Layers

Perfectly Matched Layer (PML) elements enforce acoustic baffle boundary conditions. A detailed explanation of theory and implementation of our PML formulation is available.[14] These elements serve as an absorbing boundary condition for outgoing acoustic waves, much like infinite elements. Unlike infinite elements, they are linear elements, and do not exhibit the large matrix condition numbers and convergence issues that can accompany infinite elements. Table 8-3 summarizes the parameters.

While PML are a separate block of elements in the finite element boundary, in an input file they are treated like a boundary condition. The **boundary** section is set up as follows:

```
BOUNDARY
  sideset 1
  pml_element
  use block 217
END
```

The user picks a number (or name) that is not already assigned one of the other element blocks. Here the user chose 217.

```
BLOCK 217
  pml_element
  pml_thickness 1
  stack_depth 1
  source_origin = 0 0 0
  ellipsoid_dimensions 15 15 30
  loss_function = polynomial
  loss_params 0 960 960 0
END
```

And here, PML elements will be in block 217 of the output mesh.

For PML, a loss function is a definition of the rate of decay of the outgoing wave. While the choice of the loss function $\sigma(d)$ is discussed in the literature,[10,11,36] papers in the literature use a range of formulations and implementations, and it is still unclear what the best choice is for any given problem. Typically, the loss function starts at a low value (often zero) to minimize numerical reflections, and increases at an increasing rate to maximize the loss terms near the outer boundary. One option is the polynomial loss function, that includes the constant, linear, quadratic,

Table 8-3. – PML Element Parameters.

Parameter	Description	Options
pml_thickness	length of PML extrusion from boundary	Real
stack_depth	number of elements through PML thickness	Integer
source_origin	the origin of the ellipsoid	3 real numbers
ellipsoid_dimensions	radial dimensions of ellipsoid	3 real numbers
loss_function	type of function describing PML decay	singular or polynomial
loss_params	constants in loss function	4 real numbers

and cubic terms, with four parameters that define the loss function

$$\sigma(\xi) = c_1 + c_2\frac{\xi}{t} + c_3\frac{\xi^2}{t^2} + c_4\frac{\xi^3}{t^3} \tag{8.1.2}$$

where c_1, c_2, c_3, and c_4 are specified in the input file, ξ is the distance along the normal from the Gauss point to the inner ellipsoid boundary, and t is the total thickness of the PML layer. The loss function is normalized such that changing the thickness of the PML layer does not change the maximum value of σ. Another option is the singular loss function.[11] (equation (8.1.3)), which is unbounded at the outer boundary.

$$\sigma(\xi) = \frac{c_1}{t - \xi} \tag{8.1.3}$$

PML elements can be extruded from either Tet4 or Hex8 meshes. Note that Tet4 is the default; for PML elements extruded from Hex8 meshes, it is necessary to specify `Hex` by modifying the **boundary** section as follows:

```
BOUNDARY
  sideset 1
  pml_element
  use block 217
  hex
END
```

8.1.10.1. Limitations

PML is only supported for certain formulations and element types. The implemented PML formulation only applies in the frequency domain, and will throw a fatal error for frequency or time domain simulations. Additionally, PML is only supported for Tet4, Wedge6, and Hex8 elements, which limits the exterior of the acoustic domains to those element types.

8.1.11. *Periodic Boundary Conditions*

Periodic boundary conditions can be applied in Sierra-SD using **begin-periodic** block. Built on the tied data algorithm, periodic boundary conditions are currently supported for structural/solid surfaces, with the displacement on the side B surface specified relative to the displacement of the side A surface. The A and B surfaces within a begin-periodic block may not share nodes, but surfaces across various begin-periodic blocks can share nodes (e.g. standard representative volume element (RVE) model would have three begin-periodic blocks, with the surfaces in each block sharing nodes with surfaces in other blocks, at the RVE edges). The surfaces can be curved, but the user must ensure that they have the same geometry, translated in space. Meshes on the two surfaces need not match. Matching meshes are however recommended whenever feasible, to ensure accuracy of the computed stresses on the surfaces. See discussion in Tied Surfaces section above regarding inaccuracies in stresses at the tied surfaces with mismatched meshes. The underlying algorithm results in non-homogeneous MPCs from a node-face constraint algorithm. Naturally, for matching meshes, these MPCs reduce to node-to-node MPCs.

Like a `tied data` block, each begin-periodic block represents a *single* pair of opposing sides connected by periodic boundary conditions, as shown in the example below.

```
BEGIN-PERIODIC
  side A = 12
  side B = 18
  name "PBC-x-12-18" // used as a descriptor for the output
  geometric offset = 10.0 2.0 3.0
  search tolerance = 1e-7
  Ux =  0.5
  Uz =  0.25
END
```

```
BEGIN-PERIODIC
  side A = 1
  side B = 6
  name "PBC-y-1-6" // used as a descriptor for the output
  search tolerance = 1e-7
  Ux =  0.5
  Uy = -0.5
END
```

In defining surfaces, care must be exercised to *ensure that the normal vectors of the two surfaces point towards each other*. The keyword “geometric offset” represents the distance from the A surface to the B surface in x, y and z directions. Similar to the TIED DATA block, the keyword “search tolerance” represents the normal distance from a node on the B surface, shifted by the geometric offset, to the face on the A surface. See Figure 10-2 for further details. The keywords “Ux,” “Uy” and “Uz” represent the displacement of the B surface relative the A surface in x, y and z directions respectively. The description of all the parameters are shown in Table 8-4.

8.1.11.1. Usage Guidelines

The user is referred to the verification manual for examples of periodic boundary conditions(PBC), one with unidirectional PBC and the other with multi-directional PBC, and their application to simulating periodic volume elements (PVE). Here, we provide some guidelines and cautionary remarks associated with PBC and PVE.

1. *Matching* meshes of the pair of sidesets with the PBC are preferred, but not necessary. One trick for generating matching meshes for symmetric PVEs (with deterministic microstructure) is to split the PVE using plane(s) of symmetry and meshing one part and mirroring the meshed volume to get the complementary volume.

2. *Non-matching Meshes:* When the microstructure is random or lacks symmetry, PBC may need to be applied to connect non-matching meshes. Such situations may encounter local oscillations in stresses on the surface, associated with the underlying node-face contact strategy. These oscillations are expected to decay quickly going into the volume, making the volumetric average more accurate. Since micromechanics modeling often involves average stresses, the error in homogenized global material properties is expected not to be significant provided that the representative volume element (RVE) is large enough (which will also be a requirement from the standpoint of representing random microstructure).

3. *Geometric Offset:* Theoretically, geometric offset should not be needed in the periodic-boundary block. However, whenever the opposite faces have non-matching meshes, there can be errors in automatic computation of offset, which involves centroid computation of the sideset that may involve discretization errors. Given this, whenever feasible, it is recommended that the geometric offset be provided in the input. Note that this comment applies only to non-matching meshes; automatic computation of geometric offset would not have any errors for matching meshes.

4. *Imposition of Homogenized Strain:* The user is referred to the verification manual (problem on PVE) for details of imposing global strain tensor through multi-directional PVE.

Table 8-4. – Parameters for Periodic Boundary Conditions.

Parameter	type	description
Name	*String*	name of periodic boundary condition block defaults to *periodicBC*
side A	*Integer*	sideset A id
side B	*Integer*	sideset B id
Geometric Offset	*3 Reals*	x,y,z components of B sideset location relative to a sideset automatically computed if not explicitly specified, explicit specification is recommended for non-matching meshes)
Search Tolerance	*Real*	search tolerance normal to the face default 1.0e-8
Ux	*Real*	x-displacement of B sideset relative to A sideset default value is 0.0
Uy	*Real*	y-displacement of B sideset relative to A sideset default value is 0.0
Uz	*Real*	z-displacement of B sideset relative to A sideset default value is 0.0

8.2. Exodus Mesh Boundary Condition Input

Several boundary conditions may be set to values specified in the mesh geometry file or input sources. This is used to **hand-off** loads an earlier finite element analysis, even a previous **Sierra/SD** simulation. The pressure determined in another code, and output to a sideset and can put input on the sideset, determining the right-hand side in a **Sierra/SD** analysis.

The **Exodus** file input data is defined at one or more time slices that do not necessarily correspond to **Sierra/SD** time steps. Figure 8-1 summarizes the rules for interpolating the input time slides to **Sierra/SD** time steps.

The value of a boundary condition at a time . . .

- . . . before any input **Exodus** time steps is the initial input value.
- . . . between two input **Exodus** time steps is determined by linear interpolation.
- . . . after the last input **Exodus** time step is the last input **Exodus** time step.

An example of interpolating Exodus data do **Sierra/SD** time steps is shown in Figure 8-1.

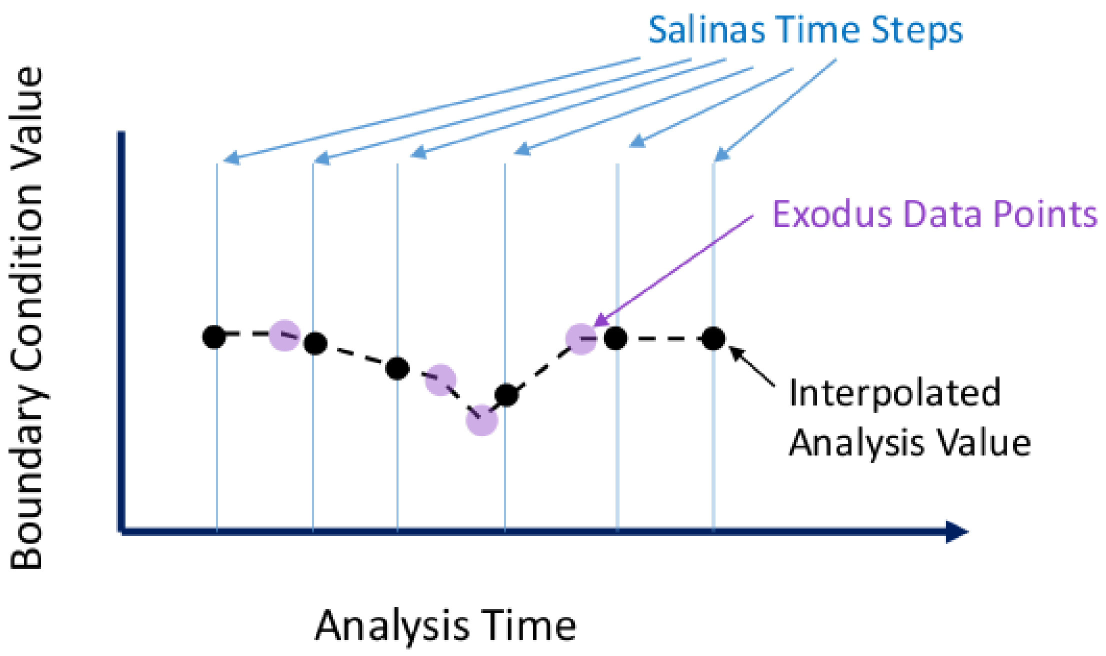


Figure 8-1. – Example of Interpolation of Exodus Data to Analysis Steps.

8.2.1. *SpatialBC Functions*

This appears to be a capability that was added for a specific purpose that was mistakenly exposed to users.

The Spatial boundary condition function is used to set a boundary conditions from the input **Exodus** mesh geometry file. It resembles the `randomlib` function 4.10.8. The difference is that the randomlib function uses a nodeset associated with the specified sideset. With Spatial boundary condition, the nodeset is specified directly.

The variable input from the **Exodus** file is specified with the parameter `exo_var`, followed by either `scalar` or `vector`, sets the variable input from the **Exodus**.

In the example the input scalar is acceleration in the Z direction:

```
BOUNDARY
  nodeset NS_top
  function from_mesh
  accelz = 1
END
FUNCTION from_mesh
  type = spatialBC
  nodeset NS_top
  exo_var scalar Acc_Z
END
```

Here “Acc_Z” is the exact name of a field on the Exodus nodeset. Exodus field names are case-insensitive.

Sierra/SD has only a couple regression tests of SpatialBC, and each test uses exactly the same boundary condition. Each results in a confusing and misleading warning about missing sidesets. The vector input feature is completely untested.

8.2.2. ExodusRead functions

A `ExodusRead` function reads in data from either the entire **Exodus** file, or a nodeset, sideset, or block that covers an area 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 variable is read from the entire mesh as a nodal or element variable (rather than a nodeset or sideset variable). For backwards compatibility the `ReadSurface`, `ReadNodal`, `ReadNodalSet`, and `NodalForce` functions are all treated as an `ExodusRead` function.

Loads such as pressures or tractions are integrated over the surface while forces are applied directly to nodes.

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 or traction loads on a structure. For this case, the load output variable currently only outputs element data for values read from a sideset. Nodal values, like those used in `randomlib` functions, are output as 0. There is 1 and only 1 test of read surface with a traction load.

As currently implemented, the ExodusRead function operates only by reading data from an external **Exodus** data file. The name of the variable to read 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 input 8.8. Pressures require a scalar variable, and tractions require a vector variable. An example for `ExodusRead` functions is given in input 8.8. A **sideset** matching the corresponding Load sideset is required. The `interp` selected the temporal interpolation algorithm as was described for the `randomlib` function. The default option, linear interpolation, is the only option available.

In input 8.8, the keyword `exo_var` specifies the type of data, such as `vector` or `scalar`, and the name of that variable in 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 input 8.8, a vector nodal variable with a base name of name ‘traction_load’ should be available in the **Exodus** file. Thus, the data in the **Exodus** file would have names `traction_loadX`, `traction_loadY`, and `traction_loadZ`. In the case of scalar data, the base name given (i.e. `traction_load` in input 8.8), should match exactly the name of the nodal variable in the **Exodus** file.

```
LOADS
  sideset 1
    traction 1 1 1
    function 55
END
FUNCTION 55
  type=ExodusRead
  interp=linear
  exo_var vector traction_load
END
OUTPUTS
  velocity
END
```

Input 8.8. Example ExodusRead Traction Specification

```
LOADS
  block 1
    point_volume_accel = -1.0
    function = 77
END
FUNCTION 77
  type=ExodusRead
  exo_var scalar dd_vol
END
```

Input 8.9. Example ExodusRead Point_Volume_Accel Specification

ExodusRead functions ignore distribution factors when used to apply loads. See section 8.3.19.

In addition to reading data from the geometry file, `ExodusRead` may also be used to read pressure loads from a separate source exodus file using the `copy variable` interface of a **transfer** section (section 4.3.2 and input 4.4). Note that a pressure boundary condition (or load) may only be applied on a side set.

Furthermore, the load data may also be interpolated in time to match the **Sierra/SD** time step. A linear interpolation is performed by default for the time step. If the nearest time step match is desired, set `interpolation = none` in the function definition. If there is only one time step on the source mesh, then that value is used for all time in the destination **Sierra/SD** run. Extrapolation is not performed if the target time is not within the time interval of the source mesh. Instead, the closest valid time on the source mesh is used.

8.2.2.1. Input an Acoustic Point Source from a Volume

There are 3 tests that exercise the capability to input a scalar acceleration from the whole mesh (body) or an element block. In acoustic point source analysis volume velocities may be input from an **Exodus** file using a `ExodusRead` function. In theory, the **Exodus** file contains the first or second derivatives of the volume velocity at the corresponding times. The name of the field input from the **Exodus** file is specified with the `exo_var` keyword. Table 8-5 lists the run time parameters for `exodusread` functions. An example is provided in input 8.10. The keyword `exo_var` must be followed by two keywords, specifying first whether the data is a scalar or a vector, and second specifying the name of the variable on the **Exodus** database. Only scalar variables are supported for point volumes. Thus, Table 8-5 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 in Section 4.10.8. It specifies the type of temporal interpolation.

`ExodusRead` determines what, if anything, to read by accepting the first match found. One after another, it searches for 3 things, and errors out if the last search come up empty. The first search is for a nodal variable with the specified name (i.e.,**volume_acceleration** in the example above). Next it looks for an element variable with the same name. Finally, it searches for a face variable.

Table 8-5. – ExodusRead function parameters.

Keyword	Values	Description
type	ExodusRead	required to specify function

interp		temporal interpolation scheme
		none=nearest
		linear=linear interpolation

exo_var	scalar|vector <name>	Exodus variable type and name

```
FUNCTION 55
  type=ExodusRead
  interp=none
  exo_var scalar volume_acceleration
END
```

Input 8.10. Example ExodusRead Function Specification

8.2.3. *Input an Acoustic Point Source from a Node Set*

Input volume velocities for acoustic point source analysis from the input **Exodus** mesh geometry file with the `ExodusRead` function. Velocities are input at each time specified in the **Exodus** file. Spatially dependent velocities are input at each nodeset node.

If a nodeset is used in the exodusread function this nodeset must match the definition in the load section.

```
FUNCTION 55
  type=ExodusRead
  interp=none
  nodeset=NS_top
  exo_var scalar volume_acceleration
END
```

Input 8.11. Example ExodusRead Function Specification

8.2.4. In-Core Transfer Functions

A limited capability exists to perform in core transfer of variables between physics codes via MPI messages rather than **Exodus** files. In this use case both codes are run simultaneously with the results from the load producing code constantly being fed to the load using code. The main use case for this capability is to **hand-off** boundary conditions from one code to another without the need for huge intermediate **Exodus** files with a large numbers of steps.

8.2.4.1. Transfer from Fuego

One field that may be transferred in core is the divergence of Lighthill's tensor from Fuego to **Sierra/SD**. This is a nodal value that can be used as an acoustics load term. An example of the relevant input for this case is:

```
PARAMETERS
  mpmd_transfer_type = fuego
END
LOADS
  nodeset = 1000
  Lighthill = 1.0  #Sets load type and scale factor
  function = from_fuego
END
FUNCTION from_fuego
  type transfer
END
```

An example of the job execution syntax for this case is:

```
$ mpirun -n 1 fuego -i Fuego.i : -n 1 salinas -i Salinas.inp
```

8.2.4.2. Transfer from SPARC

Another field that may be transferred in core is the traction load from SPARC to **Sierra/SD**. This is a face-based term that can be used for structural loads. An example of the relevant input for this case is:

```
PARAMETERS
  mpmd_transfer_type = SPARC
  mpmd_transfer_sidesets = 5
END
LOADS
  sideset 5
  traction = 1.0 1.0 1.0 #Sets load type and scale factors
  function = SPARC
END
FUNCTION SPARC
  type transfer
END
```

An example of the job execution syntax for this case is:

```
$ mpirun -n 1 sparc -i SPARC.i -c 0 : -n 1 salinas -i Salinas.inp
```

8.3. Specific Load Types

The input deck syntax for loads is presented in the General Commands section 4.5

Section	Keyword	Parameters
Region *(defines application area)*	body nodeset sideset block node_list_file	- *id/name* *id/name* *id/name* *file name*
Load Type *(defines application method)*	force moment gravity pressure point_volume_vel point_volume_accel acoustic_vel acoustic_accel Lighthill surface_charge traction thermal_load energy_load	*val1 val2 val3* *val1 val2 val3* *val1 val2 val3* *value* *value* *value* *value* *value* *value* *value* *val1 val2 val3* - -
optional specifications		
Coordinate Frame *(for vector loads only)*	coordinate	*id/name*
Scale Factor Multiplier	scale	*val1*
Function (Required for *transient analysis)*	function	*id*
`follower`	follower	*yes/no*

Table 8-6. – Load Specification Keywords.

8.3.1. Pressure

The **pressure** load type may only be applied to a sideset. Total forces are defined by integrating the pressure load over the faces. Pressure forces are always in the direction of the face normal. Positive pressures values act in a compressive sense on the surface.

By default, pressure loads are not follower loads, i.e. pressures are applied based on the area and normal of the undeformed elements for the entire simulation. The `follower` keyword may be applied to recalculate the area and normal of the faces each time step. See section 8.3.2 for follower stiffness specification.

The loads for every 3-D and 2-D element is calculated consistently when a pressure load is applied. For more details on the implementation, see the programmer's notes. It is important that consistent loading be used. This is especially true for shell elements where the consistent loading is required to properly apply rotations.

8.3.2. *Follower Stiffness*

The follower stiffness that corresponding to an applied pressure load may be included. A follower pressure load applied to a structure 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, 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
```

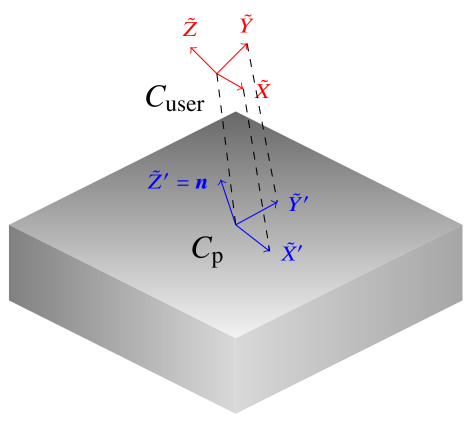
In the above example, sideset 1 is used to denote the surface where the pressure is applied. The parameter "pressure" specifies the magnitude of the applied pressure field.

Similar to other load-based stiffness modifications such as gyroscopic loads the same follower stiffness contribution will be used for all solution cases. The first load that defines a follower stiffness will be used for follower stiffness in all cases. Any subsequent follower loads will be ignored for left-hand side contributions.

8.3.3. *Traction*

The **traction** load type may only be applied to a sideset. Total force are defined by integrating the traction load over the face. Traction loads can be in an arbitrary direction with respect to the face. By default, tractions are applied based on the global XYZ coordinate frame.

If the analyst provides a coordinate frame with the traction definition, then that frame is projected onto the surface of each element. Figure 8-2 illustrates that projection. Note that when a coordinate system is used, there can be a mesh dependence on which direction the forces are applied. Note that the third coordinate of the traction will always be applied along the surface normal, and that the third component of the vector will always correspond to the surface normal (and hence will be applied as a pressure).



Consider a user defined coordinate frame, C_{user} defined by the basis vectors,

$$(\tilde{X}, \tilde{Y}, \tilde{Z})$$

A surface normal, $\boldsymbol{n}$, is defined by the element normal. The user defined coordinate frame is projected onto the surface as follows.

$$\begin{aligned} \tilde{Z}' &= \boldsymbol{n} \\ \tilde{X}' &= \tilde{Y} \times \tilde{Z}' \\ \tilde{Y}' &= \tilde{Z}' \times \tilde{X}' \end{aligned}$$

This transformation is singular when $\tilde{Y} \times \boldsymbol{n}$ is zero. Near that location, the transformation is modified.

$$\begin{aligned} \tilde{Z}' &= \boldsymbol{n} \\ \tilde{Y}' &= \tilde{Z}' \times \tilde{X} \\ \tilde{X}' &= \tilde{Y}' \times \tilde{Z}' \end{aligned}$$

When normalized, $(\tilde{X}', \tilde{Y}', \tilde{Z}')$ form the basis for a coordinate frame, C_{p}, on the surface of the element in which to apply the tractions.
Note: This transformation is dependent on the direction of the element normal, $\boldsymbol{n}$. If $\tilde{Z}$ is in the opposite direction of $\boldsymbol{n}$, the $\tilde{Y}$ direction will be preserved, but the $\tilde{X}'$ direction will be the opposite of $\tilde{X}$. This preserves the right-hand rule.

Figure 8-2. – Coordinate Frame Projection for Tractions

8.3.4. Acoustic Velocity and Acceleration

The **acoustic_vel** and **acoustic_accel** boundary conditions can only be applied to a sideset. Acoustic boundary conditions may only be used with acoustic elements and materials.

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 = velocity_function
END
```

In this case, sideset 1 is given a prescribed normal velocity of amplitude 1, with a time dependence given by function `velocity_function`.

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. For coupled structural acoustics, only the **acoustic_vel** keyword may be used. For analysis involving only acoustic elements, either keyword can be used.

The sign conventions of the **acoustic_vel**, **acoustic_accel** keywords are important. For the **acoustic_vel** and **acoustic_accel** cases, the equations of motion are given by,

$$\frac{1}{c^2}\ddot{p} - \Delta p = -\int_\Gamma \rho q(a \cdot n)d\Gamma \tag{8.3.1}$$

or, in discrete form,

$$M\ddot{p} + Kp = f \tag{8.3.2}$$

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. These makes sense, since in this case mass will be added to the acoustic space, causing an increase in pressure.

8.3.5. Acoustic Point Volume

point_volume_vel and **point_volume_accel** 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 **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. 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. 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.

The sign conventions of **point_volume_vel** and **point_volume_accel** keywords are important. For the **point_volume_vel** and **point_volume_accel** boundary conditions, the equations of motion are given by

$$\frac{1}{c^2}\ddot{p} - \Delta p = \rho \frac{\partial^2 V}{\partial t^2}\delta(x - x_0) \tag{8.3.3}$$

or, in discrete form,

$$M\ddot{p} + Kp = f \tag{8.3.4}$$

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

$$\frac{\partial^2 V}{\partial t^2} = V_{,tt}$$

A positive $V_{,tt}$ would result in a positive acoustic pressure, implying that fluid mass is added to the surrounding acoustic space. Conversely, if $V_{,tt}$ is negative, mass will be subtracted from the acoustic space, and thus a negative acoustic pressure will result.

The previous example involved spatially constant functions of time. Acoustic boundary conditions with spatially-varying functions of time are supported through the `Exodus Read` function as described in section 8.2.2 respectively.

```
LOADS
  nodeset 1
  point_volume_accel = 1.0
  function = accelFunc
END

FUNCTION accelFunc
  type LINEAR
  name "volume_acceleration"
  include inc/volume_acceleration.inp
END
```

Input 8.12. Example Input for Point Acoustic Load. 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.

8.3.6. Lighthill

The **Lighthill** load may only be applied to acoustic elements. Keyword **Lighthill** prescribes the divergence of the Lighthill tensor at nodes. The double divergence of the Lighthill tensor is a source term for noise generation in the pressure formulation of acoustics. And the divergence of the Lighthill tensor is a time varying vector quantity can be applied as a nodeset load using the Exodus Read function. A Lighthill load is only implemented for the transient simulations. It is only valid for the pressure formulation of acoustics and can only be used with **acoustic_accel** loading. The **outputs** keyword **acousticlighthill** outputs the acoustic Lighthill source term.

```
OUTPUTS
  acousticlighthill
END
```

Input 8.13. Example of Lighthill output.

8.3.7. Thermal

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.

With a thermal load, a **statics** simulation has advantages over a **transient** simulation. In the transient simulation, a dynamic preload is computed that will oscillate about the solution of the thermally loads statics problem. If damping is used, this dynamic preload will converge to what would be obtained from using a **statics** 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 uniform temperature distributions, it is more efficient to specify them block-by-block in the input deck. If temperature is specified in multiple places, the order of precedence is block temperature, nodal temperature, centroid temperature, and finally Gauss point temperature. Note, nodal and Gauss point temperatures are later interpolated to element centroid temperatures that are used for calculation.

Currently, the **thermal_load** only affects the computation for solid elements and hex shells. For most element types, a warning will be issued, but for 1D element types that do not have a material section, they will be silently ignored.

Output stress and strain

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 the *mechanical* stress, rather than the combined thermal-mechanical stress.

There is a known bug in the way that thermal stresses are computed, particularly when temperature comes from the **Exodus** file. If thermal stresses are needed, then extreme care should be taken.

Mechanical stress is the same as elastic stress. `Strain`, `elastic_strain`, and `thermal_strain` output are all available.

Input deck syntax

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 25
  material 2
  T_current 200
END
```

Note that if $T_{current}$ is specified for some blocks and not for others, the code issues an error.

For thermal statics or thermal transient analysis, each material block must be given two additional parameters, the reference temperature, T_{ref} = `Tref`, and the coefficient of thermal expansion, α_t =

`alphat`. When temperatures are specified in the `block` or read in from the **Exodus** file, the material properties can also be specified as temperature-dependent. For a detailed description and examples of material thermal property and temperature-dependent property definitions, refer to section 6.5.6.

The default **Exodus** file labels for temperatures are shown in the table below. This is the default variable format for **Sierra/SD**. 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 (4.4) allows specifying the name of the temperature variable in the **Exodus** file. A nodal variable name is expected. **Sierra/SD** will fall back to an element variable if no nodal variable of the specified name is found. If neither an element nor nodal variable of the given name is found, a fatal error occurs.

The **thermal_load** command cannot be used with **energy_exo_var**. Consult section 8.3.8 for guidance on using **energy_exo_var**.

Name	Definition
`TEMP`	the nodal temperature

The **thermal_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.

When temperature is read from the **Exodus** input mesh using either the default temperature file name or the **thermal_exo_var** keyword a constant temperature can be read from a single time step with the keyword **thermal_time_step** in the parameters block. Alternatively a time variant temperature can be updated periodically from the mesh file with use of the keyword **nUpdateTemperature** in the solution case input. The **nUpdateTemperature** keyword defines how often (i.e., every n steps) temperature should be updated. In a transient simulation, updating the temperature from the **Exodus** file can be expensive; thus, it may be advantageous to update only at a larger interval, if temperature is evolving slowly. When reading transient temperature data from the input mesh the closest **Exodus** time step at or below the current time step is used.

Unlike several other **Exodus** input capabilities in **Sierra/SD**, temperature input does *not* currently support interpolation between **Exodus** time steps.

When a new temperature value is read, that temperature is used immediately to update the applied thermal strain. In the case of temperature-dependent material properties, the **nUpdateDynamicMatrices** keyword in the solution case can also be used to update material properties (see section 6.5.6).

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 **nUpdateTemperature** keyword is placed in the solution case to specify how often to update the temperature that is read in.

```
SOLUTION
  transient
  nUpdateTemperature 5
END
```

Here a new temperature is read in every 5^{th} time step. If the transient solution specifies the last time step from the thermal analysis, then the final temperature will be used.

The next example presents input for thermal statics analysis.

```
SOLUTION
  statics
END

PARAMETERS
  thermal_time_step 10
END

LOADS
  body
    thermal_load
END
```

8.3.8. Energy Deposition

Input from energy deposition are similar to thermal loads (section 8.3.7). 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 6.5.8).

$$\tilde{E} = C_v \Delta T.$$

$\tilde{E}$ 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 advantageous to specify the loads at Gauss points, particularly when using higher order elements, even though loads will be interpolated to the element centroids. Energy loads can also be specified at nodes or centroids. Nodal energy loads should be avoided because it is not clear which materials specific heat to use when converting an energy load to a temperature change for nodes on the interface between two materials. For this case **Sierra/SD** will use the specific heat of the material that is processed last (which is not typically what is required by the analyst). **Sierra/SD** generates a fatal error if the specific heat is not specified for a material with an energy load.

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.

Using **thermal_exo_var** with **energy_load** generates a fatal error. Consult section 8.3.7 for guidance on using **thermal_exo_var**.

8.3.9. Force

Point force loads may be defined with the **force** load. Note, the load is applied at each and every node in the nodeset, sideset, block, or other application region. Thus, force loads are typically applied as point loads on a conmass or other point entity. Force loads on mesh constructs that contain many nodes such as sidesets is allowed but discouraged. To apply a known force distributed across a sideset or other large application area a RBE3 should typically be employed.

8.3.10. Moment

Point moment loads may be defined with the **moment** load. Note, the load is applied at each and every node in the nodeset, sideset, block, or other application region. Thus, moment loads are typically applied as point loads on a conmass or other point entity. Moment loads on mesh constructs that contain many nodes such as sidesets is allowed but discouraged. Additionally, the point moment loads are only meaningful on nodes that have an underlying rotational degree of freedom, such as a conmass with inertia or a node of a shell or beam. If a moment is applied to a node connected only to solid elements without underlying rotational degrees of freedom that load will have no effect or will generate an error. To apply a known net moment to a sideset or other large application area a RBE3 should typically be employed.

8.3.11. Pressure_Z

The **pressure_z** exists to simplify the common case of depth-dependent pressure loading. An example is shown in input 8.14. 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 input 8.14, the pressure is zero at $x = 5$, 10 at $x = 4$, 20 at $x = 3$. 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
    pressure_z 1e-4 above x = 5
END
```

Input 8.14. Depth Dependent Pressure Load Example. This load section applies a pressure to sideset 2 which is proportional to the distance below $x = 5$.

8.3.12. Random Pressure

A hypersonic vehicle is a prime example of a random loading. The correlations of a turbulent pressure boundary condition are low in time and complicated in space.

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. A linear solve at each step is required to compute the loads.

A correlation matrix is the inverse Fourier transform of the spectral density matrix. It is most general type of input. Load option **RandomPressure** provides a simplified means of specification of the loading. The material in this section is consistent with and builds on section 4.10.7.

Subsection Random Pressure Loading section Loads and Materials of the Theory Manual[45] describes the approximations involved in the implementation. These approximations are summarized in Figure 8-3.

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 a 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 8-3. – RandomPressure Loading Approximations.

The random loading is a component of the loads section. An example is shown here, and described in Table 8-7.

```
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
```

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	string	$\frac{\sin(t\omega_c)}{t\omega_c}$	
PSD_scale_function	string	$\Sigma(z) = 1$	
NTimes	int	5	
coordinate	string	0	defaults to basic frame
MinimumNodalSpacing	real	$1.e - 8$	smallest allowable inter-node spacing
Random_Seed	int	ignore	random number seed
NumberOfInitializationSteps	int	5	iterations to improve initial spatial distribution

Table 8-7. – Random Pressure Inputs.

Details for the parameters to the correlation matrix input are described below.

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 important to the operation of the RandomPressure 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 the Theory Manual section Loads and Materials subsection Alternative Derivation Based on Lagrange's Equations subsubsection Separation of spatial and temporal components

Typically, few 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 $\tilde{Z}$ 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 Initially the pressure spatial distribution may be too correlated. Mesh resolution exacerbates this issue. Increasing the `NumberOfInitializationSteps` mitigates the issue. Each initialization step requires about as much CPU time as an implicit time step. Default=5 (values below 5 are discouraged).

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 (subsection Random Pressure Loading section Loads and Materials of the Theory Manual[45]).

However, the Cholesky matrix factorizations are defined only if the correlation matrix is (numerically) non-singular. 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 large correlation lengths correlation_length_z or correlation_length_r, or a 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, a Rcond below 10^{-12} indicates that the factorization may fail. The precise statements in the results files are

```
TemporalCorrelationMatrixRcond = ...
Estimated SpatialCorrelationMatrixRcond = ...
Estimated CorrelationMatrixRcond = ...
```

8.3.13. Surface Charge

For electro-mechanical coupled materials such as dielectric or piezoelectric materials, surface charges may be applied to a specified sideset using the keyword **surface_charge** in the loads block. The surface charge is mechanically analogous to a pressure, where surface charge represents a charge per unit area and is applied only to the voltage degrees of freedom. Surface charges can be applied for direct frequency response and transient solution methods. An example is provided below.

```
Loads
     sideset 1
     surface_charge = 1    // scalar multiplying the specified function
     function 1            // surface_charge function
END
```

Input 8.15. Surface Charge Example

8.3.14. Gravity

The **gravity** load is used to apply a mass proportional load to the structure. A typical use is for the gravitational force acting on a structure. The gravity load can also be used to apply the pseudo force on a body in an accelerating frame. Additionally, gravity can be applied in a spatially varying way via either a **ExodusRead** function or analytic functions as a general mass proportional body load.

Gravity loads are only allowed in whole mesh (**body**) regions.

8.3.15. Angular Velocity and Angular Acceleration

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, [1] Coriolis and Euler contributions. These are termed “forces”, but the contributions are introduced from operating in a non-inertial coordinate frame as described in the Theory Manual section Loads and Materials subsection Analysis of Rotating Structures.

The associated keywords are found in Table 8-8.

[1]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	*vel*1 *vel*2 *vel*3
angular_acceleration	*accel*1 *accel*2 *accel*3
coordinate	coordinate name

Table 8-8. – Rotating Frame Parameters.

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 structural dynamics, force-based matrices are also common. These include follower stiffness matrices from applied pressures, and Coriolis/centrifugal matrices in rotating structures.

Input 8.16 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**.

For static loads analysis angular acceleration and angular velocity are applied independently. A similar static loads analysis of a rocket provides envelope survivability information during launch.

```
LOADS
  body
    angular_velocity = 0.0 2.0 0.0
    coordinate = 1
  body
    angular_acceleration = 0.0 0.0 3.0
    coordinate = 3
END
```

Input 8.16. 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.

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.

$$M\ddot{u} + G\dot{u} + \left(K + K_g + K_a + K_c\right)u = f_a + f_c. \tag{8.3.5}$$

Symbol	Description
K_a	angular acceleration
K_g	geometric stiffness matrix
G	Coriolis matrix
K_c	centrifugal softening matrix

,

Symbol	Description
f_a	angular acceleration
f_c	centrifugal force

Table 8-9. – Notation for stiffness and damping matrices (left) and forces (right).

Except for K_g, every matrix term is constant, depending only on the geometry the elements, and the angular velocity or acceleration respectively. Coriolis and centrifugal terms are independent of displacement, u.

For readers missing the Euler stiffness matrices, here the term acceleration stiffness matrix is used instead.

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.

Even if the input deck selects multiple simulations, 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 input 8.17.

```
SOLUTION
  case s1
    statics
      load=1
  case up
    tangent
  case s2
    statics
      load=1
END
```

Input 8.17. Example using Tangent Update

Limitations There are some limitations for the rotational frames' implementation.

1. Statics analyses are limited to models with essential boundary conditions that eliminate all rigid body modes.
2. Statics 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.

3. In a single case solution, eigenvalues will include Coriolis, centrifugal and acceleration terms if angular velocity or acceleration is specified in the **loads** section. This is the case even though there is no true load for QEVP.
4. 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 input 8.18. 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.
5. One rotational frame is applied to the whole model. For example, if a helicopter is modeled in a fixed frame, then the associated rotor could only be modeled in this fixed frame.
6. Rotational loads applied in the rotating frame are linear loads, and do not require a follower keyword.
7. For transient dynamics, the time varying function must be 1.0.
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 left hand side terms applies to all cases in a multicase solution. If more than one angular velocity or more than one angular acceleration load is defined only the first load will be used for left hand side contributions.

```
SOLUTION
  case 'statics'
    statics
      load=1
  case 'up'
    tangent
  case 'qevp'
    QEVP
      method=projection_eigen
      nmodes=100
      load=1
END
```

Input 8.18. Example of using QEVP for Tangent Update

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.

Update the tangent matrix to enhance numerical accuracy, at the expense of greater computational expense.

8.3.16. Modal Force

Modal force loading can be used to directly load specific modes in the **modaltransient** solution case. An example input follows.

```
ECHO
  modalvars
END

LOADS
  body
    ModalForce
    function gravity_function
END

FUNCTION gravity_function
  type table
    tablename 28
END

TABLE 28
  dimension 2
  size 100 9
  delta 0.000005 1
  origin 0.000005 0
  datafile=Qforce.txt
END
```

The format of Qforce.txt is identical to the output from a traditionally loaded modal transient solution with `echo` **modalvars**. `echo` **modalvars** does not print out a load at time zero, so either the load at time zero should be added to the datafile, or `origin` should be used to specify the first time in the file. Loads can be linearly interpolated between time values in the datafile. With redundant modes, some modes can change between subsequent **Sierra/SD** runs. Care should be taken to ensure the load are applied correctly.

Full details of table input are discussed in section 4.10.19. Only two-dimensional tables are supported for modal force loading, with rows representing equally spaced time steps and columns representing each mode. Modal force loading is the only load type that supports two-dimensional table inputs.

8.3.17. *Applying Loads for Static Analysis*

Static loads only require the definition of the load region and load keyword (e.g. force) with its accompanying parameters. A function may be used instead. In this case, the function will be evaluated at time $t = 0$.

8.3.18. *Time Varying Loads*

Additional options provide the capability of varying the load over time. The **load** 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 4.10 on page 123). 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.

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.

8.3.19. *Temporal Loads from Exodus*

Loads may be read in from previous analyses when stored in the input **Exodus** data file. These are read using an 8.2.2 function.

Note that exodus read functions do not trigger follower stiffness calculations when used as follower loads. A warning is issued when the follower stiffness calculation is skipped.

Also, note that exodus read functions ignore distribution factors on sidesets. A warning is issued in this case when the distribution factors are ignored.

For Complex Analysis

Option	Parameters
iforce	*val1 val2 val3*
imoment	*val1 val2 val3*
igravity	*val1 val2 val3*
ipressure	*val1*
itraction	*val1 val2 val3*

8.3.20. 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 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.

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
```

8.3.21. Rigid Body Filter for Input

For some analyses, it is advantageous to remove rigid body components of a solution. The input forces may be filtered so that only self-equilibrated forces are applied. This process of removing the rigid body component from the solution is sometimes called 'Inertia Relief' or 'Inertial Relief'.

The rigid body filter is applied using input in the transient or statics solution section and is illustrated in the example of input 8.19. The filtered force values can be output by requesting the `force` output option (section 9.8.5).

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.

The **constrain_rbms** solver option must be used in conjunction with **FilterRbmLoad** *only* when the rigid body modes are in the null space of the system matrix. For example, **constrain_rbms** should be used for statics where the system matrix is the stiffness matrix, but **constrain_rbms** should not be used for transient where the system matrix includes inertial terms.

Currently, only **GDSW** supports selectively constraining rigid body modes. The **FilterRbmLoad** parameter is supported for transient and static solution cases. For other solution cases this parameter will have no effect on the solution. [2] The similar capability for modal solutions is presented in section 5.30.

```
SOLUTION
  statics
  solver=gdsw
  solver_options=gdsw_options
  FilterRbmLoad=allStructural
END

PARAMETERS
  RbmTolerance=1e-10
END

SOLVER_OPTIONS gdsw_options
  constrain_rbms "X Y Z RotX RotY RotZ p"
END
```

Input 8.19. Rigid Body Filter Example Input

The names of the rigid body modes to constrain are the same as those setting boundary conditions 8.1. 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. [3]

If the rigid body filter is activated, **Sierra/SD** 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]Modal solutions, such as `modaltransient`, do not use FilterRbmLoad. However, see[44] for means of accomplishing the same process by direct use of the geometry rigid body modes.

[3]The keywords are “x”, “y”, “z”, “RotX”, “RotY” “RotZ” and “p”. The set of these keywords must be enclosed in quotation marks.

8.3.22. 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	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 (force)$^2/Hz$. It is dimensioned to the number of independent inputs.

Furthermore,

$$a_i(\omega) = \sum_{j,k} H^T_{ji}(\omega) S_{jk}(\omega) H_{km}(\omega). \tag{8.3.6}$$

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. Y_i is a spatial loading term associated with the i^{th} row and column of S, and F_{ij} is a spectral only matrix function.

$$S_{ij}(\omega, x) = Y_i(x) Y_j(x) F_{ij}(\omega) \tag{8.3.7}$$

It typically has units of $1/Hz$.

The **RanLoads** section contains the following required keywords.

Parameter	Argument	Description
matrix	*Int/String*	matrix-function identifier
load	*Integer*	row/column identifier

The **matrix** keyword identifies the appropriate **matrix-function** (see Section 4.10.17). 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 8.3.7) 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 3×3, 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 section 4.5) 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 (or 454 kg) 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)*386.4)
END
```

Scaling the input force for a random vibration analysis can be confusing. [4] This is especially true since enforced acceleration cannot be used to apply the force . The example above uses United States customary units. The **wtmass** parameter has been applied. In SI units, `wtmass`= 1, and 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 (9.3), 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.

8.4. Initial Conditions

Initial conditions are specified via the **initial-conditions** section. The initial conditions are used as the initial state for transient analysis. Both linear and nonlinear transient are supported. The variables supported for initial conditions are given in Table 8.4.

[4]Note that we are scaling the spatial forces, Y_i, which are combined as a product in equation 8.3.7. Thus, the scale factor is linear in the load. The resulting input power spectrum, S_{ij}, will contain the square of the scale factor.

Keyword	Field Names	Description
displacement	dispX, dispY, dispZ, displacement_x, displacement_y, displacement_z	Translational displacement
velocity	velX, velY, velZ, velocity_x, velocity_y, velocity_z	Translational velocity
rotation	rotX, rotY, rotZ, rotational_displacement_x, rotational_displacement_y, rotational_displacement_z	Rotations for beams, shells, etc.
rotational_velocity	velRX, velRY, velRZ, rotational_velocity_x, rotational_velocity_y, rotational_velocity_z	Rotational velocity for beams, shells, etc.
acoustics	acoustics	Primary acoustic variable, usually pressure
acousticsdot	acousticsdot	Derivative of primary acoustic variable, usually particle velocity

Each term may be initialized in one of three ways. Initial conditions may be either

1. Read in from the **Exodus** file,
2. specified globally in the **initial-conditions** section,
3. or specified on a block-by-block basis in the input deck.

8.4.1. *Reading Initial State from a* Sierra/SM *Analysis*

To specify initial conditions to an SD analysis from an existing **Sierra/SM** analysis result, a special solution case, `receive_sierra_data`, is available. Consult section 5.23 for a detailed description of this option.

8.4.2. *Reading Initial Conditions from the Mesh File*

Initial conditions can be read from fields present on the mesh file. This method would usually be used when handing of initial conditions from the end state of a previous **Sierra/SD** or Sierra/SM run. The `time` command can be used to set which time step from the exodus file the initial conditions are read from. Valid options for time include reading from a specific time, start time of the transient simulation, or from the first or last step of the **Exodus** database. By default, time is set to the first step of the input **Exodus** database.

```
step = <int>time_step|last|first
OR
time = <real>time_val|last|first|transient_start_time
```

Note that as with **Exodus** in general, steps are one-based: `step = 2` refers to the second step on the mesh. In the case of `time = <real>`, the step chosen will actually be the nearest step with a time *greater than or equal to* the requested value.

An example of initial conditions input from **Exodus** file is given below. In this case, **Sierra/SD** will read both velocity and displacement initial conditions from the last time step of the **Exodus** file using the field names in Table 8.4.

```
INITIAL-CONDITIONS
  velocity=from_file
  rotational_velocity=from_file
  displacement=from_file
  rotation=from_file
  time = last
END
```

8.4.3. *Setting Initial Conditions in the Input Deck*

An example of setting initial conditions globally 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

BLOCK 2
  velocity = 0 1 0
END
```

In this case, the velocity is read in from the input deck on a block-by-block basis. 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, it is recommended to 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. The subsequent transient cases will get initial conditions from the previous case.

Care must be taken when setting initial conditions. If initial conditions are set from a **Sierra/SM** analysis, then that analysis must in general have small deformations. For example transferring an initial condition displacement from **Sierra/SM** with significant rigid body rotation would yield large stress in a linear **Sierra/SD** analysis due to the incompatible linear and nonlinear deformation state. Additionally, for elements like shells the nodal rotations must be compatible with the overall translational displacement.

8.5. Use cases for initial acceleration

All these cases can be applied as a spring mass system.

1. Apply force, displacement and velocity as an initial condition for transient dynamics. Compute,
$$A_n = (F_{applied} + Cv_n + Kd_n)/M$$
With a constant force, we should get a constant acceleration.

2. Apply a static load to the mass on the spring. Then apply (in a second case) the same load as a transient load. This should also result in a constant displacement.

3. TSR read. Read a static stress to the body. This is usually followed by a subsequent transient run.

   Ring down is expected. Internal stresses are not treated the same as applied forces. Applied external forces are not carried forward. Internal stresses are carried forward in the solution. When you start the transient run, you have an internal force only. Compute
$$A_n = (F_{internal})/M$$

4. `receive_sierra_data`. Read undeformed coordinates, X_o, and a nodal displacement, U_o. Compute *initial* coordinate,

$$X_1 = X_o + U_o.$$

   Afterwards, no load is required to maintain the X_1 configuration. Initial stresses may cause a deformation from that initial system. We may get velocity, force, and acceleration from file.

5. Prescribed acceleration. $A_o = a(t)$. This is applied only on a surface or node set. We convert this to a prescribed force, and treat it identically to an applied force.

6. Restart. A(0), v(0) and d(0) are prescribed at all locations. No solve needed.

7. There is a jump in force at some time other than t=0. Do nothing. We assume that the analyst wants it to ring.

8.5.1. Summary

- Use `velocity`, `displacement` from state change (whether read_sierra, user input, or multicase). Eigen may store displacement on the database. It is never used in a subsequent transient or statics analysis. It is always used in modal superposition cases. We get this by calling them PHI.
- Retain stresses across state change (multicase or read). Some cases retain stresses (statics, TSR, transient). Eigen may compute stresses, but does not store them on the database.
- Do NOT retain accelerations or forces across state change. We should use the forces calculated at that time (in the next state) for computing the balance.

This page intentionally left blank.

9. OUTPUT

This chapter identifies how to select different types of simulation data for output in different formats. The primary results are reported in the output **Exodus** file. This file can be large. The output syntax for over 50 different fields is described in section 9.7.

Two smaller and more specialized **Exodus** files can also be output, the History (see section 9.2) file and, for frequency response data, the Frequency (section 9.3) file.

Sierra/SD has the capability to output select variables in MATLAB format, as well, which is explained briefly in section 9.4.

Sierra/SD also supports the User Output capability of **Sierra/SM**, to reduce the need for external post-processing tools and allow finer-grained access to the user to interact with **Sierra/SD** behavior over the course of a simulation. Many flexible user output options are detailed further in section 9.5.

9.1. Exodus

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 5.1.1), the case identifier is used in place of `out`. More details are available in the `file` section (4.2).

Non-geometry-based finite element data, including system matrices and tables may be output in a format that is compatible with MATLAB. These text files have the extension `m`. The base file names describe the nature of the output. These files are generated in the current working directory.

Table 9-1. – Output section options

Option	Description	Section
acceleration	acceleration at nodes	9.7.3
acousticlighthill	acoustic Lighthill source term	N/A
aforce	acoustic forces	N/A
block_energies	block kinetic and strain energies	9.7.32
constraint_info	nodal constraint information	9.8.1
disp	displacements at nodes	9.7.1
EForce	element forces for beams	9.7.8
ElemEigChecks	first,seventh, and largest eigenvalues	9.8.2

ElemQualChecks	on || off || (*any boolean*), default is on	9.8.3
energy	element strain energy and strain energy density	9.7.30
eorient	element orientation vectors (geometry-defined coordinate system)	9.8.4
faa	force vector in the a-set	9.7.37
fatigue	Fatigue Damage Estimates	9.7.40
force	the applied force	9.8.5
GEnergies	global sum of energies	9.7.30.1
globals	ensure global output	9.7.31
Kaa	stiffness matrix in the a-set	9.7.36
line_weld	line weld elements	9.7.9
Maa	mass matrix in the a-set	9.7.35
material	material parameter element output	9.8.10
material_direction	user-defined element coordinate system	9.8.11
MLumped	nodal lumped mass in modal solution	9.7.41
MPhi	Mass × Displacement in modal solution	9.7.38
mesh_error	mesh discretization errors	9.7.33
MFile	MATLAB MFiles	9.7.34
nodal_charge	applied charge at nodes	N/A
pressure	pressure load vector	9.7.13
rainflow	Rainflow cycle counter output	9.7.39
reaction_force	the Dirichlet boundary reaction force	9.7.7
relative_disp	1D element nodal separation	9.7.10
RHS	RHS of system of equations to be solved	9.8.6
statistics	Mean and Std deviation of some variables	9.5.7
strain	element strain	9.7.19
stress	element stress	9.7.20
stress = gp	element stresses at Gauss points	9.7.19.1
principal_stresses	element principal stress vectors and magnitudes	9.7.20.3
signed_vonmises	element signed von Mises	9.7.20.5
temperature	element centroid temperature	9.7.29
velocity	velocity at nodes	9.7.2
voltage	voltage at nodes	9.7.29.1
vonMises	element von Mises stress	9.7.20.4
vrms	RMS quantities (random vibration only)	9.7.21

All the various options of the **outputs** section are shown in Table 9-1. The next sections describe each of the options and their results assuming an input file named `example.inp` and a geometry file named `example.exo`.

```
OUTPUTS
  Maa
  Kaa
  // displacement
END
```

9.1.1. Database Name

Option **database name** in the **outputs** section allows the user to specify an output file name instead of the default behavior as described above. As before, `-out` (or the case name for multi-case solutions) will be inserted before the file extension.

For example, the following input snippet would produce the output file name `new_example-out.exo` instead of the default `example-out.exo`.

```
FILE
  geometry_file = example.exo
END

OUTPUTS
  database name = new_example.exo // default is 'example.exo'
  // output keywords
END
```

which would result in output written to the **Exodus** database `new_example-out.exo`.

Although `database name` is supported, the solution case name must still be appended to the database name for compatibility with **Sierra/SD** multicase analyses.

9.1.2. Properties

Exodus Output Properties is currently BETA release.
Enable with the "`--beta`" command-line option.

Some options are available to control the contents of **Exodus** output files. These options can be used to shrink the output size or change the format of the output file. These properties are preceded by the **property** keyword in any input deck. Available properties are given in Table 9-2.

The below example parameters would provide minimal output file size for a run.

```
OUTPUT
  property integer_size_db = 4
  property real_size_db = 4
```

Table 9-2. – **Exodus** Property Output Options.

Property Type	Description
integer_size_db	Number of bytes for integers in **Exodus** output. Valid sizes are 4 or 8. By default, the size is the same as used in the input mesh file. 8 byte integers are required if node/element IDs greater than about two billion are used.
real_size_db	Number of bytes for real numbers in **Exodus** output. Valid sizes are 4 or 8. Default is 8, double precision. Use of 4 byte single precision output will roughly halve the output size but reduce output accuracy. Of particular note, Do not use four byte reals for restart.
compression_level	Compression level can be given values from zero to nine. Zero is no compression and nine is maximum compression. The default is generally zero compression. The compression level provides only a suggestion to the **Exodus** library, the actual amount of compression will vary. Potentially compression can reduce the size of the file by a factor of three or more. Compressed files may be incompatible with some post-processors.
maximum_name_length	Sets the maximum allowable name length for **Exodus** field names. Valid values are in the range $(32, 256]$, and the default is 64. Increasing this property can be useful for example if a user has very long user output names, which would otherwise be truncated. Alternatively, it can be lowered to shrink the file size if desired.

```
  property compression_level = 9
  property maximum_name_length = 32
END
```

9.2. History

There are two types of outputs available in the history file:

1. All results from the **outputs** section can be output to a *specified region of the model*, such as a block, side set, or node set. For a list of supported output names, refer to Table 9-1. See Section 9.2.1 for details.
2. Several special global history outputs that pertain to *locations in the model*, not node sets, side sets, or element blocks, may also be output to the history file. See Section 9.2.2 for details.

Note that the following **outputs** section names are ignored when found in the **history** section, since they involve auxiliary output files and formats, while history only supports **Exodus** format.

- MFile
- Kaa, Maa, Faa
- vrms

9.2.1. History Output for Node Sets, Side Sets, and Element Blocks

In addition to the **outputs** selection options, which control the variables that will be output to the history file, options are also available in the **history** section to control the model regions on which the variables are written to output. The user may specify node sets, side sets, a node list file (see Section 8.1), or element blocks. If side sets are selected, the output will include the nodes associated with the side sets, as well as the elements themselves. Selecting an element block automatically selects the associated nodes in that block. History also supports variables defined on virtual blocks, such as the element force in a virtual Joint2G element block. The default is *no* output selection; note that a warning will be issued when no model region is specified, and requested variables that pertain to a region of the model will *not* be output to the history file at all.

Multiple regions may be selected using a series of MATLAB concatenated ranges in the input syntax, similarly to subdomain selection in the echo block (Section 9.9.5), and the keyword *all* can be used to select all nodesets, sidesets, or blocks for output. For example,

```
HISTORY
  nodeset tail,1:10,17
  sideset  all
  nodeset '8,15' coordinate 4
  block '5,6, nose'
  stress
  disp
  nskip 10
  element centroid stress nearest
    location 0 1 2 as stress_gauge
  element strain nearest location 1 2 1 as strain_gauge
  element stress at element 1 as element_1_stress_gauge
  element strain nearest center of
    sideset wing as sideset_1_strain_gauge
  node acceleration at node 7 as node_7_accel_gauge
  node displacement nearest location 2 2 10 as disp_gauge
END
```

The corresponding nodeset, block, and sideset IDs/names should be specified as shown in the above example, with either a list of IDs/names, or the keyword “all”. The rules for specifying these lists are the same as for integer lists, and are detailed in Section 4.1.

Unlike subdomains, node set and side set IDs need not be contiguous in the **Exodus** file. If the selection criterion is a range, it may identify nonexistent sets. These will be silently ignored. In example 9.2.1 above in which nodesets `tail`, `1:10` and `17` are selected, the **Exodus** mesh must have nodeset number 10. Nodeset, sideset, and block names/IDs in the history file must be consistent with the corresponding **Exodus** input file.

9.2.1.1. Coordinate transformations

Any number of nodeset, block, and/or sideset selections can be specified in the history section. Nodeset, sideset, and block specifications may be followed by an optional coordinate entry. If a **coordinate** is specified (see section 4.9), all nodal results for the nodes in the region are transformed to the specified coordinate system before output to the file. If a particular node is identified in more than one specification, a warning will be issued to the log `.rslt` file, indicating which coordinate system will be used. The **coordinate** keyword for history section output will only work with nodesets, sidesets, and blocks; it is not supported for node list files.

The coordinate *name* 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; if the coordinate name is not convertible to an integer id (i.e., if it is a string), the *CID* output will be -1.

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. The history file will display variables as Cartesian regardless of coordinate choice. Table 4-21 shows the corresponding values for cylindrical and spherical coordinates.

9.2.2. *Global History Output Near a Location in the Model*

The history data of a mesh entity near a location can be written to the output file as a global variable using any of the following syntax:

```
Node|Element (optional, for element)centroid
   <string>variable_name
   Nearest Location <real>global_x <real>global_y <real>global_z
   as <string>history_variable_name
```

```
Node|Element <string>variable_name
   AT Node|Element <int>global_id
   AS <string>history_variable_name
```

```
Element <string>variable_name
   Nearest centroid of sideset <int(/string)>sideset_id(/sideset_name)
   AS <string>history_variable_name
```

Outputs at multiple locations can be requested in the history block. Each output location needs a unique `history_variable_name`. This capability currently only supports output of data at the closest *location* for three-dimensional elements or the closest *centroid* for both two- and three-dimensional elements (using the optional `centroid` flag). The closest location to the centroid of a specified sideset is calculated at the nearest point on a three-dimensional element *surface* that touches the given sideset. Data requested at a specified element is calculated at that element's centroid for both two- and three-dimensional elements; one-dimensional elements are not currently supported. If a one-dimensional element is specified directly, or if no elements in the model satisfy the search conditions, the `.rslt` file will issue the user a warning.

By default, **Sierra/SD** will look for the `variable_name` field(s) in the HISTORY region. For nodal fields **only**, fields from a different region may be used by adding `from OUTPUTS|FREQUENCY` after the variable name.

```
NODE <string>variable_name from OUTPUTS|FREQUENCY
   NEAREST LOCATION <real>global_x <real>global_y <real>global_z
   AS <string>history_variable_name
```

```
NODE <string>variable_name from OUTPUTS|FREQUENCY
   AT NODE <int>global_id
   AS <string>history_variable_name
```

Note that to match the **Exodus** convention, user-specified element, nodal, and sideset IDs should be one-based, i.e., starting from one, not zero.

Model regions, such as side sets, node sets, or element blocks, need not be specified to obtain global history outputs in the history **Exodus** file. *A warning that no region has been specified will still be output to the result log file*. In the context of global history outputs, this warning can be safely ignored.

Note that any output request may be used for element or nodal output, but only a subset have been tested. For element outputs, testing has focused on stress, strain, von Mises stress, and vrms outputs. Also, note that **only** stress/strain/VonMises are valid for *non-centroid* `nearest location` and `nearest centroid of sideset` element outputs. Tested nodal outputs include displacement, velocity, acceleration, and force. Additionally, any nodal field may be used in place of the nodal request.

A similar capability and syntax is used in **Sierra/SM**. A summary of the global history data output will be written to the log file in the following format:

```
======================= GLOBAL HISTORY OUTPUT ============================
 Prefix        Block   Entity, GId(: Type) Variable Near     At
------------   -----   ------------------- -------- -------  -------------
stress_gauge   1       element 1: Hex8b    stress   (0,1,2)  (0.5,0.5,0.5)
strain_gauge   1       element 5: Hex8b    strain   (1,2,1)  (1,2,1)
stress_gauge   1       element 1: Hex8b    stress   N/A      (2.5,1.5,0.5)
strain_gauge   1       element 6: Hex8b    strain   (2,3,1)  (2,2,1)
accel_gauge    N/A     node 7              accel    N/A      (1,1,0)
disp_gauge     N/A     node 38             disp     (2,2,10) (2,2,1)
==========================================================================
```

In the **Exodus** history file output, the global variable's prefix, or `history_variable_name` in the syntax given above, is prepended to the element variable's regular output name (Sections 9.7.20 and 9.7.19 give details on stress and strain variable names in **Sierra/SD**). For example, if two outputs are requested near a shell centroid and a solid with names "shell_strain" and "solid_stress," respectively, the corresponding global variables written in the history file would be

```
shell_strainSStrainX1, shell_strainSStrainY1, shell_strainSStrainXY1,
shell_strainSStrainX2, shell_strainSStrainY2, shell_strainSStrainXY2,
shell_strainSStrainX3, shell_strainSStrainY3, shell_strainSStrainXY3,
solid_stressVStressX,  solid_stressVStressY,  solid_stressVStressZ,
solid_stressVStressXY, solid_stressVStressXZ, solid_stressVStressYZ
```

Variables requested for global history output will be output on every other element block in the history block.

9.2.3. Output database options

In transient dynamics solutions 5.28, user control of output step interval `nskip` and output buffer Flush 4.6.1 operations are provided to increase efficiency of output. 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` parameters 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.

As with output in general (section 9.1.1), users can define a custom history output file name via the **database name** keyword. Additionally, **Exodus** properties can be defined as shown in Section 9.1.2.

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”.

9.3. Frequency

The frequency section provides information for data output from the Modal FRf, Direct FRF, shock, modalshock, and random vibration 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.

As with output in general (section 9.1.1), users can define a custom frequency output file name via the **database name** keyword. Additionally, **Exodus** properties can be defined as shown in Section 9.1.2.

The frequency values may be specified using the methods specified in Table 9-3. 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
```

Table 9-3. – 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 4.10.19)

```
  freq_min=10    // starting frequency in HZ
  freq_step=10   // frequency increment
  freq_max=2000  // stop freq. Outputting 201 freq points.
END
```

For the "log" method, $N_F + 1$ total frequencies are obtained by the following equation:

$$F_k = F_{min} \exp\left(\frac{k}{N_F} \log\left(\frac{F_{max}}{F_{min}}\right)\right) \quad \forall\, k \in [0, N_F]. \tag{9.3.1}$$

For the "linear" method, the following combinations of inputs are supported:

freq_min/freq_step/freq_max $N + 1$ frequencies: $[F_{min},\ F_{min} + F_{step},\ \cdots,\ F_{min} + N * F_{step}]$
where N is smallest integer such that $F_{min} + N * F_{step} \geq F_{max}$

freq_min/freq_step/NF $N_F + 1$ frequencies: $[F_{min},\ F_{min} + F_{step},\ \cdots,\ F_{min} + N_F * F_{step}]$

freq_min/NF/freq_max $N_F + 1$ frequencies: $[F_{min},\ F_{min} + \Delta f,\ \cdots,\ F_{min} + N_F * \Delta f]$
where $\Delta f = \frac{F_{max} - F_{min}}{N_F}$

Frequency 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 (9.2) has more information on specification of the output region.

The SIERRA translator **exo2mat** may be used to translate the output into MATLAB format for further manipulation and plotting.

9.4. MATLAB **output format**

In the following example, the mass and stiffness matrices are output in MATLAB format, but the displacement variables will not be output. As usual, output in MATLAB format is distinguished by file names ending with the extension `.m`. For example, on one processor, a mass matrix is written to the file `Maa.m`. Scripts are provided for assembling the subdomain matrices into the global matrix in MATLAB.

9.5. User Output

Sierra/SD enables several output processing operations to be computed during the run via the user output blocks. Use of these outputs can simplify post processing and reduce the need to output large quantities of data or time steps to generate quantities of interest.

The user output command syntax computes new output fields. These output fields can then be included in `outputs`, `history`, or `frequency` outputs.

9.5.1. Element Variable Spatial Statistics

```
USER OUTPUT
  block <list>(block names/ids)

  compute global <string>Name1 as average|avg
   ↪ of element stress|strain|vonMises (weighted by volume)

  compute global <string>Name2 as max|min|maxabs|minabs|sum
   ↪ of element stress|strain|vonMises
END
```

These `user output` commands calculate the spatial statistics of an element variable within a user specified list of blocks as a global variable. Multiple blocks can be selected in the same way as subdomain selection in the echo block (Section 9.9.5). The names such as `Name1` and `Name2` are given in the input deck. These names can then be included in `outputs`, `history`, or `frequency` sections to turn on that user output in the corresponding output file.

The optional 'weighted by volume' command performs a volumetric average over the elements. Without this command an equal weight per element average is used.

The max and min commands output the maximum or minimum value for each stress/strain component over all elements in the set. For example, given the values $\{1, -10, 5\}$, the max would be 5 and the min would be -10. In contrast, the maxabs and minabs commands output the value with the maximum or minimum **magnitude**. Thus, given the same values $\{1, -10, 5\}$, the maxabs would be -10 and the minabs would be 1.

9.5.1.1. Element Averaging Example

```
USER OUTPUT
  block = block_5
  compute global strain_gauge as average of element strain
   ↪ weighted by volume
END

HISTORY
  strain_gauge
END
```

If `block_5` contained solid elements then the outputs would be a set of global variables `stress_gaugeVStressX`, `stress_gaugeVStressXY`, `stress_gaugeVStressXZ`, ... Where each value is the volumetric average of the strain over all elements in `block_5`.

9.5.2. Nodal Variable Spatial Statistics

```
USER OUTPUT
  block <list>(block names/ids)
  surface <list>(sideset names/ids)
  nodeset <list>(nodeset names/ids)
  compute global <string>Name as max|min|maxabs|minabs|sum|avg|average
   ↪ of nodal <string>Variable (from OUTPUTS|HISTORY|FREQUENCY)
END
```

This `user output` section calculates the spatial statistics of a nodal variable or field within a user specified region for output file or history file as a global variable. Multiple regions are selected in the same way as subdomain selection in the echo block (Section 9.9.5).

Each user output can then be included by name in the `outputs`, `history`, or `frequency` section, as with element outputs (section 9.5.1).

The `Variable` used can typically be one of three things:

- One of the output requests typically available from `outputs`, `history`, or `frequency` files.
- One of the output fields written to the `outputs`, `history`, or `frequency` fields.
- A previously computed user output

By default, **Sierra/SD** will look for the `Variable` field(s) in the same region where the user-defined field is declared. A `from` keyword may be appended to the `expression variable` line to always read fields from a particular output region.

This is best shown via example.

```
USER OUTPUT
  nodeset = 6
  compute global avg_accelY as avg of nodal AccelY from OUTPUTS
END

OUTPUTS
  accel
END

HISTORY
  avg_accelY
END
```

Input 9.1. Nodal Statistics of a Field

In this case `AccelY` is one of the fields written by the `accel` request in the `OUTPUTS` region. The outputs written to the `HISTORY` region will be a single global variable `avg_accelY`, which is average Y acceleration of all nodes in nodeset 6.

```
USER OUTPUT
  nodeset = 5
  compute global d1 as max of nodal disp
END

OUTPUTS
  d1
END
```

Input 9.2. Nodal Statistics of an Output Request

In this case `disp` is one of the output requests that can be normally requested in `outputs`, `history`, or `frequency` sections. The outputs written to the **Exodus** results will be three global variables `d1DispX`, `d1DispY`, and `d1DispZ` as the maximum displacement values seen at a node in nodeset 5.

```
USER OUTPUT
  nodeset = 7
  compute nodal coord_r as function rCoord
  compute global coord_r_min as min of nodal coord_r
END

OUTPUTS
  coord_r_min
END
```

Input 9.3. Nodal Statistics Output of a Previously Computed Output

In this case `coord_r` is computed on each node of nodeset with ID 7 by a function `rCoord` defined in user output 9.5.6, then the global `coord_r_min` is computed as the minimum value of `coord_r` on any node. The output of this combination of operations is the single global variable `coord_r_min`.

9.5.3. *The Closest Distance Output*

```
USER OUTPUT

  skin blocks = <list>(block names/ids)
  block <list>("A" block names/ids)
  surface <list>("A" sideset names/ids)
  nodeset <list>("A" nodeset names/ids)

  compute global <string>name_1 as closest distance
   ↪ to block <list>("B" block names/ids)
   ↪ (search node_face|node_node|mixed)

  compute global <string>name_2 as closest distance
   ↪ to surface <list>("B" sideset names/ids)
   ↪ (search node_face|node_node|mixed)

  compute global <string>name_3 as closest distance
   ↪ to nodeset <list>("B" nodeset names/ids)

  compute nodal <string>name_4 as closest distance
   ↪  to surface <list>("B" sideset names/ids)

  compute nodal <string>name_5 as closest distance
   ↪  to block <list>(B block names/ids)
END
```

The closest distance output computes the closest distance between two different pieces of the mesh.

The **compute global** option requests, as a single global variable, the closest distance between the “A” set blocks, sidesets, nodesets and the “B” set blocks, sidesets, or nodesets. This is the minimum separation distance between these two sets anywhere in the model in the current displaced shape as computed by closest point projection.

When skin blocks is used, all skinned blocks will only use nodes on their exterior surface, omitting interior nodes in the context of that user output section. This can improve performance when there are many interior nodes, and can be less cumbersome than creating sidesets for all closest distance interfaces.

An optional search command applies to the block/sideset from the **compute global** line. For mixed the default is `node_face`. While the node-face search calculates the distance from the nodes in set “A” to the closest face in set “B”, the node-node search option computes the closest distance from the nodes in set “A” to the nodes in set “B”. If the “B” set contains only nodesets, then only the `node_node` search can be used, and a search option there will be ignored (with a warning).

The closest distance nodal variables is currently BETA release.
Enable with the “`--beta`” command-line option.

The **compute nodal** option computes the closest distance between each node in the “A” set to the faces in the “B” set. This output is a full field result for the closest distance calculation.

9.5.3.1. Modal Random Vibration Closest Distance Output

The closest **nodal** outputs have special meaning for the **modalranvib** solution case (section 5.16), where **Sierra/SD** does not typically calculate or output enough information to post-process the difference in displacement between two points. The names (and meanings) of the nodal closest distance outputs also change in **modalranvib** results.

The closest distance nodal outputs are available in PSD form if requested in the **frequency** section during **modalranvib** (table 9-4).

Real Names	Imaginary Names	Purpose
<name>_RelDispGxx	<name>_ImagRelDispGxx	Power spectral density of relative displacements
<name>_RelDispGxy	<name>_ImagRelDispGxy	
<name>_RelDispGxz	<name>_ImagRelDispGxz	
<name>_RelDispGyx	<name>_ImagRelDispGyx	
<name>_RelDispGyy	<name>_ImagRelDispGyy	
<name>_RelDispGyz	<name>_ImagRelDispGyz	
<name>_RelDispGzx	<name>_ImagRelDispGzx	
<name>_RelDispGzy	<name>_ImagRelDispGzy	
<name>_RelDispGzz	<name>_ImagRelDispGzz	
<name>		Distance to the closest point

Table 9-4. – ModalRanVib Frequency Closest Distance Nodal Output. <name> is the name of the user output request.

Additionally, the closest distance nodal outputs are available in RMS form if requested in the **outputs** section during **modalranvib** (table 9-5).

Real Names	Imaginary Names	Purpose
<name>_RelDispRMSxx <name>_RelDispRMSxy <name>_RelDispRMSxz <name>_RelDispRMSyx <name>_RelDispRMSyy <name>_RelDispRMSyz <name>_RelDispRMSzx <name>_RelDispRMSzy <name>_RelDispRMSzz	<name>_ImagRelDispRMSxx <name>_ImagRelDispRMSxy <name>_ImagRelDispRMSxz <name>_ImagRelDispRMSyx <name>_ImagRelDispRMSyy <name>_ImagRelDispRMSyz <name>_ImagRelDispRMSzx <name>_ImagRelDispRMSzy <name>_ImagRelDispRMSzz	RMS of relative displacement
<name>		Distance to the closest point

Table 9-5. – ModalRanVib **Exodus** Closest Distance Nodal Output. <name> is the name of the user output request.

9.5.3.2. Closest Distance Examples

```
USER OUTPUT
  block block_5
  compute global dist57 as closest distance to block block_7
  compute global dist59 as closest distance to block block_9
END

HISTORY
  dist57
  dist59
END
```

Input 9.4. Closest Global Distance

This will output a two global variables `dist57` and `dist59` which are the closest distance between blocks 5 and 7 and blocks 5 and 9 respectively in the current deformed configuration. This form of the command will work with the statics, transient, Modaltransient, and NlTransient solution cases.

```
USER OUTPUT
  block block_11
  compute nodal dist_to_surf_6 as closest distance to surface 6
END

HISTORY
  dist_to_surf_6
END
```

Input 9.5. Closest Nodal Distance

This will output nodal variable `dist_to_surf_6` which at each node of `block_11` is the distance from that node to the closest part of `block_6`. For the statics, transient, and modaltransient solution cases this will be a single value per node. For the modalranvib solution case this will be a set of variables describing the statistical properties of that distance calculation.

9.5.4. Temporal Variable Statistics

These `user output` variables support the same use case as the `Statistics` file, with more granular control over variable selection, and broader support for variables to be derived. These `user output` variables are intended as a replacement for the `Statistics` file. The rules governing output "frequency" are consistent with Outputs 9 , History 9.2 , and Frequency 9.3. In transient and modaltransient simulations the number of steps skipped between writing output is configured using the option `nskip`5.28.

Statistical `user output` values do not match values in the `Statistics` file. The cause of this is the inclusion of the model's initial state in `user output` calculations, and its exclusion from `Statistics` file calculations. The `OutputInitialTime` parameter has no effect on either calculation.

```
USER OUTPUT
  compute element <string>name1 as average|standard deviation|rms|min|max
   ↪  over time of element <string>var1

  compute nodal <string>name2 as average|standard deviation|rms|min|max
   ↪  over time of nodal <string>var2
END
```

This `user output` block computes two new derived variables `name1` and `name2` by applying an operator over time to the built-in variables `var1` and `var2`. These derived outputs require time histories, and are only tested for transient, modaltransient, and statics solution cases.

Temporal statistics calculations periodically output the specified statistical value of the chosen field each point in the model has experienced *up until that time, at each requested output step*. To avoid outputting intermediate steps and instead output only the variable statistic *over all time*, set `nskip` in the corresponding `history`, `frequency`, or `solution` section to the final time step. Note that other requested output quantities will also be affected by the `nskip` option. Sample input is included in the examples section below.

Supported operators include minimum(`min`), maximum(`max`), mean estimate(`average`), the root mean square (`rms`), and the `standard deviation`. Note that the `standard deviation` operator uses Bessel's correction in its calculations. That is, we define the variance as:

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2, \quad \text{not} \quad \frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2.$$

The source variables var1 and var2 must be built-in variable request strings; "chaining" derived variables together is not currently supported. Additionally, these variable request strings must match variable requests in other contexts; Stress and Displacement are valid, but StressXX and DispX are not.

Most built-in variable requests represent several variable fields in the results file. Therefore, derived variable fields are defined by prepending the user string name1 to the built-in field names StressXX, StressYY, StressZZ, . . . The result in this case would be name1StressXX, name1StressYY, name1StressZZ, . . .

Every built-in element request and most built-in nodal requests are available for use in derived outputs.

Some specialized nodal requests are not currently supported for derived outputs. For one example, variables derived from AcousticLighthill will always report 0. This is a known deficiency which can be addressed as needed.

9.5.4.1. Temporal Statistics Examples

The below input example writes three nodal variables RMSAccAccelX, RMSAccAccelY, RMSAccAccelZ which is the root-mean-squared acceleration components seen at each node over time.

```
USER OUTPUT
  compute nodal RMSAcc as rms over time of nodal accel
END

OUTPUT
  AvgAcc
END
```

The following input snippets demonstrate how to output *only* the maximum over all time of element von Mises stress at the final simulation time step.

```
SOLUTION
  Transient
    Time_step = 1e-4
    nsteps = 15030
END
```

```
HISTORY
  Nskip = 15030
  Block All
  Max_VM_Stress
END
```

```
USER OUTPUT
  Block all
  Compute element Max_VM_Stress as Max Over Time of element vonMises
```

```
END
```

9.5.5. *Rotating Output Variables*

```
USER OUTPUT
    transform nodal|element|global variable <string>Variable to|from
     ↪  coordinate system <coordinate>|from_geometry as <string>Name
  END
```

`User outputs` can also be used to rotate variables, including 2D and 3D stresses, into a given coordinate system. This is similar to the `Variable Transformation Command` in **Sierra/SM**. This is enabled for both vectors and tensors for element and nodal vars, and for global vectors. The `Variable` here is defined the same way as in section 9.5.2. `To` is used for transforming variables from global into a new coordinate system. `From` is used for transforming variables from a given coordinate system into the global coordinate system. See section 6.6.1.3 for info on the `from_geometry` option.

Below is an example where transform is used to rotate stresses and net forces:

```
USER OUTPUT
  transform element variable VStress to coordinate system swap_XZ as
   ↪  transformS
  transform nodal variable nodalStress to coordinate system swap_XZ as
   ↪  rotated_node_stress
  transform global variable NetForce to coordinate system swap_XZ as
   ↪  rotated_NetForce
END

OUTPUT
  stress
  stress=nodes
  force
  transformS
  rotated_node_stress
  rotated_NetForce
END
```

This will write rotated versions of all of the associated derived variable fields to `outputs`. For shells, stress is transformed on a layer by layer basis. The "*" is used followed by the layer number in this special case. An example of this is shown below, where the shell stress (typically in the element-local coordinate system) is transformed into the global coordinate system.

```
USER OUTPUT
  transform element variable SStress*1 from coordinate system
   ↪  from_geometry as global_sstress*1
  transform element variable SStress*2 from coordinate system
   ↪  from_geometry as global_sstress*2
  transform element variable SStress*3 from coordinate system
   ↪  from_geometry as global_sstress*3
END

OUTPUT
  stress
  global_sstress*1
  global_sstress*2
  global_sstress*3
END
```

9.5.6. *Analytic Function Output*

Nodal, element, and global variables can also be defined based on the output of an analytic function as follows:

Global Analytic Function Output is currently BETA release.
Enable with the "- -beta" command-line option.

```
USER OUTPUT
  surface <list>(surface names/ids)
  nodeset <list>(nodeset names/ids)
  block <list>(block names/ids)
  compute nodal <string>name as function <function>
  compute element <string>name as function <function>
  compute global <string>name as function <function>
END
```

where <function> refers to the function ID of a valid analytic function (section 4.10.9). The analytic function computes a single value at each node or element. If the surface, block, or nodeset option is combined with nodal function output, the function is *only* be evaluated at the nodes of the corresponding mesh region, with a value of zero elsewhere in the mesh.

9.5.6.1. Total Pressure Example

```
FUNCTION totalPressure
  type = analytic
  expression variable scatteringPressure = nodal apressure
  expression variable incidentPressure = nodal acousticIncident
  evaluate expression = "scatteringPressure + incidentPressure"
END
```

```
FUNCTION cavitationFlag
  type = analytic
  expression variable totalPressure = nodal totalPressure
  evaluate expression = "totalPressure < 0 ? 1 : 0"
END

USER OUTPUT
  compute nodal total_pressure as function totalPressure
  compute nodal cavFlag as function cavitationFlag
END

OUTPUT
  total_pressure
  cavFlag
END
```

For an acoustic scattering transient analysis this example computes a nodal field `total_pressure` which is the summation of two other fields computed by **Sierra/SD**, the scattering and acoustic pressure. Further if that total pressure is below zero it sets a cavitation flag to “1” on the node, if the total pressure is positive then the cavitation flag would be “0”.

9.5.6.2. Displacement Magnitude Example

```
FUNCTION mag
  type = analytic
  expression variable dX = nodal dispX
  expression variable dY = nodal dispY
  expression variable dZ = nodal dispZ
  evaluate expression = "sqrt(dX^2 + dY^2 + dZ^2)"
END

USER OUTPUT
  nodeset 3
  compute nodal dispMag as function mag
END

OUTPUT
  dispMag
END
```

This output will compute at each node of nodeset 3 the magnitude of displacement and output it as a nodal value.

9.5.7. Statistics

For transient dynamics solutions only, summary statistical information may gathered and output for the time history of variables listed in Table 9-6. We gather information about the mean, the min/max, and the standard deviation. Data is gathered at each time step, independent of the frequency of output (e.g. **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 a `.stat` extension to store that data.

Statistical data requires *two* keywords for output. Both “statistics” and the keyword associated with that output quantity must be selected. For example, to output statistics of the force, the following output section is required.

```
OUTPUTS
  statistics
  force
END
```

As with output in general (section 9.1.1), users can define a custom statistical output file name via the **statistics** block:

```
STATISTICS
  database name = <string>
END
```

Keyword	Section	Comment
Displacement	9.7.1	
Velocity	9.7.2	
Acceleration	9.7.3	
Force	9.8.5	applied force
RHS	9.8.6	Right-Hand Side vector at each load.
vonMises	9.7.20	von Mises stress, supported for `echo` output only.

Table 9-6. – Supported Statistical Data types for Transient Dynamics. Selection of these quantities along with “statistics” results in an addition **Exodus** file containing mean, min/max and standard deviation data.

9.6. Output of Internal Variables

Output of Internal Variables is currently BETA release.
Enable with the “`--beta`” command-line option.

A limited capability exists to write internal nodal variables to the **Exodus** output file.

The syntax is given below:

```
OUTPUTS
  node|nodal variables = <string>internal_name (as <string>output_name)
  ...
END
```

where `internal_name` is the name of the nodal field used internally (e.g., **dispX** for the X-component of displacement), and `output_name` is an (optional) user-defined alias to use when writing to file.

Alternatively, vector-valued variables may be referred to by the root name and a component. For example, using `internal_name := disp(X)` would select the X-component of displacement.

Vector-valued variables like displacement can also use a shorthand notation for defining all components at once. This is accomplished simply by replacing `internal_name` with the root name. For example, `nodal variables = disp as d` would output *all* displacement fields (stored internally as **dispX, dispY, . . .**) as **dX, dY, . . .** .

As this is a beta capability it has some usability limitations. In most cases to properly output an internal variable with a different name, it must also be output with the standard. For example to output displacement in a Sierra/SM compatible field use the following

```
OUTPUTS
  disp
  nodal variables = disp as displacement_
END
```

This will output displacement information in SD default named fields dispX, dispY, and dispZ as well as the same information to the fields 'displacement_x', 'displacement_y', 'displacement_z'. Outputting to both locations is currently required to get the correct information populated.

9.7. Output of Simulation Results

Simulation results can determine over 50 different fields. The output syntax for each of the quantities is described in this section.

9.7.1. Displacement

Option `disp` 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 (**ceigen** Section 5.19.3, 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 imaginary component in the X direction is "`imagDispX`".

9.7.2. Velocity

Option `velocity` will output the velocities at the nodes to the output **Exodus** file.

9.7.3. Acceleration

Option `acceleration` will output the accelerations at the nodes to the output **Exodus** file.

9.7.4. Rotational_displacement

Option `rotational_displacement` will output computed root mean squared (RMS) quantities for rotational displacement in a random vibration analysis to the output file. In the **frequency** section, it will output the corresponding Power Spectral Densities to the frequency file. (See section 5.16).

9.7.5. Rotational_acceleration

Option `rotational_acceleration` will output computed root mean squared (RMS) quantities for rotational acceleration in a random vibration analysis to the output file. In the **frequency** section, it will output the corresponding Power Spectral Densities to the frequency file. (See section 5.16).

9.7.6. Constraint force

Option `constraint_force` will output the forces required to tie blocks together due to multipoint constraints, RBE3s, Rbars, or rigid elements, including tied surfaces (see section 10.1) and contact definition (section 10.2). Additionally, it will output the net constraint force on the active degrees of freedom, and the net moment about the origin of constraint forces on the active degrees of freedom. This output is currently only active for the static and transient solution cases and for the GDSW solver.

9.7.7. Reaction Force

Option `reaction_force` will output the Dirichlet boundary reaction force vector to the output **Exodus** file. Additionally, it will output the net reaction force on the active degrees of freedom, and the net moment about the origin of reaction forces on the active degrees of freedom. This output is currently only active for the static and transient solution cases and for the GDSW solver.

9.7.8. EForce

Option **eforce** will output the element forces for line elements (such as beams and springs) to the output **Exodus** file. Each two node, one-dimensional element will have 6 force entries for each node, for 12 element forces per element, and an additional 3 variables describing the difference in displacement across the element. Eforce results are output to the element local coordinate system which is defined by the user in the block section using the coordinate parameter. If no coordinate parameter is input, the global coordinate system will be used by default.

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 7.23). The output is the direct output of our internal force routine (which is a nonlinear routine). It can be confusing to output these nonlinear forces in a linear analysis. [1]

Eforce variable names are different for each solution case. When requested in typical analyses, such as **statics**, **transient**, or **eigen**, **eforce** produces the variables in Table 9-7. For **FRF** solutions, "Imag" is prepended to the existing names 9-8. For **modalranvib**, only translational forces are output, and are different between **outputs** and **frequency** files. In an **outputs** result file,

[1]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 use the global coordinate frame by default. Forces for zero length elements are linear in the displacement.

you will see the RMS of the force in the translation directions 9-9. In `frequency`, the Spectral Densities of the element force are represented by a Hermitian tensor at each frequency 9-10.

Variable Names
eforce1_x
eforce1_y
eforce1_z
emoment1_x
emoment1_y
emoment1_z
eforce2_x
eforce2_y
eforce2_z
emoment2_x
emoment2_y
emoment2_z
e_dx
e_dy
e_dz

Table 9-7. – Typical Output.

Real Names	Imaginary Names
eforce1_x	Imageforce1_x
eforce1_y	Imageforce1_y
eforce1_z	Imageforce1_z
emoment1_x	Imagemoment1_x
emoment1_y	Imagemoment1_y
emoment1_z	Imagemoment1_z
eforce2_x	Imageforce2_x
eforce2_y	Imageforce2_y
eforce2_z	Imageforce2_z
emoment2_x	Imagemoment2_x
emoment2_y	Imagemoment2_y
emoment2_z	Imagemoment2_z
e_dx	Image_dx
e_dy	Image_dy
e_dz	Image_dz

Table 9-8. – FRF Output.

Real Names	Imaginary Names
eforceRMS_x	ImageforceRMS_x
eforceRMS_y	ImageforceRMS_y
eforceRMS_z	ImageforceRMS_z

Table 9-9. – ModalRanVib Exodus Output.

Real Names	Imaginary Names
eforce_xx	Imageforce_xx
eforce_yy	Imageforce_yy
eforce_zz	Imageforce_zz
eforce_xy	Imageforce_xy
eforce_yz	Imageforce_yz
eforce_xz	Imageforce_xz

Table 9-10. – ModalRanVib Frequency Output.

9.7.9. *Line_Weld*

Option `Line_Weld` outputs line-weld-specific outputs as described in Section 7.24). The outputs include whether a given beam is in an active weld, and the force per unit length produced by the weld

The force returned is in the element (not global) coordinate frame.

9.7.10. *Relative_Disp*

Option `relative_disp` in the **outputs**, `history` , and/or **frequency** section(s) will output the relative displacement between the two nodes of 1-D elements in the model. That is, it will output the difference in displacements across the 1-D element. Currently, **relative_disp** is only supported for the Joint2G element type, and is not supported for **FRF** solutions. Relative rotations are not available. For solutions other than **modalranvib**, including **FRF**, this data is also accessible through **eforce**.

relative_disp is output in the element-local coordinate system if a coordinate system has been defined for the Joint2G block, and in the global coordinate system otherwise. This is consistent with **eforce** outputs, but differs from **displacement** outputs. Using the `coordinate` keyword in the `outputs` , `history` , or **frequency** will not affect this output.

relative_disp is particularly useful for the **modalranvib** solution case, where **Sierra/SD** does not typically calculate or output enough information to post-process the difference in displacement between two points. The names of **relative_disp** outputs also change in **modalranvib** results. See tables 9-11, 9-12, and 9-13.

Variable Names
RelDispX
RelDispY
RelDispZ

Table 9-11. – Typical Output.

Real Names	Imaginary Names
RelDispGxx	ImagRelDispGxx
RelDispGyy	ImagRelDispGyy
RelDispGzz	ImagRelDispGzz
RelDispGxy	ImagRelDispGxy
RelDispGyz	ImagRelDispGyz
RelDispGxz	ImagRelDispGxz

Table 9-12. – ModalRanVib Frequency Output.

relative_disp outputs are available in PSD form if requested in the **frequency** section during **modalranvib** (Table 9-12). Note that the diagonal **relative_disp** terms are output for both real and imaginary components. Hermitian symmetry is assumed in the calculation of **relative_disp** outputs, and incorrect results will be reported if this assumption is broken. Neither the input load, nor the **relative_disp** output should have non-zero imaginary terms along the diagonal. That is, **ImagRelDispGxx**, **ImagRelDispGyy**, and **ImagRelDispGzz** should all be zero. The warning message "Correlation Matrix is negative" indicates you may have this problem.

Real Names	Imaginary Names
RelDispRMSX	ImagRelDispRMSX
RelDispRMSY	ImagRelDispRMSY
RelDispRMSZ	ImagRelDispRMSZ

Table 9-13. – ModalRanVib **Exodus** Output.

relative_disp outputs are available in RMS form if requested in the **outputs** section during **modalranvib** (Table 9-13). Only diagonal terms are reported. The imaginary diagonal terms are set to 0.

9.7.11. 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 eigenvalues, 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. 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 defined in subsection Linear transient analysis section Solution Procedures of the Theory Manual. A displacement-based Newmark-Beta integrator has dynamic stiffness,

$$K + \frac{2}{\Delta t}C + \frac{4}{\Delta t}M.$$

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.

9.7.12. TIndex

It is occasionally useful to examine the residual after each iteration or solve. In the cases of nonlinear transient or nonlinear statics analysis, there may be many solves per output. Because of limitations in the output database format, it is 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 9-14.

Keyword	**Application**
standard	use time step or mode number as index
iteration	use the iteration count as index

Table 9-14. – TIndex parameters.

```
OUTPUT
  disp
  residuals
  Tindex=iteration // output on each iteration
END
```

Input 9.6. TIndex example

TIndex makes sense only in solutions that require multiple iterations per solve, such as nonlinear solutions. In other solutions, it is ignored, and output is provided at the standard time step.

NOTE: TIndex is a debugging function. As such, we do minimal checks. In some solutions, it might be possible to output data using both steps.

9.7.13. Pressure

Option **pressure** in the **outputs** selects applied pressure output to the **Exodus** file as both a sideset variable and a new nodeset variable. 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.

9.7.14. NPressure

Option **npressure** in the **outputs** 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).

9.7.15. APressure

Option **APressure** in the **outputs** 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. At the wetted interface, the nodes of node-face interactions inherit the degrees of freedom from the face, this will result in four degrees of freedom per node in the output **Exodus** file.

9.7.16. acousticIncident

Option **acousticIncident** outputs the incident pressure from scattering loads. This pressure is for visualization purposes only.

9.7.17. acousticHydrostatic

Option **acousticHydrostatic** outputs the hydrostatic pressure defined for acoustic materials. This hydrostatic pressure is defined the commands **hydrostatic_gravity** and **free_surface_point** in the block input and primarily affects cavitation computations.

9.7.18. APartVel

Option **APartVel** in the **outputs** will output the acoustic particle velocity to the output **Exodus** file as an element variable. This is 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.

9.7.19. Strain

Option `strain` will output the strains for all the elements to the output **Exodus** file. The total strain ϵ, elastic strain ϵ^e, and thermal strain ϵ^θ are defined by:

$$\epsilon = \frac{1}{2}\left(\nabla u + \nabla u^T\right) \tag{9.7.1}$$

$$\epsilon^e = \epsilon - \epsilon^\theta \tag{9.7.2}$$

$$\epsilon^\theta_{ii} = \alpha \Delta T \tag{9.7.3}$$

They can be requested separately by using the options

- **strain**: ϵ
- **elastic_strain**: ϵ^e
- **thermal_strain**: ϵ^θ.

Strains will be output for shell elements. The output variable names start `SStrain`. Shell elements output engineering strain. E.g., shells output ϵ_{xx}, ϵ_{yy}, τ_{xy} where $\tau_{xy} = 2\epsilon_{xy}$. This is distinct from solid elements which output strain as ϵ_{xx}, ϵ_{yy}, ϵ_{zz}, ϵ_{xy}, ϵ_{yz}, ϵ_{xz}. This difference in strain definition means that for an equivalent deformation a shell element will report twice the numerical shear strain that a solid element will report even though the two shear states are identical.

SStrainX1, SStrainY1, SStrainXY1	- *top layer of the shell*
SStrainX2, SStrainY2, SStrainXY2	- *mid-plane of the shell*
SStrainX3, SStrainY3, SStrainXY3	- *bottom layer of the shell*

For elastic strain,

SElasticStrainX1, SElasticStrainY1, SElasticStrainXY1	- *top layer of the shell*
SElasticStrainX2, SElasticStrainY2, SElasticStrainXY2	- *mid-plane of the shell*
SElasticStrainX3, SElasticStrainY3, SElasticStrainXY3	- *bottom layer of the shell*

and for thermal strains we have

SThermalStrainX1, SThermalStrainY1, SThermalStrainXY1	- *top layer of the shell*
SThermalStrainX2, SThermalStrainY2, SThermalStrainXY2	- *mid-plane of the shell*
SThermalStrainX3, SThermalStrainY3, SThermalStrainXY3	- *bottom layer of the shell*

Strains are evaluated in the local element coordinate system. The local element coordinate system and also the ordering of the layers both depend on the ordering of the element's nodes.

The following strains will be output for volume elements:

```
VStrainX, VStrainY, VStrainZ,
VStrainYZ, VStrainXZ, VStrainXY
```

Likewise, for elastic and thermal strains, we have

```
VElasticStrainX, VElasticStrainY, VElasticStrainZ,
VElasticStrainYZ, VElasticStrainXZ, VElasticStrainXY
VThermalStrainX, VThermalStrainY, VThermalStrainZ,
VThermalStrainYZ, VThermalStrainXZ, VThermalStrainXY
```

Note that these strains are in the global coordinate system, not the local coordinate system.

For more information on stress/strain recovery, see Section 9.7.22.

9.7.19.1. Strain = GP

An output specification of `strain = GP` reports strain at the Gauss points of volumetric elements. For more information, see section 9.7.20.6.

9.7.20. Stress

Option `stress` will output the stresses for all supported elements to the output **Exodus** file.

This is the Cauchy stress $\sigma^C = C : \epsilon^e$; see equation 9.7.2.

9.7.20.1. Shell Stresses

The following stresses will be output for shell elements.

SStressX1, SStressXY1, SStressXZ1, SStressY1, SStressYZ1, SStressZ1, SvonMises1	- *top layer of the shell*
SStressX2, SStressXY2, SStressXZ2, SStressY2, SStressYZ2, SStressZ2, SvonMises2	- *mid-plane of the shel*
SStressX3, SStressXY3, SStressXZ3, SStressY3, SStressYZ3, SStressZ3, SvonMises3	- *bottom layer of the sh*

Note that 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 9.8.4). Also, the stresses are in the local (XY) element coordinate system defined by the ordering of the nodes.

9.7.20.2. 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 9.7.22.

9.7.20.3. Principal Stresses

Option `principal_stresses` will output the three principal stress vectors and magnitudes in order of descending signed value. `Principal_stresses` are only supported for volume elements.

The magnitudes are output as:

```
Max_Principal_Stress, Intermediate_Principal_Stress, Min_Principal_Stress
```

The eigenvectors are output as:

```
Max_Principal_Stress_x, Intermediate_Principal_Stress_x, Min_Principal_Stress_x,
Max_Principal_Stress_y, Intermediate_Principal_Stress_y, Min_Principal_Stress_y,
Max_Principal_Stress_z, Intermediate_Principal_Stress_z, Min_Principal_Stress_z
```

9.7.20.4. Von Mises stress

Option `vonMises` 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 as `VonMises`. Surface elements define stresses on the top, center and bottom layers. The von Mises stress is reported in the output **Exodus** file at each of the 3 layers (`SVonMises#`) as well as the maximum over all layers (`VonMises`). See section 9.7.25 for more information on shell stress recovery. For beam elements, the von Mises stress is reported at each stress recovery point (`VonMises_SRP#`), as well as a maximum over all stress recovery points (`VonMises`). See section 9.7.26 for more information on beam stress recovery.

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.

9.7.20.5. Signed von Mises Stress

Option `Signed_VonMises` will output the magnitude of von Mises stress, given the sign of the principal stress with the largest magnitude. `Signed_VonMises` is only supported for volume elements.

The output variable is:

```
Signed_VonMises
```

9.7.20.6. Stress = GP

An output specification of `Stress = GP` reports stress at the Gauss points of volumetric elements. It is currently only available for Hex20, Tet10, Wedge15, and Pyramid13 elements. Note that for a Hex20 there are 27 Gauss points with 6 stresses, for 162 outputs per element.

Stress components are labeled as `SIGMA_XX_etype_GP%d`, where `%d` is replaced with the ijk numbering scheme detailed below, and `_etype` is replaced with the element type (e.g. `_hex20` or `_tet10`).

The Gauss point ordering assigns each component of a given gauss point a value based on its relative position vs. the other gauss points in that component's direction.

For example, a hex with 2 points in each direction (-A, +A) would have the following relationship between labels and parametric coords:

```
(-A, -A, -A) -> "000"
(+A, -A, -A) -> "100"
(+A, +A, -A) -> "110"
(-A, +A, -A) -> "010"
(-A, -A, +A) -> "001"
(+A, -A, +A) -> "101"
(+A, +A, +A) -> "111"
(-A, +A, +A) -> "011"
```

Likewise, a tet with 2 unique gauss point values a & b, with b>a would have the following labels:

```
(b, a, a) -> "100"
(a, b, a) -> "010"
(a, a, b) -> "001"
(a, a, a) -> "000"
```

The gauss point labels for a Hex20 element is shown below.

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 9-15. – Hex20 Gauss Point Locations. The constant A=0.77459666924148. The unit element is 2x2x2, with a volume of 8 cubic units.

9.7.20.7. Stress = Nodes

Stress = Nodes is currently BETA release.
Enable with the "- -beta" command-line option. An output specification of `Stress = Nodes` computes a set of nodal stress fields `nodalStressX`, `nodalStressY`, `nodalStressZ`, `nodalStressXY`, `nodalStressYZ`, `nodalStressXZ`. These nodal fields are computed using an L_2 projection of the element Gauss point field data, allowing one to visualize field variation within an element. Additionally, von Mises (section 9.7.20.4) and principal stresses (section 9.7.20.3) will be output at each node.

The nodal stress fields are currently available only for Hex8, Hex20, Wedge15, Tet10, CuTet10, and Pyramid13 element types. Tet10 and CuTet10 elements project stress to the element vertices using the shape functions of a lower-order Tet4 element formulation. The edge nodes are subsequently assigned averaged values from connected vertices to avoid intrinsic interface discontinuities. The nodal stress fields will be zero on any element types that are not currently supported.

The L_2 projection requires solving a linear system of equations for each output step. The cost of formulating and solving this system of equations may impact run time. There are currently no user-configurable solver options to control the projection linear solver.

Diagnostic information on the projection linear solve statistics (number of iterations, final solve residual) may be requested using the `debug` option in the `ECHO` section.

9.7.21. 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. With the SVD option, the D matrix terms[52] which contribute to the von Mises stress are also output (see Section 5.16).

9.7.22. Stresses and Strains

Stress and strain values at element centroids are available. Solid element stress is evaluated in the global or basic frame. However, shell element stress and strain are evaluated in element space.

The total strain (9.7.1), elastic strain (9.7.2), and thermal strain (9.7.3) can all be requested separately; see Section 9.7.19. The stress reported is always the Cauchy stress $\sigma^c = C : \epsilon^e$; see Section 9.7.20.

9.7.23. *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 9.7.20), then stress variables are defined for solids and shells. [1] Space is allocated in the output **Exodus** file, and data is written only if it is applicable. Table 9-16 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.

Table 9-16. – 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(σ_{vm})	
Signed_VonMises	$\pm\sigma_{vm}$		
Max_Principal_Stress	$\|\sigma_1\|$		
Intermediate_Principal_Stress	$\|\sigma_2\|$		
Min_Principal_Stress	$\|\sigma_3\|$		
Max_Principal_Stress_x	σ_{1x}		
Max_Principal_Stress_y	σ_{1y}		
Max_Principal_Stress_z	σ_{1z}		
Intermediate_Principal_Stress_x	σ_{2x}		
Intermediate_Principal_Stress_y	σ_{2y}		

[1]The variables are defined for solids and shells even if only one or the other occurs in the model

Intermediate_Principal_Stress_z	σ_{2z}		
Min_Principal_Stress_x	σ_{3x}		
Min_Principal_Stress_y	σ_{3y}		
Min_Principal_Stress_z	σ_{3z}		
ElemForce			forces

9.7.24. Solid Elements

If stresses are requested, solid elements will output the values of stress at the element centroid. The values reported are the engineering stresses in the global coordinate system.

If principal stresses are requested, solid elements will output the three principal stress vectors and magnitudes in order of descending signed value. The vectors are of unit length, defined in the global coordinate system.

9.7.25. Shell Elements

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, mid-plane and bottom surfaces. Second, 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 in-plane coordinate frames. As shown in Figure 9-1, 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. Element orientation vectors may be output with the **eorient** keyword described in section 9.8.4. von Mises stress, an invariant, is independent of the element orientation.

Stress recovery for the **TriaShell** is interesting. A **TriaShell** is a shell element created by combining Allman's triangle[2] with a DKT element.[8] Its stress vector $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_{xy})^T$ is the sum of stresses $\vec{\sigma}_{at}$ for Allman's triangle and $\vec{\sigma}_{dkt}$ for the DKT element,

$$\vec{\sigma} = \vec{\sigma}_{at} + \vec{\sigma}_{dkt}. \tag{9.7.4}$$

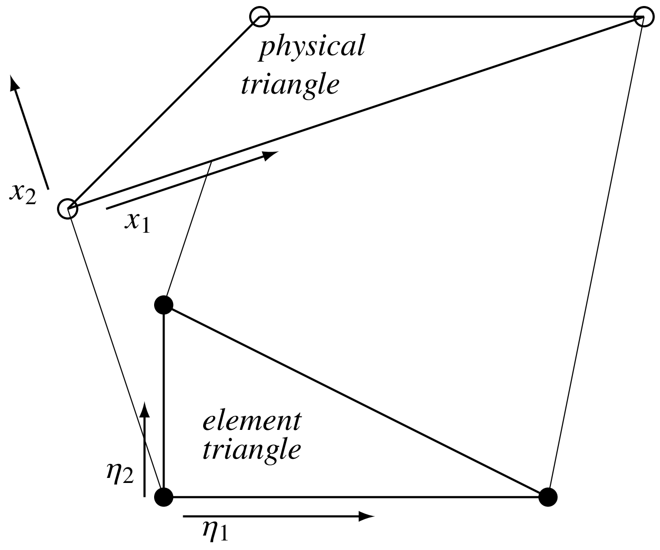
Allman's triangle represents the membrane dof, i.e., (u, v, θ_z). If the element lies in the x-y plane, then β_x and β_y are rotations of the normal to the undeformed middle surface in the x-z and y-z planes, respectively.

$$\{\kappa\} = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} \tag{9.7.5}$$

$\{\epsilon\}$ is the strain vector, and $[D]$ is the elasticity matrix for Allman's triangle. The stresses through the three surfaces of the shell element are the same. Therefore,

$$\vec{\sigma}_{at} = [D]\{\epsilon\}. \tag{9.7.6}$$

Figure 9-1. – 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.



For the DKT element, z is the coordinate direction normal to the element, with $z = 0$ representing the mid-plane. $[D]$ is the elasticity matrix.

$$\vec{\sigma}_{dkt} = z[D]\{\kappa\} \tag{9.7.7}$$

$\vec{\sigma}_{dkt}$ 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 9-1.

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 7-1, 7-2 and 7-3 describe these composite elements.

Stress output for HexShells (section 7.10) is calculated for a standard Hex8 element, even when multiple layers are defined.

9.7.26. Beam Elements

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. 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 most line type elements. However, some recovery may be obtained using the element force output (see Section 9.7.8).

For Beam2, Nbeam, and TiBeam elements (sections 7.11 to 7.13), axial stress will be reported if **stress** is requested in the `outputs` section. Additionally, stress recovery points may be requested using the **stress recovery point** keyword. For each stress recovery point, the bending and 2 shear stresses will be reported (see input 9.7). Each stress recovery point represents a point on the beam cross-section and is defined by a pair of offsets in physical coordinates from the neutral axis of the beam. The von Mises stress at each stress recovery point is reported, as well as the maximum von Mises stress over all the stress recovery points. If no stress recovery points are requested, von Mises is taken to be absolute value of the axial stress.

```
BLOCK beam_block
  material=1
  beam2
    Area=8
    I1=2
    I2=10
    J=7
    stress recovery point  0.5  0.5
                          -0.5 -0.5
END
```

Input 9.7. Beam Stress Recovery Points Example

9.7.27. Surface Projection of Element Variables

Element output such as stress, strain or temperature is evaluated at the element centroid. Hexahedra, tetrahedra and prisms support the projection of some element quantities to element faces. We do this by sampling the gradients of the element shape functions at the local coordinates associated with that face's centroid.

Notes:

1. Surface Projection is triggered by the `output_sideset_data parameters` flag.
2. Surface Projection results are stored as sideset variables on the **Exodus** mesh.
3. Only some outputs have been enabled for surface projection, most notably the strain tensor, stress tensor, and von Mises stress.
4. Due to the nature of bubble shape functions when sampled away from the element centroid, the default Hex8 element (Hex8b) will report skewed surface projection results, and is not expected to give the same benefit as other volumetric elements, given the same displacement field.

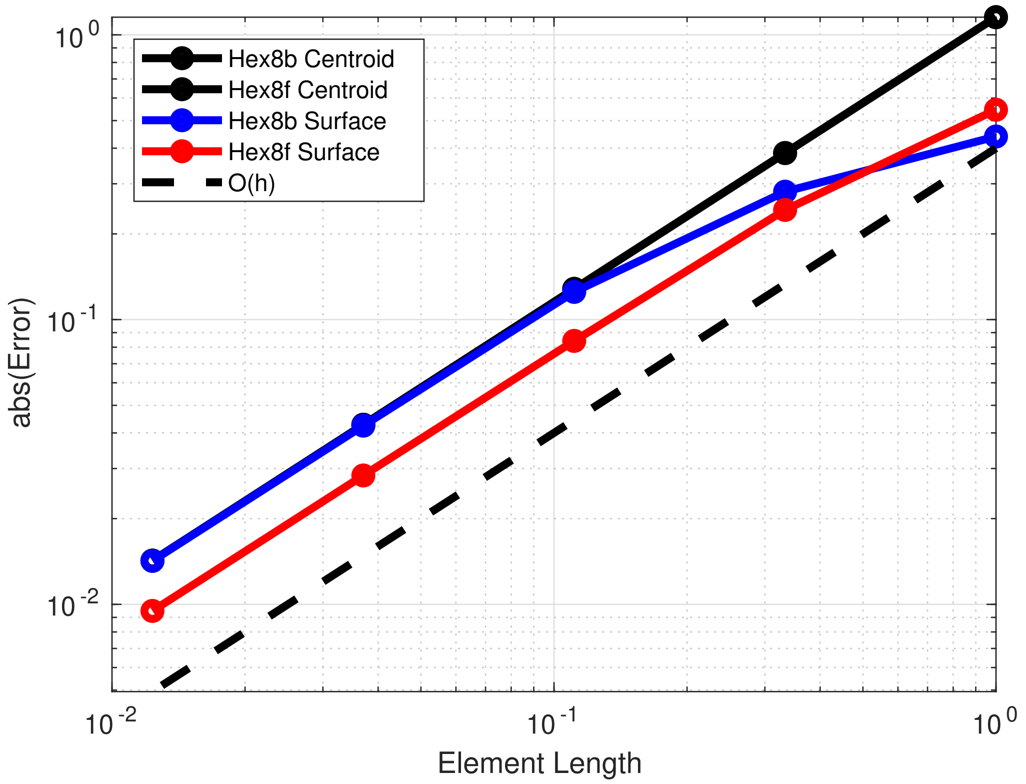


Figure 9-2. – Convergence of maximum stress at element centroids and surfaces.

5. Even when using the bubble hex element, surface projection is expected to give a more accurate representation of maximum stress, given that the maximum stress occurs on a sideset figure 9-2.

9.7.28. Ddamout

Table 9-17 lists the nodal and element variables that are output when the `ddamout` keyword is selected in the OUTPUTS or HISTORY sections. Element variables will be skipped when writing to the history file.

In Ddam analysis some of this data is also written to text files.

NRL sums of variables are calculated across modes with the equation:

$$R_{ia} + \sqrt{\sum_{b=1}^{N} R_{ib}^2 - R_{ia}^2}$$

Where:

$\boldsymbol{R}_{ia}$ is the maximum absolute value at location i for all modes.

$\boldsymbol{R}_{ib}$ is the value at location i and mode b.

Table 9-17. – Variables that are output from DDAM analysis.

Option	data type	Description
DDAM_MDISP	nodal	modal displacements
DDAM_MVEL	nodal	modal velocities
DDAM_MACC	nodal	modal accelerations
DDAM_MFOR	nodal	modal forces
DDAM_NRL_SDISP	nodal	nrl-summed displacements
DDAM_NRL_SVEL	nodal	nrl-summed velocities
DDAM_NRL_SACC	nodal	nrl-summed accelerations
DDAM_NRL_SFOR	nodal	nrl-summed forces
DDAM_VStress	element	modal volumetric stresses (tensor values)
		see sections 9.7.20, 9.7.22 and 9.7.24
DDAM_NRL_SUM_VStress*	element	nrl-summed volumetric stresses
DDAM_SStress*	element	modal surface stresses (tensor values)
		at each of 3 shell layers
		see sections 9.7.20, 9.7.22 and 9.7.25
DDAM_NRL_SUM_SStress*	element	nrl-summed surface stresses
DDAM_axialStress	element	modal axial stress (1D only)
		see sections 9.7.20, 9.7.22 and 9.7.26
DDAM_NRL_SUM_axialStress	element	nrl-summed axial stress (1D only)
DDAM_bendingStress*	element	modal bending stress (1D only)
		at each stress recovery point
		see sections 9.7.20, 9.7.22 and 9.7.26
DDAM_NRL_SUM_bendingStress*	element	nrl-summed bending stress
DDAM_shear Stress*	element	modal shear stress (1D only)
		2 shear directions at each stress recovery point
		see sections 9.7.20, 9.7.22 and 9.7.26
DDAM_NRL_SUM_shear Stress*	element	nrl-summed shear stress
DDAM_MVMSTR	element	modal von Mises stress
		see section 9.7.20.4
DDAM_NRL_SVMSTR	element	nrl-summed von Mises stress
DDAM_HYDROSTATIC	element	modal hydrostatic stress (3D only)
DDAM_NRL_SUM_HYDROSTATIC	element	nrl-summed hydrostatic stress
DDAM_MaxShear	element	modal max shear stress (3D only)
DDAM_NRL_SUM_DDAM_MaxShear	element	nrl-summed max shear stress

N is the number of modes.

Finally, we note a couple of additional details for output of `DDAM` data.

- In parallel runs, the text 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.

9.7.29. Temperature

The **Temperature** keyword is a single keyword that can be used to trigger **Exodus** output of element temperature at the centroid. Element temperature can either be read in as input temperature or calculated from energy deposition. The element centroid temperature can originate from the reference or block temperature, and the centroid, nodal or Gauss point **Exodus** data. If temperature is specified in multiple places, the order of precedence is block temperature, nodal temperature, centroid temperature, and finally Gauss point temperature. Note, nodal and Gauss point temperatures are later interpolated to element centroid temperatures that are used for calculation.

9.7.29.1. Voltage and Charge

The **voltage** keyword is a single keyword that can be used to trigger **Exodus** output of the nodal voltages for electro-mechanical coupled models. The **nodal_charge** keyword is a single keyword that can be used to trigger **Exodus** output of the applied nodal charges.

9.7.29.2. Volume

Keyword **Volume** selects **Exodus** output of element volume in the undeformed state. Volume is the integral of shape functions for solid elements. For shell elements the volume is the area times the thickness of the elements. For bar and beam elements the volume is length times the cross-sectional area of the element.

9.7.30. Energy

Option `energy` will place strain energies and strain energy density in the output **Exodus** file. Note that the current implementation of strain energies requires updating the element stiffness matrix, which can be expensive.

9.7.30.1. GEnergies

Option `GEnergies` in either the `echo` or `outputs` will trigger computation of global energy sums for the results or output **Exodus** file, respectively.

strain energy The strain energy is computed from $u^T Ku/2$ where u is the displacement and K is the current estimate of the tangent stiffness matrix.

kinetic energy Computed as $v^T Mv/2$. Here v is the velocity and M is the mass matrix.

work As a particle moves along $x(t)$ in the force field F, it does `work`

$$\begin{aligned} W(t) &= \int_{x(0)}^{x(t)} F(x)dx \\ &= \int_0^t F(\tau)\dot{x}(\tau)d\tau \end{aligned}$$

The simplest possible approximation is used,

$$W_n \sim \sum_{i=0}^{n} F_i \dot{x}_i \Delta t.$$

Integral approximation errors may introduce inconsistencies with the other energies. For the **outputs** case, the total energy is written out at each time step.

Strain energy calculation may not be complete for nonlinear solutions with linear viscoelastic materials.

When used in statics solutions, GEnergies only currently works with `echo`, not results output.

The strain energy and work are currently not computed correctly in models with non-zero displacement, velocity, or acceleration boundary conditions.

9.7.31. Globals

Option `Globals` in the **outputs** section will ensure that any global data is output, even if no other outputs are requested. Without this option (and with an empty output block), no output file would be written. If no global data is defined for a given solution type, or if any other outputs are requested, this option will have no effect. This option is also available for history, frequency, and statistics output, and behaves similarly.

> The `globals` option does not actually trigger any global outputs. Instead, it simply ensures that default global fields (e.g. `EigenFrequency`, `EigenVectScale`, and `ModeNumber` for eigen) will still be output in the absence of other output requests such as **displacement**.

9.7.32. Block_Energies

Option **block_energies** in the `echo` or **outputs** will trigger block-wise energy sums for the results or output **Exodus** file, respectively. The energy computations are done as described in 9.7.30.1, where the displacement and velocity vectors have been restricted to the element block. Kinetic and strain energies are computed. Global variables in the **Exodus** file are "KineticEnergy_" and "StrainEnergy_" with the block name appended.

9.7.33. 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. See [66] for a detailed description of the MeshErr calculation.

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

$$\text{ErrExplicitLambda} = \frac{\lambda_h - \lambda}{\lambda} \tag{9.7.8}$$

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).

The error (R) for a single element (K) is given as

$$R_K(u_h, \theta_h) = \nabla \cdot \sigma(u_h) + \theta_h \rho u_h + \sum_f \frac{1}{2} R_F \tag{9.7.9}$$

where θ is the eigenvalue, ρ is the density, and u is the displacement for mode h. R_F is the error on a single face

$$R_F(u_h) = J_F(N_F \cdot \sigma(u_h)) \tag{9.7.10}$$

where J_F is the jump of the stress across the element boundary in the direction of the element normal. Then, the global error estimate is written as:

$$MeshErr = \sqrt{\sum_e \frac{h_K^2}{p^2 d_{K,min}} ||R_K||^2 + \sum_f \frac{h_F}{p d_{F,min}} ||R_f||^2} \tag{9.7.11}$$

where h is the element length, p is the element order, and d is the maximum eigenvalue of the element or face.

9.7.34. MFile

Option **MFile** instructs **Sierra/SD** to output many MFiles including *Ksrr.m*, *Mssr.m* in the standard format 9.4. A partial index of the files written using this option is provided in Table 9-18. For a model with a large numbers of elements, the MATLAB files are also large in size. Binary MATLAB output is no longer supported.

MATLAB Output Example As an example, consider obtaining the Y component stiffness matrix diagonal entry of global node 77. The maps may be used as follows.

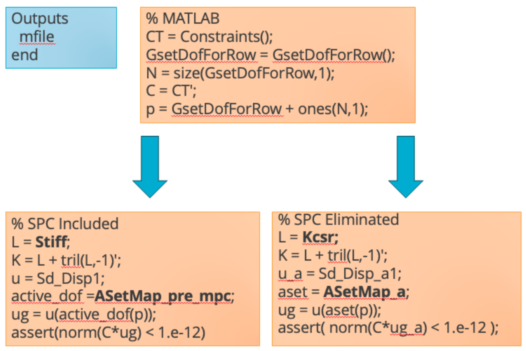
- Search through xxx_gid.m to find the global node number 77. Call index at which the node is found is $inode$.
- Calculate the GSet index, which in this case is $igset = (inode * 9) + 2$. The 9 is for 9-DOFs per node. The 2 is for Y being the second DOF of the nine (X, Y, Z, RotX, RotY, RotZ, Acoustic, Voltage, Temperature.)
- Lookup the Aset index, $iaset = ASetmap(igset)$
- If iaset is zero, then there is no defined stiffness matrix diagonal (could be an undefined DOF, or a fixed DOF.) Otherwise, the stiffness diagonal is $kaa(iaset, iaset)$.

```
OUTPUTS
mfile
constraint_info
END
```

Table 9-18. – Data Files Written Using the MFile Option.

Filename	**Description**
Stiff.m	Unreduced stiffness matrix including all active dofs
Kssr.m	Reduced stiffness matrix
Kgeomssr.m	Geometric 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
ASetmap_a.m	Map to convert from G-set to A-set The right-hand side is the equation number. The left-hand index is 9*(node index)+coordinate
Dampr.m	unreduced damping matrix (real components)
Dampi.m	unreduced damping matrix (imaginary components)
xxx_accelN.m	G-set acceleration output of step N
xxx_accel_aN.m	A-set acceleration output of step N
xxx_afN.m	G-set applied force output of step N
xxx_af_aN.m	A-set applied force output of step N
xxx_dispNN.m	G-set displacement output of step N
xxx_disp_aN.m	A-set displacement output of step N
xxx_presN.m	G-set nodal applied pressure of step N
xxx_pres_aN.m	A-set nodal applied pressure of step N
xxx_velocN.m	G-set velocity output of step N
xxx_veloc_aN.m	A-set velocity output of step N
modal_amp.m	modaltransient output of mode amplitude vs time

- Above the `xxx` refers to the input file name root.
- G-set output has dimension 9 (number of nodes).
- **Sierra/SD** adheres to the standard MATLAB conventions 9.4.
- 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.
- The 1 to N node ordering of the input **Exodus** file defines the **ASetMap**. Output file ordering may be different if there is a node order map.



```
Outputs
  mfile
end
```

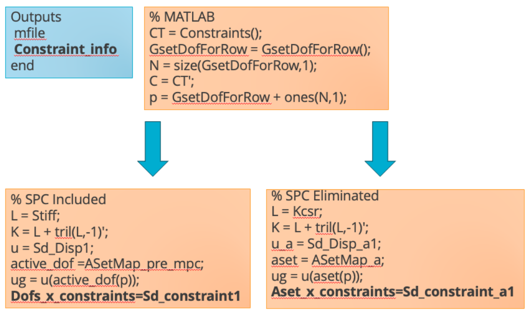
```
% MATLAB
CT = Constraints();
GsetDofForRow = GsetDofForRow();
N = size(GsetDofForRow,1);
C = CT';
p = GsetDofForRow + ones(N,1);
```

```
% SPC Included
L = Stiff;
K = L + tril(L,-1)';
u = Sd_Disp1;
active_dof =ASetMap_pre_mpc;
ug = u(active_dof(p));
assert(norm(C*ug) < 1.e-12)
```

```
% SPC Eliminated
L = Kcsr;
K = L + tril(L,-1)';
u_a = Sd_Disp_a1;
aset = ASetMap_a;
ug = u(aset(p));
assert( norm(C*ug_a) < 1.e-12 );
```

Figure 9-3. – In models with constraints, due to a contact or tied data input deck section for example, the MFile output includes the constraint matrix. This figure describes how to use it for a serial run

Input 9.8. MFile output of constraints matrices



```
Outputs
 mfile
 Constraint info
end
```

```
% MATLAB
CT = Constraints();
GsetDofForRow = GsetDofForRow();
N = size(GsetDofForRow,1);
C = CT';
p = GsetDofForRow + ones(N,1);
```

```
% SPC Included
L = Stiff;
K = L + tril(L,-1)';
u = Sd_Disp1;
active_dof =ASetMap_pre_mpc;
ug = u(active_dof(p));
Dofs_x_constraints=Sd_constraint1
```

```
% SPC Eliminated
L = Kcsr;
K = L + tril(L,-1)';
u_a = Sd_Disp_a1;
aset = ASetMap_a;
ug = u(aset(p));
Aset_x_constraints=Sd_constraint_a1
```

Figure 9-4. – In model with constraints, if both mfile and constraint info output are selected, then in addition, the constraints on the x degrees of freedom are also listed.

9.7.35. Maa

Option `Maa` selects the analysis-set mass matrix (if it exists) for output to the file `Maa.m` as usual 9.4.

9.7.36. Kaa

Option `Kaa` will output the analysis-set stiffness matrix to a file named `Kaa.m` as usual 9.4.

9.7.37. Faa

Option `faa` will output the analysis-set force vector (if it exists) to a file named `Faa.m`. following the standard convention 9.4.

9.7.38. MPhi

Option `MPhi` triggers computation and output of the mass matrix product $M\Phi$ to the **Exodus** file for the mode shapes, Φ, computed in the previous modal solution. `MPhi` is only implemented for the modal solution case. A consistent mass matrix is used. Mass matrix output is explained in Section 9.7.35. Like the mode shapes, the mass matrix is defined only on the analysis-set dofs. An **ASetMap** is also provided. The $M\Phi$ vanishes on fixed dofs. The names of the different dofs at a node are specified in Table 9.7.38.

MPhi Variable Names
MPhi_X (x translation)
MPhi_Y (y translation)
MPhi_Z (z translation)
MPhi_RX (x rotation)
MPhi_RY (y rotation)
MPhi_RZ (z rotation)
MPhi_A (acoustic)
MPhi_V (voltage)
MPhi_T (temperature)

9.7.39. Rainflow Cycle Counting

Option `rainflow` triggers the rainflow cycle counter to track stress cycles encountered by each element over time. `Rainflow` relies on `Signed_VonMises` to convert the stress tensor to a scalar signal, and is intended as a preprocessing step to time domain fatigue calculations. `Rainflow` is only supported for volume elements, and only in transient analyses.

The output variables are:

```
NumCycles, LastCycleAmplitude, LastCyclePeak
```

These outputs contain insufficient information to be useful on their own except in simple verification exercises. `Rainflow` is intended to be a silent dependency of `fatigue`, rather than a standalone output option.

9.7.40. Fatigue Damage

Option `fatigue` will output a damage estimate for each element using the stress history of that element. This process is supported for transient solutions and for modal random vibration, but in very different ways.

In a transient analysis, `fatigue` damage is calculated using stress cycles identified by the `rainflow` algorithm, and applying those cycles to the Walker damage function:

$$log_{10}(N) = A_1 + A_2 * log_{10}(S_{max} * (1 - R)^{A_3} - A_4)$$

Where S_{max} is the peak stress of the cycle, S_{min} is the minimum Stress of the cycle, and $R = S_{min}/S_{max}$. The number of cycles to failure N is then related to damage D by:

$$D = 1/N$$

A_1, A_2, A_3, and A_4 are material constants. Note that a cycle is ignored if $S_{max} <= 0$ or $S_{max} * (1 - R)^{A_3} <= A_4$, because purely compressive cycles are assumed to cause no damage, and because cycles below the endurance limit causes no damage.

The output variable is:

```
Damage
```

In modal random vibration, `fatigue` is its own solution case. See section 5.11 for more details. The output variables are:

```
NarrowBandDamageRate, WirschingDamageRate, ZeroCrossingRate,
PeakFrequency, Damage, Vrms
```

9.7.41. MLumped

Option `MLumped` will output the lumped mass matrix to the **Exodus** mesh as a nodal variable. `MLumped` is only implemented for the `Eigen` solution case. The lumped mass output is based on the analysis-set reduced mass matrix. Thus, mass on fixed degrees of freedom will be zero.

MLumped Variable Names
M_X (x translation)
M_Y (y translation)
M_Z (z translation)
M_RX (x rotation)
M_RY (y rotation)
M_RZ (z rotation)
M_A (acoustic)
M_V (voltage)
M_T (temperature)

9.8. Output of input for checks

Core input fields can be output for sanity checks: forces 9.8.5, materials 9.8.10, user-defined material direction 9.8.11, and geometry-defined element orientations 9.8.4. Diagnostics are provided for elements 9.8.2, 9.8.3, assembled matrix diagonals 9.8.8 9.8.7 linear system right-hand sides 9.8.6 and constraints 9.8.1. Finally Line Sample 9.8.9 returns input field values along a user specified line.

9.8.1. Constraint_Info

Linear system solvers are sensitive to redundant or inconsistent constraints. The option `constraint_info` selects nodal constraint information that is useful in preparing models with tied surfaces or other types of multipoint constraints. The Gap Removal solution case section 5.32 is too, and they are often used together.

Constraint information has several fields.

MPC_Status Indicates if a given node is involved in any MPC equation. A value of '1' indicates the node is used in at least one equation, '0' otherwise.

MPC_Touched Indicates how many times a given node shows in MPC equations.

Node_Face_MPC_Count For only node-face contact constraints, this indicates how many times that a node is used as the node of a node on face constraint. A value greater than one can be problematic as a given node can only be correctly tied to a single face without introducing over-constraint.

Node_Face_MPC_Redundancy For only node-face contact constraints this highlights nodes that may be over-constrained. Pay attention to values greater than one. The Node_Face_MPC_Redundancy is typically one less than Node_Face_MPC_Count unless there are more than 3 independent constraints for a specific sideset pair, as may occur if a sideset pair is used in both a tied constraint and a slip contact, or Tied Joint.

Node_Face_MPC_Gap indicates the distance the node of a node-face constraint must be moved to be placed on the face. Many problems with constraints stem from surfaces that do not properly match up geometrically.

Node_Face_MPC_Both_Node_and_Face For node-face constraints only this indicates if a particular node acts as a node in one node-face constraint and also is attached to the face of separate node-face constraint. A '1' means such situation occurs. In some cases such nodes may be part of a problematic cyclic constraint. However, in other cases this situation may be expected and cause no problems.

MPC_Origin Indicates what capability created the MPC. For example sliding contact vs. tied contact. The index number output matches a table in the rslt log file. Currently, this output is only available from node-face contact constraints.

9.8.2. *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 of the element stiffness matrix. 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 that this output is not supported for rigid elements (RBE2, RBE3, Rrod) and will be skipped on those blocks. Finally, if *ElemEig_1st* < *-1e-12 ElemEig_max*, a negative eigenvalue warning will be printed.

9.8.3. *ElemQualchecks*

Option `Elemqualchecks` takes a boolean, e.g., `on` (the default) or `off`. Unless this option is `off`, all the elements in the input file are checked for quality using various element quality metrics. If the option `on` is selected and the element's condition numbers falls outside the acceptable range, a warning message is printed. A summary is also printed, reporting the min/max quality of each block in the mesh.

The Tet4, Wedge6, Hex8, Tria3, and Quad4 elements implement a condition number from Verdict.[62] The acceptable limit for that warning may be modified by the *condition_limit* parameter, specified in the **parameters** section (4.4), and defaults to 10^6. The following table shows the acceptable ranges.

Element Type	Full Range	Recommended Range
Tet4	$1 - \infty$	1 – 3
Hex8	$1 - \infty$	1 – 8
Wedge6	$1 - \infty$	1 – 5
Tri3	$1 - \infty$	1 – 1.3
Quad4	$1 - \infty$	1 – 4

Additionally, several other element types inherit their condition number from the elements listed above. Those are listed in the following table.

element	uses condition number of	element
Tet10		Tet4
CuTet10		Tet4
Hex20		Hex8
Hexshell		Hex8
Wedge15		Wedge6
Tria3		Tri3
Tria6		Tri3
TriaShell		Tri3
NTria		Tri3
QuadT		Quad4
QuadM		Quad4
QuadTM		Quad4
Quad8T		Quad4
QuadS_GY		Quad4
NQuad		Quad4
KHQuad		Quad4

Table 9-19. – Elements using other elements condition number.

These approximations are optimistic as the condition number of the element is based on the overall element topology and rather than the specifics of the element formulation. For example the KHQuad, NQuad elements use a bilinear mapping while the QuadT uses two affine mappings (behaving as two triangles side by side.) In other cases elements such as NTria and Tria3 both use affine mapping but different details of element formulations may make them sensitive to different types of poor shapes. For higher order elements, such as Tet10 or Hex20, the element quality is typically determined just by the vertex nodes. Mid-edge nodes that conform to a curved geometry lead to much more accurate simulation than is possible with polyhedral elements, but, on the other hand, these elements are also more sensitive to poor element quality. The quality metrics discussed here ignore mid-edge nodes.

Some **Sierra/SD** elements that can invert do not implement a condition number. This is true for Pyramid5/13, InfiniteElement, PHex, PTet, PWedge and PmlIsoSolid elements. Condition numbers are also unimplemented for rigid elements, superelements, and all 1 dimensional and 0 dimensional elements.

9.8.3.1. Additional Volumetric Element Shape Metrics

In addition to condition number quality checks solid elements are checked for negative volumes. Negative volume 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.

These negative volumes are checked by evaluating the Jacobian at the element integration points. A negative Jacobian indicates the element is either fully inverted, or poorly shaped. The various solid element formulations have differing degrees of rigor in these checks. For example at Tet10 element evaluates these Jacobians based on only the vertex node positions, and, by using the absolute value of the Jacobian, can never detect an inverted element. For simulations in which inverted elements are undesirable, the CuTet10 is recommended.

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.

9.8.3.2. Additional Shell Shape Metrics

In addition to a shape based condition metric shell elements output a “Thickness Ratio” metric as the ratio of the thickness t to a length l,

$$t/l. \tag{9.8.1}$$

The length is defined as the minimum diagonal for quadrilaterals or the minimum edge length for triangular elements. The acceptable range may be modified by the *min_thickness_ratio* and *max_thickness_ratio* parameters, specified in the **parameters** section (4.4), and defaults to $10^{-5} - 10$. Shells with large thickness to length ratios can be ill-conditioned and have stability problems. Elements with very small length to thickness ratio can also be ill-conditioned due to a vanishing rotational stiffness term.

9.8.3.3. Additional Beam Shape Metrics

Beam elements implement a “Area Ratio” metric as the ratio of cross-sectional width $sqrt(a)$ to element length l.

$$\sqrt{a}/l \tag{9.8.2}$$

where a is the cross-sectional area, and l is the beam length. The acceptable range may be modified by the *min_area_ratio* and *max_area_ratio* parameters, specified in the **parameters** section (4.4), and defaults to $10^{-5} - 50$. Additionally, beam elements also implement a bending moment metric:

$$I_1/I_s \tag{9.8.3}$$

where I_1 is first bending moment, and I_s is the bending moment of a square cross-section ($a^2/12$). And likewise for I_2 (the second bending moment). The acceptable range may be modified by the *min_moment_ratio* and *max_moment_ratio* parameters, specified in the **parameters** section (4.4), and defaults to $10^{-4} - 10^4$.

Nbeam elements (section 7.12) also implement an offset length ratio metric:

$$|l - l_0|/l_0 \tag{9.8.4}$$

where l is the beam length with offsets, and l_0 is the length without. The acceptable maximum value may be modified by the *max_offset_ratio* parameter, specified in the **parameters** section (4.4), and defaults to 0.15. This is consistent with the GeomCheck condition for NASTRAN CBAR elements, which the Nbeam was developed from.

Beams with properties outside of the listed ranges will have very poor numerical conditioning and may cause issues for accurate and robust solution.

9.8.4. Eorient

Option `eorient` in the **outputs** 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.

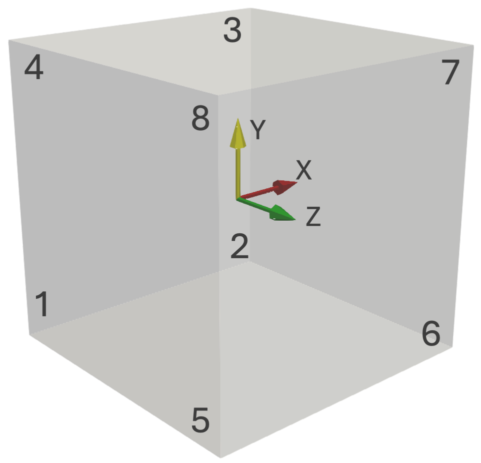
The orientation vectors are output as nine variables that collectively make up the three vectors required for element orientation. The output variables and the associated meanings for various elements are shown in tables 9-20 and 9-21 and figure 9-5.

Table 9-20. – Element Orientation Outputs.

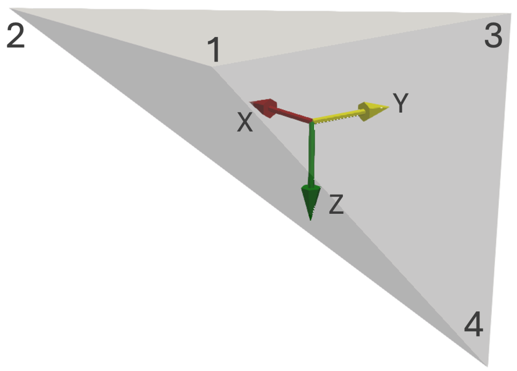
Name	Description
EOrient1_X	first orientation vector
EOrient1_Y	
EOrient1_Z	
EOrient2_X	second orientation vector
EOrient2_Y	
EOrient2_Z	
EOrient3_X	third orientation vector
EOrient3_Y	
EOrient3_Z	

Table 9-21. – Element Orientation Interpretation.

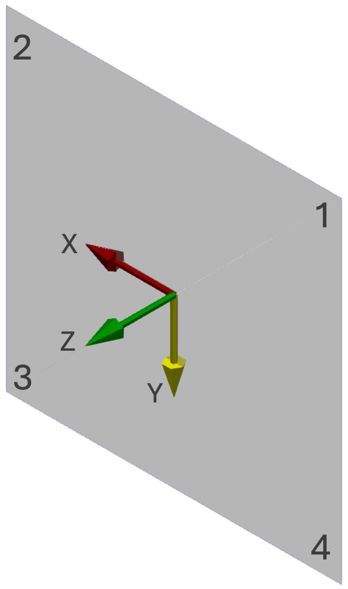
Element	EOrient1	EOrient2	EOrient3
Beam2	axial	first bending (I1)	2nd bending (I2)
Shells	Element X	Element Y	Normal
Solids	Element X	Element Y	Element Z
Hexshell	Element X	Element Y	thickness
ConMass	null	null	null



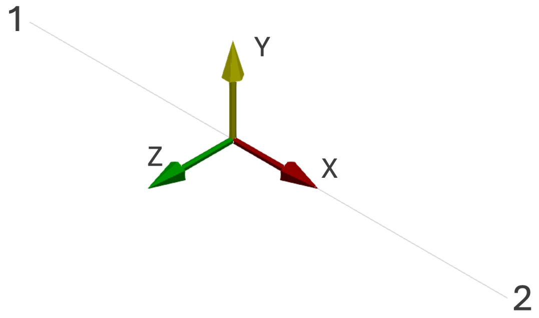
(a) Hex8



(b) Tet4



(c) QuadT



(d) Beam2

Figure 9-5. – Diagram of Element Orientation for Several Common Element Types.

9.8.5. Force

Option `force` will output the applied force vector to the output **Exodus** file, and the net force applied to active degrees of freedom, and the net moment about the origin of forces applied to active degrees of freedom. The net force is calculated from the right-hand side given to the solver, so implementations that modify the right-hand side (cavitation) may display non-physical net forces and moments. Net forces are available for static and direct transient solution cases.

If rigid body filtering is requested via the `FilterRbmLoad` option (section 8.3.21), the `intertialrelief` option in the outputs section will also output the filtered force to the output **Exodus** file, and the net force and moment about the origin. The naming convention for both nodal and net (global) values are **force_inertia_relief** for forces, and **moment_inertia_relief** for moments.

9.8.6. Right-hand side

Option `RHS` selects the right-hand side vector for the analysis type. It is used for verification and debugging purposes. For statics and dynamics, it represents the applied forces, pressures, inertial forces, or any pseudo forces introduced in preload (say by TSR).

9.8.7. KDiag

Option `kdiag` in the **outputs** 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 stiff regions of a model. All 7 terms may be seen by outputting KDiag in the `echo` section.

Figure 9-6 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. Beam stiffness is proportional to $1/L^3$. A common mistake is to generate stiff beams, which can ruin the numerical solution. See Section 9.8.8 for a related diagnostic on the dynamics matrix. [2]

[2] 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.

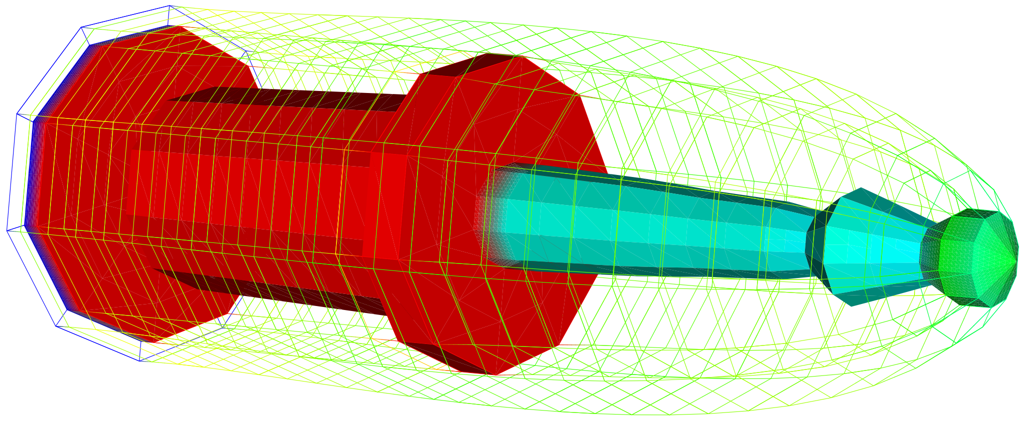


Figure 9-6. – Example *KDiag* output.

9.8.8. ADiag

Option `ADiag` in the **outputs** 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, (9.8.7), 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 9-22. 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 eigenvalue 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 9-22. – Selected Dynamic Matrix Definitions.

9.8.9. Line Sample

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 5.29 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[46]). Energy deposition is interchangeable with supplying an applied temperature. Line sample is also used for far-field processing in acoustics problems (see 8.1.9.1 or How To[44]), for example with infinite elements.[43]

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*
nskip	*integer*
database name	*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 supported: **Exodus** and MATLAB `MFile`. The default is `MFile`.

nskip Results output frequency; defaults to the value specified in the solution section; see Section 5.28.

database name The name of the output file; defaults to linedata.m for MATLAB output and linedata.exo for **Exodus** output.

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, voltage, and temperature. For transient data, the time values are also output for each array.

```
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
```

9.8.10. Material

The `material` keyword will output material properties for each element. Currently, this capability is only enabled for elasticity calculations, and is also not enabled for Lamé materials or isotropic_viscoelastic_complex materials.

9.8.11. Material Direction

Local coordinate systems may be defined to orient directional materials (sections 4.9, 6.6.1 and 6.6.1.3). The **material_direction** output provides the user-defined coordinate system ($\hat{r}$, $\hat{s}$, $\hat{t}$) vectors at each element. Visualizing these vectors can help inform if material coordinate systems have been setup as intended. Additionally, these coordinate vectors are the rows of an 3×3 rotation matrix for transforming quantities between the global (X, Y, Z) and local ($\hat{r}$, $\hat{s}$, $\hat{t}$) coordinate systems within the element.

Table 9-23. – Material_Direction Outputs.

Name	Description
material_direction_1_X	first coordinate vector
material_direction_1_Y	
material_direction_1_Z	
material_direction_2_X	second coordinate vector
material_direction_2_Y	
material_direction_2_Z	
material_direction_3_X	third coordinate vector
material_direction_3_Y	
material_direction_3_Z	

9.9. Echo

Results, in ASCII format, from the various intermediate calculations may be output to a results file, e.g. `example.rslt`, where the file name is generated by taking the base name of the input deck (without the extension) and adding the extension `rslt`. 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 9-24.

```
ECHO
  materials
  elements
  Jacobian
  mesh
  nodes
  MPC
END
```

Joint2G elements are supported as virtual blocks. With virtual blocks, the element variable such as element force, **eforce**, is also written to the results file.

Use `input off` to not echo anything .

Table 9-24. – Echo Section Options.

Option	Description
ADiag	diagonal of dynamics matrix
acceleration	nodal accelerations (better in output section)
block	block wise mass properties (used only following mass)
debug	debug output
displacement	nodal displacements (better in output section)
EForce	element force for beams
ElemEigChecks	element eigenvalues
elements	element block info, i.e. what material, element type, etc
Elmat	element material properties
energy	element strain energy and strain energy density
orient	element orientation (geometry-defined)
fatigue	fatigue related parameters
force	applied forces (better in output section)
GEnergies	global kinetic and strain energy sums
block_energies	block kinetic and strain energies
input (<bool>)	echo of post-Aprepro input *(for parse errors)* – default = `input on`
input_summary	summaries of many sections
Jacobian	block summary of Jacobians
KDiag	diagonal of stiffness matrix
line_weld	Line-weld-specific output variables
mass	mass properties in the basic coordinate system
materials	material property info, e.g. E, G
memusage	prints per processor per task memory use to results file and an external text file
mesh	summary of data from the input **Exodus** file
mesh_error	mesh discretization error metrics
ModalVars	modal force and amplitude for modal solutions (echo section)
MPC	MPC equations
NLresiduals	turns on residual output per iteration of the Newton loop for nonlinear solution methods
nodes	nodal summary
residuals	residual vectors
rhs	Right Hand Side vector (better in output section)
strain	element strains at centroids
stress	element stresses at centroids
subdomains “0:3:6,10”	Controls which processor will output results file
threading_summary	threading summary table
timing_summary	timing and threading summary tables
used	summary of unused variables (see section 9.9.1)
velocity	nodal velocities (better in output section)
vonMises	von Mises stress only
vrms	RMS quantities (random vibration only)

9.9.1. *Used*

The **used** option will list any unused variable requests to the log/rslt file. For example, requesting **acceleration** in a **statics** solution case will result in the following message:

```
NOTICE: 'out' analysis has no support for these output variables:
    'acceleration'
```

Without it, the following general note will be printed when unused variables are found.

```
Some requested output variables do not apply to all solution cases.
See more information and a list of valid exodus output options by
requesting 'used' in the ECHO section.
```

9.9.2. *Mass Properties*

The mass properties may only be reported in the `Echo` section only. The mass properties are the total mass, the center of gravity and the moments of inertia of the system. They are reported in the basic coordinate system. Furthermore, moments are about the origin, not about the center of gravity. Masses are reported in a unit system consistent with the input, with or without the **wtmass** parameter (see Section 4.4).

Note that dead/omitted blocks do not contribute to the total mass of your system.

Although mass properties are reported for any problem, they are undefined and nondeterministic in certain categories of analyses. Any model with a Superelement 7.32 has undefined mass properties. Also, mass properties are undefined for all acoustic problems, including structural acoustic models and Wet Modes simulations. Finally, mass properties are undefined for all Waterline simulations.

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 following the `mass` option as shown below.

```
ECHO
  materials
  elements
  mass=block
  nodes
END
```

Note: dead/omitted blocks (or in general any block with zero mass) will be reported in the block-wise mass properties table with zero mass, zero moment of inertia, and “N/A” center of gravity.

9.9.3. *Multipoint constraints*

Text descriptions of the MPC equations are output to the result file using `MPC` . This is 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.

9.9.4. *ModalVars*

modalvars text output which contains modal forces and modal amplitudes for modal based superposition solutions including “modaltransient” and “ModalFrf”. Four text files are written: `Qdisp.txt`, `Qvel.txt`, `Qaccel.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. In terms of the physical force at time t_n, $F(t_n)$, and the i^{th} eigenvector ϕ_i, the modal force displacements, and accelerations are

$$\begin{aligned}
f_{q_i}(t_n) &= \phi_i^T F(t_n) \\
u(t_n) &= \sum_i \phi_i q_i(t_n) \\
\dot{u}(t_n) &= \sum_i \phi_i \dot{q}_i(t_n) \\
\ddot{u}(t_n) &= \sum_i \phi_i \ddot{q}_i(t_n)
\end{aligned}$$

The expressions in the frequency domain are

$$\begin{aligned}
f_{q_i}(\omega_n) &= \phi_i^T F(\omega_n) \\
u(\omega_n) &= \sum_i \phi_i q_i(\omega_n) \\
\dot{u}(\omega_n) &= \sum_i \phi_i \dot{q}_i(\omega_n) = i\omega \sum_i \phi_i q_i(\omega_n) \\
\ddot{u}(\omega_n) &= \sum_i \phi_i \ddot{q}_i(\omega_n) = -\omega^2 \sum_i \phi_i q_i(\omega_n)
\end{aligned}$$

The text files are readable by either MATLAB or MS/excel.

9.9.5. Subdomains

In parallel calculations, one results file is written per subdomain. Only data associated with that subdomain are written to the file. By default, results are only written to subdomain 0 (the root processor), and in the results for subdomain 0 will *always* be output. Use the “subdomains” option to specify additional subdomains for which data will be written. The **subdomains** specification is made using a MATLAB like string, as detailed in Section 4.1. For example,

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

selects subdomains 0, 2, 4, 6 and 8 (again, subdomain 0 will always be output, and so it is redundant here). 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.

9.9.6. Memusage

The **Memusage** keyword selects memory usage information output for the results file. In addition, it also requests that a per processor, per task break down be written to a separate text file. The memory in the text file is shown in megabytes. 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"
```

The primary use of the memory usage printing is to diagnose where or why memory is being exhausted by an analysis.

9.9.7. Elmat

The bulk modulus, mass density and shear modulus of each element are reported if the `Elmat` keyword is added to the echo section.

This page intentionally left blank.

10. CONTACT

10.1. 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 4.11. 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 some ways of combining surfaces that have not been consistently meshed. The simplest method constrains the nodes of one surface (node-surface) to lie on the faces of another surface (face-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.

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** section is the same. For structural-structural interfaces, the nodal displacements on the node-surface are constrained to lie on the faces of the face-surface. In the last case, the nodal acoustic pressures on the node-surface are constrained to match the interpolated value of pressure on the face-surface.

For tied structural-acoustic interfaces, it is necessary to ensure a weak continuity of both stress and displacement (velocity) across the wet interface.[65,45] Also, we recommend that the acoustic surface be defined as the face-surface (and hence should have its sideset number listed first in the input deck). Defining the structural surface as the face-surface sometimes causes an error related to singular subdomain matrices.

A simulation may have multiple tied surfaces as long as certain requirements are met. For example, acoustic-acoustic and tied structural-acoustic data blocks in the same input deck are supported. However, it is necessary that each sideset be exclusively attached to either structural elements or acoustic elements. A sideset containing both acoustic and structural elements is not supported. This does not restrict the possible types of analysis. It can increase the number of **tied data** blocks. However, this extra input will reduce confusion and likely also reduce potential modeling errors.

The keyword `transverse` controls the constraints on transverse displacements. Transverse displacements can be `tied` or `slip`. Its default is `tied`. The tied option is the standard inconsistent tied surface approach. The slip option only constrains normal degrees of freedom between the 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. The `friction` option for specifying a simple friction model is not currently supported.

In a `tied data` block, if the tied contact is inconsistent, i.e., the method is not mortar or other options, then by default the gap is removed. Gap removal only works with `tied data`. In the tied data section, the order of the surface ids is important. The first surface id becomes the face surface, and the second becomes the node-surface. Gap removal moves node-surface nodes to the face surface. Set `gap removal` to *off* to skip gap removal. ACME's gap and push back vector quantities provide the gap. Updated coordinates instead of the original coordinates appear in the output **Exodus** file, and are also used by the system matrices.

Debugging tied contact is easier using the gap removal solution case 5.32.

10.1.1. Contact Normal Vectors

For all the contact type interactions, including tied surfaces, tied joints, and contact definition the algorithms used restrict the search to matching faces that have opposing normal vectors. For solids, this is seldom an issue. The normal vectors for a solid are always outward from the solid, so two interacting solids (unless they occupy the same volume), will naturally have opposing normal vectors. 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 too. 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 10-1.

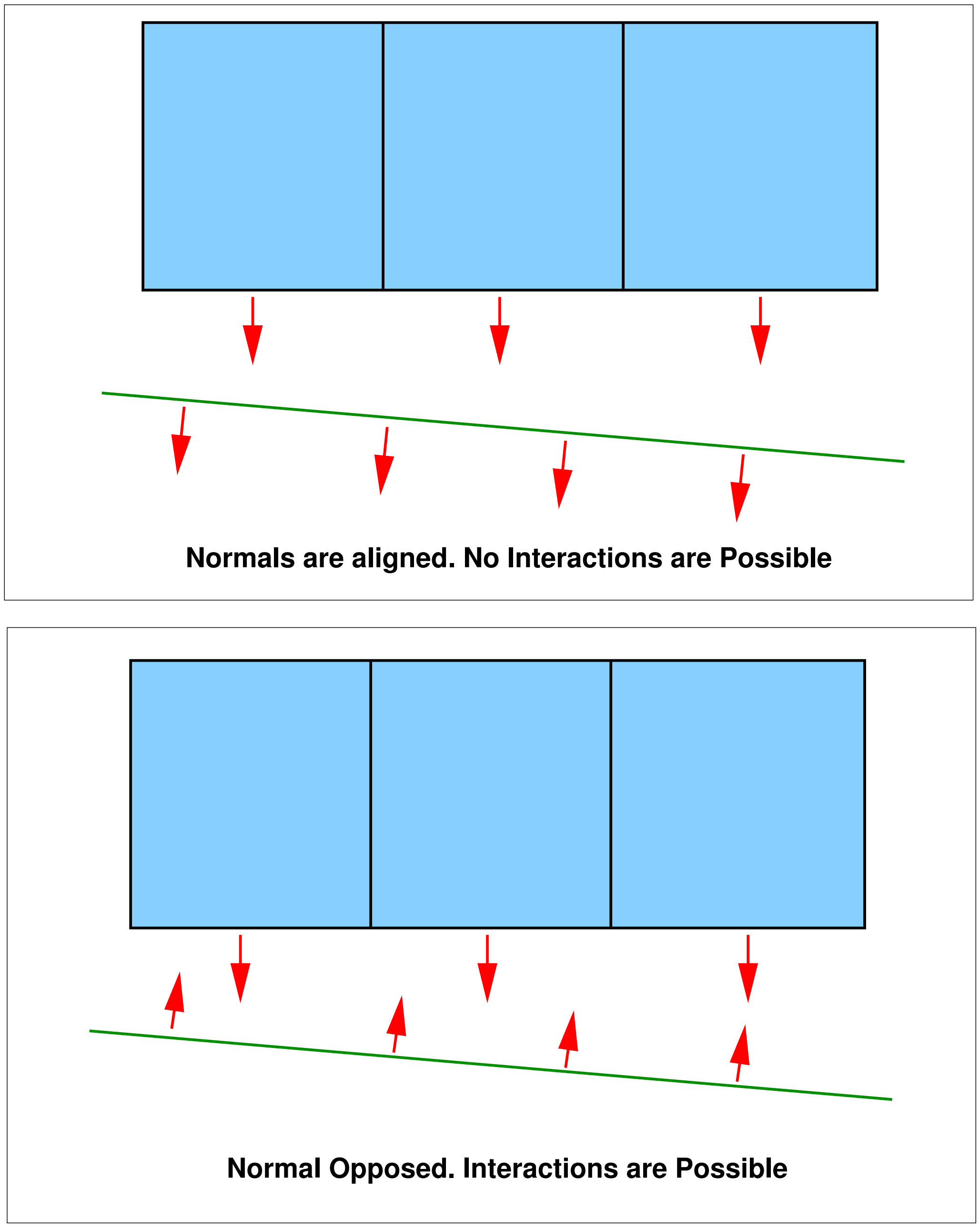

Normals are aligned. No Interactions are Possible
Normal Opposed. Interactions are Possible

Figure 10-1. – Shell Normal in Contact or Tied Interactions.

10.1.2. 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 many 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 different mortar methods are available. Both constrain surfaces together in an integral (or weak) sense. Standard mortar methods are somewhat simpler, but can result in a constrained system which fully couples all the nodes of both surfaces together in a single constraint. Dual mortar methods are much more friendly to the linear solver, as the constraint system decouples the constraints similarly to what is seen in node-face contact. The dual mortar method is the default.

Mortar methods are specified by adding `mortar` to the `tied data` block. To select the type of method, standard or dual, in the **parameters** block specify `MortarMethod=standard` or `MortarMethod=dual` respectively. [1]

10.1.3. Node to Face

Tied surfaces are specified by a listing of face-surface and node-surface side sets. Any number of tied data blocks may be specified in the input. Each tied data block represents a *single* logical pairing of constraint side sets.

```
TIED DATA
    tied faces 12
    tied nodes 18
    name "tying_12-18"
    transverse slip
    search tolerance = 1e-7
    edge tolerance = 1e-8
    gap removal = on
END
```

In the example above, sideset 12 is the face-surface. Side set 18 is the node-surface. Each node in the node-surface may be tied to the nearest face in the face-surface by a constraint equation. 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-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 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 10-2.

[1]There is no means of applying standard mortar methods to some interactions, and dual mortar methods elsewhere.

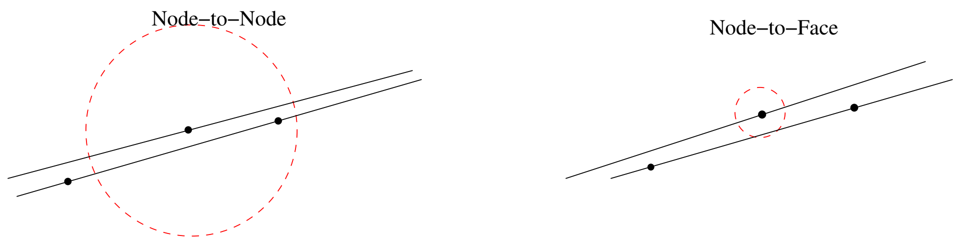


Figure 10-2. – For node-to-node searches the *search tolerance*, must be large enough to capture nearby nodes. For node-face searches (as used in tied surfaces), it should only capture the nearby surface.

Special care should be used when using the "edge tolerance". If this tolerance is too large, non-intuitive interactions can be created.

> The current implementation ties a face-surface that consists of the two-dimensional faces of shell or solid elements. It is not possible to tie a node to the one dimensional edges of shell elements.

The relevant parameters for tied surfaces are shown in Table 10-1.

Table 10-1. – Tied Surface Parameters

Parameter	type	description
Name	*String*	name of tied data block, useful for diagnosing error messages, defaults to tied
Surface	*integer pair*	face-surface and node-surface sidesets separated by comma or space
Tied Faces	<*sideset*>	face-surface sideset
Tied Nodes	<*sideset*>	node-surface sideset
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.
Method	*String*	inconsistent *(default most solvers)* mortar
Transverse	*String*	tied *(default)* slip *(transverse displacements can slip)*
Gap Removal	*String*	Yes *(default: for inconsistent only)* No
smooth angle	*Real*	maximum angle for smoothing (def=30)
smoothing resolution	*String*	"node" or "edge" based

Smoothing parameters may be needed to control smoothing of the normal. Figure 10-3 illustrates the normal definitions on a faceted surface. The discontinuity in normal vectors can be an important consideration on curved surfaces where faceting affects tangential sliding. Smoothing parameters are illustrated in Figure 10-4 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^o.

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.

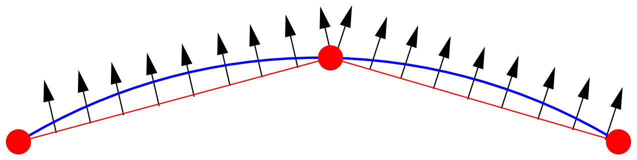


Figure 10-3. – Normal Definitions on Faceted Geometry. When low order elements are used to describe a curved boundary, the normal is poorly defined at the edge of the facets.

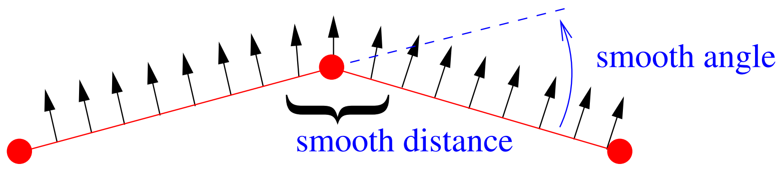


Figure 10-4. – Smoothing Parameters for Surface Normal Vectors. No smoothing occurs for faces that are misaligned by more than the specified "smooth angle". Within the "smooth distance", normal vectors vary linearly with relative distance from the node.

10.2. Contact Definition

The contact definition block provides flexible syntax to define tied MPCs. The contact definition provided a similar capability to Tied Data (see Section 10.1) but with a more powerful and flexible ways to define contact surfaces. The contact definition capability leverages the same Dash contact algorithms used in Sierra SM. Both the syntax and capability of the Dash contact definition is compatible with SM recommending Dash contact for both SM/SD **hand-off** analyses and analyses with a lot of general contact.

A brief description of contact definition commands is given here. More detailed descriptions of how contact surface and interaction definition commands function can be found in the Sierra Solid Mechanics User Manual.[47]

The full set of available contact commands are as follows:

```
BEGIN CONTACT DEFINITION <name>
  contact [surface|nodeset] <surf_name> contains <strings>
  skin all blocks = off|on(off) [exclude <names>]
  gap removal = off|on(on)
  Begin Interaction Defaults
    general contact = off|on(off)
    self contact = off|on(off)
    normal tolerance = <Real>
    normal tolerance behavior = auto|user_defined
    constraint formulation = Node_Face|Face_Face(Node_Face)
    friction model = tied|frictionless(tied)
    cutoff variable <String> ‘<’|‘>’ <Real>
  End Interaction Defaults

  Begin Interaction
    normal tolerance = <Real>
    normal tolerance behavior = auto|user_defined
    side a = <string_list>
    side b = <string_list>
    surfaces = <string>
    interaction behavior = No_Interaction
    constraint formulation = Node_Face|Face_Face
    friction model = tied|frictionless
    cutoff variable <String> ‘<’|‘>’ <Real>
  End Interaction

  Begin Dash Options
    separate disconnected mesh components = true|false(false)
    ignore_shells = true|false(false)
  End Dash Options
END CONTACT DEFINITION
```

Information is presented in the format command = option (default) format. For example gap removal is on by default. Debugging contact is easier using the gap removal solution case 5.32. Gap removal is another name for initial overlap removal.

A `begin interaction` section can only define interacting surfaces, if the surface is assigned a local name. To define a surface named `surf_name`, use

```
contact surface <surf_name> contains <entity_strings>
```

Here `‘entity_strings’` may reference blocks, sidesets or nodesets in the input mesh. It may reference a block, surface, or node set by its name. Or mesh entities can be accessed using the **Sierra/SM** convention, by index using ’block_##’, ’surface_##’ or ’nodelist_##’.

General contact is a far-reaching capability. If there are 2 (i.e. more than one) interaction, and general contact is on, then the contact module automatically detects all interactions between any pair of surfaces (except self interactions, and assuming that self contact is off).

10.2.1. Defining Contact Surfaces

Each contact definition block is a self-contained description of contact surfaces and how those contact surfaces interact. Several options are available for defining contact surfaces as shown in the following examples.

```
//  Create contact surface from exterior skin of blocks
contact surface fixtures contains block_1 block_9 block_12

//  Create contact surface from union of sidesets
contact surface bolt_flange contains surface_1 surface_10

//  Shortcut to create contact surface based on the
//  exodus part names.
contact surface block_7
contact surface surface_10
contact surface bearing contains bearing1 bearing2

//  Create contact nodeset from union of node sets
contact nodeset ns1 contains nodelist_7 nodelist_11

//  Create contact nodeset containing all nodes in a
//  set of blocks
contact nodeset ns2 contains block_15 block_19

//  Generate exterior skins for all blocks
skin all blocks = on

//  Generate exterior skins for most blocks
skin all blocks = on exclude block_7 block_11
```

The skin of a finite element block contains all the exterior faces. When using 'skin all blocks' one contact surface is created for each block in the mesh, the contact surface is given the same name as the block. The block skinning algorithm is described in the Sierra SM Users Manual. For a model using only solid elements block skinning relieves the user from having to set an extensive number of sidesets in the input mesh.

Note the commands that create contact surfaces from blocks only work on solid and shell elements (hexes, tetrahedra, wedges, quads, triangles, etc.). If “ignore_shells” is set to true in the Dash options block, then all-to-all contact will ignore shells. Sidesets can be used to define contact surfaces on either solid or shell blocks. Contact nodesets can be defined on any element type, solid, shell or beam. When defining interactions a contact nodeset must be paired with a sideset or block skin to find node-face constraints. Two contact nodesets cannot directly interact.

In rare situations poorly posed cyclic/self contact constraints are created that are problematic for the linear solver. First the model is divided into disjoint components (without shared nodes). Pairs of disjoint components then tie to each other with one sided node-face interactions. Multipoint constraints tie nodes on one side of the interface to faces on the other side of the interface. Parts sharing nodes may have no unique node-face pairing. A conformally meshed part that contacts itself has no unique pairing.

If no unique pairing exists, nodes on side A of the interface generate MPCs with faces on side B and also nodes on side B generate MPCs with the faces of side A. The constraints may be over-determined. Some redundant constraints are removed. Determining unique constraints is an open problem.

The Interaction Weight Matrix shows the potential node-face interactions specified in the input deck. It is in the log file. A pair of surfaces must be in proximity to actually interact.

$$\begin{cases} \text{disjoint} & \text{if} IJ = JI = 0 \\ \text{dependent node} I & \text{if} IJ = 1 \text{and} JI = 0 \\ \text{symmetric} & \text{if} IJ = JI = H \\ \text{error} & \text{otherwise} \end{cases} \tag{10.2.1}$$

Ideally the weight is either 1 or 0. This indicates one-way node-face pair where nodes on one side of the interface interact with faces on the other side. Self contact is denoted by H for half. For self contact nodes on side A of the interface interact with faces on side B and also nodes on side B interact with faces one side A. Such self contact constraints are often redundant and cyclic. The attempt to remove and make the contact constraints uniquely determined creates messages and warnings about removed and redundant constraints. Even after the redundant constraint removal step the self contact constraints can cause solver robustness and accuracy issues. Thus, self contact should be avoided.

The disconnected component finder is available for setting up contact surfaces. Say a mesh contained a flashlight with four batteries, and the four batteries were all in the same element block in the mesh. The disconnected component finder would split this battery block into four separate contact surfaces. The disconnected component finder is useful for setting up interactions in a way that avoids self contact. See the **Sierra/SM** User Manual[47] for more details on use of the disconnected component finder.

10.2.2. Setting up Contact Interactions

Once the contact surfaces are defined, the next step is to set up the interactions between those contact surfaces. The `interaction defaults` block can be used to define both which surfaces will interaction with each other and how those surfaces interact. One and only one interaction defaults block may be present in a contact definition. The `interaction` block can enforce contact between specific surface pairs and set the interaction parameters for that pair (overriding the interaction defaults behavior for the surface pair.) Any number of `interaction` command blocks may be present in the contact definition.

Contact is used to tie structures together that are in adjacent. The normal tolerance defines how far away from a face a node can be and still find contact. By default, a reasonable normal tolerance is automatically computed in the Dash contact library of 15% of the characteristic element length. Thicknesses of shell elements are also considered in the default tolerance. For shell blocks, the default search tolerance is set to at least sixty percent of the maximum element thickness of the elements in the block.

```
BEGIN CONTACT DEFINITION <name>
  Begin Interaction Defaults
    General Contact = Off|On(Off)
    Self Contact = Off|On(Off)
    Normal Tolerance = <real>
    Normal Tolerance behavior = auto|user_defined
    Constraint Formulation = Node_Face|Face_Face(Node_Face)
    Friction Model = Tied|Frictionless(Tied)
    cutoff variable <String> ‘<’|‘>’ <Real>
  End Interaction Defaults

  Begin Interaction
    Side A = <string_list>
    Side B = <string_list>
    Surfaces = <string_list>
    Normal Tolerance = <real>
    Normal Tolerance behavior = auto|user_defined
    Constraint Formulation = Node_Face|Face_Face
    Friction Model = Tied|Frictionless
    Interaction Behavior = No_Interaction
    cutoff variable <String> ‘<’|‘>’ <Real>
  End Interaction
END CONTACT DEFINITION
```

Command options:

- General Contact: `On` means that every surface will contact every other surface. By default, the Dash contact library will pick which surface is used for nodes and which for faces in each surface-to-surface pairing automatically. Generally the surface with the smaller characteristic face (defined by the smallest face width anywhere on the surface) will provide the nodes and the coarser the faces. Note, if surfaces have a large variation in mesh size the less-optimal node/face pairing may be selected by default as the default node/face pairing is controlled by the smallest surface face, not the characteristic size of faces actually in contact. Additional considerations may also be taken into account for selection of node and face surfaces such as avoiding cyclic constraints.

- **self contact**: Self contact on indicates a surface may contact itself, this could occur if a structure folds over on itself. Self contact should generally be avoided in SD as it leads to over-constraint problems.

- **normal tolerance**: The default face-sized based search distance can be overridden by manually specifying the normal tolerance. This tolerance has units of length.

- **normal tolerance behavior**: This option controls the behavior when the user-defined normal tolerance is smaller than the default (15% of element characteristic length) threshold.

  auto : use the maximum of the user-defined and auto-generated tolerances. Note: this means that user-defined tolerances smaller than the default threshold will be **ignored**. This is the equivalent to the behavior in **Sierra/SM**.

  user-defined : respect the user-defined tolerance, even if the auto-generated tolerance is larger. This is the default behavior in **Sierra/SD**.

  Note: as of the 5.26 release, the behavior of **Sierra/SD** and **Sierra/SM** differs when a user-defined normal tolerance is smaller than the default 15% threshold. **Sierra/SM** behavior is equivalent to `normal tolerance behavior = auto`, while **Sierra/SD** defaults to `normal tolerance behavior = user-defined`. To match **Sierra/SM** behavior (or **Sierra/SD** behavior prior to 5.26), use the interaction (defaults) option `normal tolerance behavior = auto`.

- **constraint formulation**: The constraint formulation line defines the constraint type to be used. Only the node_face option should be used with **Sierra/SD**. The face_face option is experimental.

- **friction model** : The friction model line selects the type of contact constraint, either `tied` or `frictionless`. Tied contact ties all translational DOFs together at the interface preventing any normal or tangential motion. For structural problems each of the three translational degrees are constrained together. Rotational DOFs are never tied. For acoustic-acoustic contact the acoustic degree of freedom is tied together. For structure-acoustic contact the normal-motion of the structure is tied to the acoustic degree of freedom. The frictionless keyword selects sliding contact that is tied in the normal direction only. For frictionless structural contact only the surface-normal motion of the contacting surfaces are constrained together, the surfaces are free to slip in the tangential directions. The normal direction for the frictionless constraint is taken from the normal of the contacting face. For acoustic-acoustic or structural-acoustic contact frictionless contact is equivalent to tied.

- **SIDE A** `SIDE B` : For node-face contact the nodes are defined by the *B* surface and the faces by the *A* surface. If multiple surfaces are given for side A or side B, then the faces of each side A surface will be constrained to then nodes of each side B surfaces.

- **surfaces** defines a set of surfaces in contact. The Dash library picks the node-face pairing automatically based on relative mesh density and other considerations such as avoiding cyclic constraints. If more than two surfaces are given a contact interaction will be formed between each surface in the list and each other surface in the list. Using $SIDEA$ or $SIDEB$ and $SURFACES$ commands in the same interaction section will cause an error.

- Interaction Behavior The special interaction behavior command allows turning off contact between specific surface pairs. The no_interaction option would generally be paired with interaction defaults general contact on and used to turn off specific pairings where contact should not occur, such as slide lines.
- **cutoff variable** The `cutoff variable` interaction option enables users to further filter contact based on any nodal input variable. Fine-grained contact information determined by **Sierra/SM** may be passed to **Sierra/SD** to refine its contact constraints. Including the following lines in a contact definition will ignore contact where the **Sierra/SM** `celement` field is below 0.55 on the nodes of the node-face constraints.

  ```
  Begin Interaction Defaults
    cutoff variable celement < 0.55
  End Interaction Defaults
  ```

  Multiple such lines may be defined in a single interaction or interaction defaults block. This would enable the filtering of constraints based on the union of multiple nodal variables, or only retaining contact where a variable is inside a range (a, b) (i.e. `cutoff variable var_name < a` and `cutoff variable var_name > b`). This union effect also applies when cutoff variables are defined in both the interaction defaults block and an interaction block. This means that, unlike other interaction defaults, the default cutoff variables defined here cannot be overridden at a single interaction, they will simply add an additional filter. If it is desirable to apply a cutoff variable at all interactions except for 1, the cutoff variable would have to be defined at all other interactions explicitly.

10.2.3. Gap removal

As with tied surfaces 10.1, 5.32 the Contact Definition by defaults removes gap from the interaction constraints. Gap removal can optionally be turned off. All contact constraints are node-on-face contacts. Gap removal is accomplished by moving the node to the face.

```
GAP REMOVAL = OFF|ON(ON)
```

The rewards and risks of gap removal are described in Section 5.32.

`Gap removal` will trigger output of an element quality table listing the worst element in each block before and after gap removal. The element quality metric used in this table is not consistent with Sierra Solid Mechanics. Element condition number is used instead. This maintains consistency with other features within Sierra Structural Dynamics, and with other Sierra applications such as Cubit.

The table will begin with the header shown below:

```
=================== ELEMENT CONDITION QUALITY INFORMATION ====================
A value of 1.0 is an ideal element. As the value approaches INF the quality is
decreasing. Values less than or equal to zero indicate the element cannot
```

```
compute condition number.  For example a beam, conmass, Rbar, etc. A value of
N/A also means no shape metric exists for that element topology. Element quality
can be plotted on the mesh with the ElementQuality variable, triggered by the
gap\_removal solution case.
-------------------------+-------------+------------+-------------+-------------
```

The table contents below the header are too wide to fit in this document.

10.2.4. Examples

```
// Most basic contact definition to tie everything that touches
BEGIN CONTACT DEFINITION
  skin all blocks on
  BEGIN INTERACTION DEFAULTS
    general contact = on
  END INTERACTION DEFAULTS
END CONTACT DEFINITION
```

```
// Tie a few specific surfaces, like in 'tied data' block
BEGIN CONTACT DEFINITION
  contact surface s2 contains surface_2
  contact surface s3 contains surface_3
  contact surface s4 contains surface_4
  BEGIN INTERACTION
    side A = s2
    side B  = s3 s4
    normal tolerance 0.25
  END INTERACTION
END CONTACT DEFINITION
```

```
//  All-to-all contact with some custom tolerances
BEGIN CONTACT DEFINITION
  skin all blocks = on
  BEGIN INTERACTION DEFAULTS
    general contact = on
    normal tolerance 1e-3
  END INTERACTION DEFAULTS
END CONTACT DEFINITION
```

```
// Tie the nodes of the two bolt flanges to a fixture block
BEGIN CONTACT DEFINITION
  contact nodeset boltFlange1 contains nodeset 901
  contact nodeset boltFlange2 contains nodeset 902
  contact surface fixtures contains blocks_1 block_3 block_4
  BEGIN INTERACTION
    side A = fixtures
    side B = boltFlange1 boltFlange2
  END INTERACTION
END CONTACT DEFINITION
```

```
// All-to-all contact, but turning off tying of a specific
// surface pair and using sliding contact for different
// surface pair.
BEGIN CONTACT DEFINITION
  skin all blocks = on
  BEGIN INTERACTION DEFAULTS
    general contact = on
  END
  BEGIN INTERACTION
    surfaces = piston piston_housing
    friction model = frictionless
  END INTERACTION
  BEGIN INTERACTION
    surfaces = drive_shaft drive_bearing
    interaction behavior = no_interaction
  END INTERACTION
  END CONTACT DEFINITION
```

10.2.5. Notes and Usage Guidelines

- Multiple 'begin contact definition' blocks may be included in a single analysis. However, as with **Sierra/SM** using multiple contact definition blocks is generally discouraged. If multiple contact definitions are used, then the surfaces used in the contact definitions can not overlap. If the same surfaces are used in multiple contact definitions duplicate and/or incompatible constraints may be found between the contact definition blocks causing solver difficulty.
- Contact constraints are ultimately enforced by translational MPCs. The MPCs generated by contact tie only translational degrees of freedom.
- Self contact (contact constraints generated from a surface folding over on itself) are often over-determined and cannot be enforced accurately. Though the contact definition can find such self contact constraints, the use of self contact should be avoided.
- If contact constraints are defined at a significant gap those constraints will artificially impede the rotation of the model. Gap removal can help with this issue, but it is recommended to only use contact to tie objects that are in close physical proximity.

- Use the 'outputs' option [constraint_info] to visualize more information about the generated contact constraints. The nodes of a node-face contact constraint will have a '1' for [node_face_mpc_count] output. The [mpc_status] nodal field will be painted with '1' for any node involved in a contact constraint on either the node or face side. If any nodes have the [node_face_mpc_both_node_and_face] flag this could indicate an issue of over-constraint.

10.2.6. Differences Between SM and SD Defaults

- By default, SD uses node-face constraints. SM uses face-face constraints by default. Face-face constraints may be optionally used within SD. However, face-face constraints are considered experimental for SD at this time and not recommended.
- By default, SD will try to remove the gap from contact constraints. SM does not have a gap removal option at this time.
- In SD the friction model defaults to 'TIED'. In SM the friction model defaults to 'FRICTIONLESS'.
- The SD friction model is a linearization of the SM frictionless model. A key difference is that in SD the frictionless constraint is free to slide on the face but can have no motion in the normal direction of the face. In SM the sliding along the face is also unconstrained, the node is prevented from penetrating the face, but differing from SD the frictionless constraint can open gap and separate the surfaces. This different behavior for the positive and negative normal directions is a fundamentally nonlinear behavior not applicable to linear structural dynamic analysis.

10.3. Lofted Surfaces and Gap Removal

Lofted surfaces are important because analysts often build meshes with an initial gap between the surfaces. If standard methods are used to tie the surfaces, but the separation (or *lofting*) is not taken into account, then the constraints are no longer consistent with rigid body motion. Generally this means that the rotational rigid body motion introduces strain into the system.

There is a gap removal 5.32 solution.

10.3.1. Example

This example illustrated in Figure 10-5 uses the coordinates listed in Table 10-2. In this figure, the face (represented by nodes 1-4) constrains each of the 3 nodes (nodes 5, 6, and 7.) Node "6" is on the face. It will be clear that standard methods apply the proper constraints. However, nodes "5" and "7" are offset from the face. As a consequence, constraint equations written for a node constrained to the face introduce errors when applied to the lofted node.

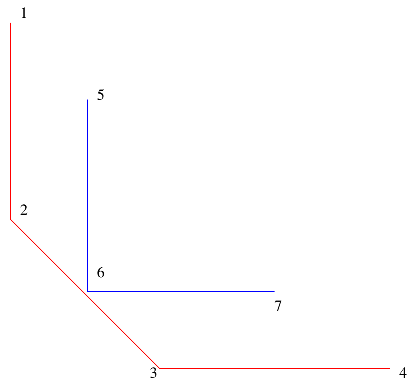


Figure 10-5. – Lofted Constraint Example.

Node	Coordinates		
1	0	4	0
2	0	2	0
3	2	0	0
4	4	0	0
5	1	3	0
6	1	1	0
7	3	1	0

Table 10-2. – Coordinates of Face (red) and Nodes (blue).

10.3.2. Projection Approach

The constraint equations from a conventional approach (meaning that the constraints are written by projecting the node location to the plane of the face, but not adjusting for the lofting) are shown in Table 10-3. These equations are not orthogonal to rigid body modes, and as a consequence, there are only two zero energy modes for this system rather than the 3 we anticipate. [1]

$$\begin{aligned}
u_x(1) + u_x(2) - 2u_x(5) &= 0 \\
u_y(1) + u_y(2) - 2u_y(5) &= 0 \\
u_x(2) + u_x(3) - 2u_x(6) &= 0 \\
u_y(2) + u_y(3) - 2u_y(6) &= 0 \\
u_x(3) + u_x(4) - 2u_x(7) &= 0 \\
u_y(3) + u_y(4) - 2u_y(7) &= 0
\end{aligned}$$

Table 10-3. – Conventional Constraint Equations.

[1]All motion out of the xy plane has been eliminated.

These constraints are represented by the matrix C, where the x and y dofs are grouped together.

$$C = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix}$$

The three rigid body vectors (in this 2D frame) are,

$$R = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ -3 & -1 & -1 & -1 & 1 & 1 & 1 & 3 & -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where the first two vectors represent translations, and the last is a rigid body rotation about point "6". The product of $C * R'$ can be computed.

$$C * R^T = \begin{bmatrix} 0 & 0 & -2 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix}$$

Each row of this matrix corresponds to a constraint equation from Table 10-3. Each column is associated with one of the three rigid body vectors. The translational rigid body vectors are orthogonal to the constraint matrix; the products are zero and no strain is induced. However, rotation about node 6 induces strain. The constraints are not invariant to rotation.

In a transient dynamic analysis with modest rotations and small gaps the effects of these constraint errors are often imperceptible. However, for large rotations or large gaps they may become apparent. They are always observable in modal analysis where they manifest as nonzero rigid body modes.

Mitigation Strategies As demonstrated in the previous section, constraint errors can introduce resistance to either translational or rotational rigid body motion. There are several strategies to mitigate these issues.

1. The analyst building the model ensures there are no projection errors.
2. Correct the initial geometry (using gap removal) so there are no projection errors.
3. Modify the constraints through algebraic means to ensure that they are orthogonal to rigid body motion.
4. Use the constraints appropriate to the lofted geometry. The software to do this is currently only available for spot welds.

If these methods cannot be applied, then the analysis must absorb the errors due to the constraint errors introduced by projection.

10.4. Spot Welds

Spot Welds are a flexible alternative to contact in which each node-face interaction is given a stiffness in the normal and tangential directions. Spot Welds can be used to represent discrete attachment points such as rivets, or as a scalable alternative to tied joints. In either case, **Sierra/SD** Spot Welds share syntax and functionality with **Sierra/SM**, with a slightly altered implementation.

An individual Spot Weld is defined by a surface and a nodeset. It may represent a bolt/rivet/screw or something similar. An individual Spot Weld stiffness has units of force/length.

An *area-weld-mode* Spot Weld is defined by a pair of surfaces. It may represent a cohesive zone [47]. Its stiffness has units of force/area.

Spot Welds may be specified between two parts which are not touching. A gap at a Spot Weld interface *will not cause grounding* of rotational rigid body modes.

10.4.1. Syntax

```
SPOT WELD
  nodeset = <list(nodeset)>
  node set = <list(nodeset)>
  second surface = <list(sideset)>
  remove node set = <list(nodeset)>
  sideset = <list(sideset)>
  side set = <list(sideset)>
  surface = <list(sideset)>
  remove surface = <list(sideset)>
  normal displacement function = <function>
  normal displacement scale factor = <real>
  tangential displacement function = <function>
  tangential displacement scale factor = <real>
  ignore initial offset = <bool>
  search tolerance = <real, gt 0>
END
```

This table is a little confusing at first glance, and a few words of explanation are helpful. The allowable inputs are either 'sideset+nodeset' or 'sideset+second_surface'.

The remove surface and remove node set options define the spot weld by boolean operations of multiple nodesets/sidesets. Some (maybe all) SD developers are less confident that these remove options are actually hooked up fully in SD.

The "ignore initial offset" option affects where the stiffness functions are evaluated if the spot weld is defined with an initial gap.

Spot welds are parallel scalable in a way that tied data is not scalable. Namely, while each Spot Weld is stored on a unique subdomain, Tied Data is implemented by storing the multi-point constraints on all processors.

Tangential inputs like stiffness are scalars representing a radial stiffness. In a cylindrical coordinate system a Spot Weld has a axial stiffness, a radial stiffness, and zero θ stiffness.

10.4.2. Outputs

Spot Welds support thirteen element variables, all of which are triggered by the `spot_weld` keyword, as detailed in table 10-4.

Table 10-4. – Available spot_weld outputs

Name	Description
spot_weld_scale_factor	Area scale factor if in second surface mode
spot_weld_normal_force	Normal force on dependent node
spot_weld_tangential_force	Tangential force on dependent node
spot_weld_norm_stiffness	Normal stiffness of element
spot_weld_tang_stiffness	Tangential stiffness of element
spot_weld_normal_displacement	Normal displacement of dependent node
spot_weld_tangential_displacement	Tangential displacement of dependent node
spot_weld_initial_offsetx	Initial gap vector, x component
spot_weld_initial_offsety	Initial gap vector, y component
spot_weld_initial_offsetz	Initial gap vector, z component
spot_weld_initial_normalx	Initial normal vector, x component
spot_weld_initial_normaly	Initial normal vector, y component
spot_weld_initial_normalz	Initial normal vector, z component

10.4.3. Specifying Spot Weld Stiffnesses

Sierra/SD models Spot Welds as linear spring elements, but shares syntax with **Sierra/SM**. Therefore, we assign stiffness to the elements by estimating the derivative of the normal and tangential displacement XY functions at the initial state. If the option `ignore initial offset` is set to yes, then the normal stiffness function tangent is evaluated at $X = 0$. If the option `ignore initial offset` is off (the default) the normal stiffness function tangent is evaluated at $X = initial\_gap$. The tangential function is always evaluated about $X = 0$.

Both functions should have **positive** slope to have positive stiffness, following static pull test conventions. The normal function should be defined for both positive (tensile) and negative (compressive) values of X. For the tangential function radial displacement is always positive so only the positive X portion of the function has meaning.

Analysts looking to specify the stiffness directly should use a linear function, for example: $y = x$ as the displacement function, then input their stiffness as a scale factor. For example:

```
FUNCTION y_equals_x
  type analytic
  evaluate expression ‘‘x’’
END

BEGIN SPOT WELD
  node set = nodelist_1000
  surface = surface_2000
  search tolerance = 1.0e-4
  normal displacement function = y_equals_x
  normal displacement scale factor = 1.0e+6
  tangential displacement function = y_equals_x
  tangential displacement scale factor = 1.0e+5
END
```

Best practices will evolve with time and usage, but perhaps consider the neighboring material stiffness divided by the distance between parts E/L as an initial stiffness guess in `second surface` mode. Try to avoid setting the stiffness too high (like $1e^{16}$), as it will impact the conditioning of the linear system and the ability of the solver to converge. Contact is the proper tool for infinite interface stiffness.

10.4.4. Usage at discrete points

This is the most basic use case, where we apply the same linear stiffness to each constrained node within search tolerance. Ideally, each node in the provided nodeset would represent a different attachment point in the model. One node per welded spot; a literal Spot Weld.

```
SPOT WELD
  sideset = bulkhead
  nodeset = rivet_nodes
  normal displacement function = y_equals_x
  normal displacement scale factor = 1e4
  tangential displacement function = y_equals_x
  tangential displacement scale factor = 2e3
  search tolerance = 0.25
END
```

10.4.5. Usage as an alternative to Tied Joint or Surface Contact

Here we use `second surface` to define the dependent side of the interaction instead of using `nodeset`. In this mode, the stiffness of each node-face interaction is scaled by the area of the faces attached to the dependent node. The user is now defining the stiffness per unit area of the joint. This mode is expected to provide a solution which converges with mesh refinement.

```
SPOT WELD
  sideset = independent_faces
  second surface = dependent_faces
  normal displacement function = y_equals_x
  normal displacement scale factor = 1e8
  tangential displacement function = y_equals_x
  tangential displacement scale factor = 1e8
  search tolerance = 0.1
END
```

This method has several advantages over Tied Joints.

- Each node-face interaction connects exactly one face to one node. This avoids the connectivity problems of Tied Joints when mesh density is high.
- The connections between parts are distributed across the interface, which preserves the bending and ovaling modes of the interfacing parts.
- Spot Welds create neither Type-1 nor Type-2 constraints; only stiffness.

Additionally, this method has potential advantages over Tied Data or Contact

- the spot welds surfaces can join separated surfaces in a way that does not impede global model rotation and does not require gap removal.
- the spot weld surfaces can be given a finite stiffness which can be tuned to the stiffness of an adhesive, or tuned based on experimental data.

10.5. Moving MPCs

Sierra/SD supports moving contact through the use of moving MPCs. This can be enabled using the solution parameter `nUpdateConstraints = 1` as shown in 4.4. There are several other recommended parameters to be used with moving contact. These will be described in this section.

The example input blocks below show how the moving MPCs may be enabled. The `predictorCorrector` is set to 0 within the `transient` block. In the `Parameters` block, the solver should be set to update constraints without updating the matrices by setting `solverReset = constraints`. There is an inexpensive preconditioner for acoustic problems with moving constraints. To select this preconditioner, set the `preconditioner_type`, `krylov_method`, `orthog`, and `max_numterm_C1` as shown in the example input deck below. The final two parameters in the `GDSW` block enable the block diagonal preconditioner with block size defined by the element condition number. To make a more powerful, but also more expensive, preconditioner increase the `max_element_condition`.

```
SOLUTION
  solver GDSW
  transient
    time_step 1.e-4
    nsteps 100
    nUpdateConstraints = 1
    predictorCorrector = 0
END

PARAMETERS
  solverReset = constraints
END

GDSW
```

```
  preconditioner_type DIAG
  krylov_method PCG
  orthog 0
  max_numterm_C1 0
  identify_low_quality_elements true
  max_element_condition 5
END
```

This page intentionally left blank.

11. EXAMPLE INPUT DECKS

Example input decks are shown for several types of analyses. The input deck is case-insensitive except for special cases such as file names,

11.1. Eigenvalue problem

The following input deck requests output to the **Exodus** file *hexplate-out.exo* of the mode shapes corresponding to four lowest frequency modes.

No one knows that the following paragraph means: 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 'Mode Shapes of Lowest Frequency Modes'
end
FILE  // finite element mesh
  geometry_file hexplate.exo
end
Boundary
  nodeset 77
  fixed
end
Outputs
  deform
end
Block 44 // The default is the Hex8b
  material 3
  hex8
end
Material 3
  name "steel"
  E 30e6 +/- 1 %
  nu .3
  density 0.288
end
```

```
Sensitivity
  values all
end
```

11.2. Anisotropic Material

The following input deck is an example of a hexahedron mesh with anisotropic properties.

```
Solution
         eigen
         title 'Anisotropic Format'
end

file
         geometry_file         mesh.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
           deform
end

block 1
         hex8
         material my_material
end

Material my_material
         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
```

11.3. Multiple materials

The next example shows the input for an **Exodus** model with many element blocks and materials. Keyword `lumped` in the `Solution` section selects a lumped (nearly diagonal) mass matrix.

```
Solution
         eigen
         nmodes 1
         title 'Dozen blocks and six materials'
         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
     deform
end
        // A block is required for each element block
block 1 //in the input Exodus (Genesis) mesh database.
         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  // Tip. Not capitalizing "material" here
         material 5 // helps to distinguish it
end      // from a Material section.
block 105
         wedge6
         integration full
         material 5
end
block 6
         material 6
end
block 106
         wedge6
         integration full
         material 6
end        // Each material referenced in a necessary block
Material 1 // must be defined here.  Extra materials are ignored.
         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          // Tip. Capitializing material helps to
Material 5   // distinguish it from a material in a block.
         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
```

11.4. Modaltransient

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

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
```

11.5. ModalFrf

In this **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

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
// a smooth pulse of duration .05 sec
// peaking near t=.02 sec at 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
```

11.6. Direct FRF

A `Direct FRF` is run with displacements written to to Frequency **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

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
```

11.7. Statics

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
block 1
         material 1
         QuadT
end
Material 1
         name "steel"
         E 30.0e6
         nu 0.25e0
         density 0.7324e-3
end
```

BIBLIOGRAPHY

[1] J. L. Aklonis and W. L. MacKnight. "Introduction to Polymer Viscoelasticity". In: Wiley, 1983. Chap. 1-6, pp. 1–316 (cit. on p. 257).

[2] D. J. Allman. "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems". In: *Computers and Structures* 19.1-2 (1996), pp. 1–8 (cit. on pp. 299, 464).

[3] Kenneth F. Alvin. "Implementation of Modal Damping in a Direct Implicit Transient Algorithm". In: *Presented at the 42nd AIAA/ASME/ASCE/AHS/ASC SDM*. Apr. 2001, p. 1589 (cit. on p. 284).

[4] Kenneth F. Alvin et al. "Incorporation of Sensitivity Analysis into a Scalable Massively Parallel Structural Dynamics FEM code". In: *Presented at the 5th U.S. Congress on Computational Mechanics*. Boulder, CO, Aug. 1999 (cit. on p. 114).

[5] ATA Engineering. *Attune User's Guide*. URL: http://www.ata-e.com/software/attune/Attune%5C_Users%5C_Guide%5C_v2/index.html (cit. on p. 111).

[6] Philip Avery, Charbel Farhat, and Garth Reese. "Fast Frequency Sweep Computations Using a Multi-point Pade-Based Reconstruction Method and an Efficient Krylov Solver". In: *International Journal for Numerical Methods in Engineering* 69.13 (Sept. 2006), pp. 2848–2875 (cit. on p. 179).

[7] C. G. Baker et al. "Anasazi Software for the Numerical Solution of Large-Scale Eigenvalue Problems". In: *ACM Transactions on Mathematical Software* 36.3 (2009), pp. 1–23 (cit. on p. 189).

[8] Jean-Louis Batoz, Klaus-Jurgen Bathe, and Lee-Wing Ho. "A Study of Three-Node Triangular Plate Bending Elements". In: *International Journal for Numerical Methods in Engineering* 15 (1980), pp. 1771–1812 (cit. on pp. 299, 464).

[9] T. Belytschko, CS Tsay, and WK Liu. "A stabilization matrix for the bilinear Mindlin plate element". In: *Computer Methods in Applied Mechanics and Engineering* 29.3 (1981), pp. 313–327 (cit. on p. 298).

[10] J-P Berenger. "Perfectly Matched Layer for the FDTD solution of wave-structure interaction problems". In: *Antennas and Propagation, IEEE Transactions on* 44.1 (1996), pp. 110–117 (cit. on p. 385).

[11] A. Bermúdez et al. "An optimal Perfectly Matched Layer with unbounded absorbing function for time-harmonic acoustic scattering problems". In: *Journal of Computational Physics* 223.2 (2007). bermudezPML, pp. 469–488 (cit. on pp. 385, 386).

[12] Matthew R. W. Brake. *A Reduced Iwan Model that Includes Pinning for Bolted Joint Mechanics*. Tech. rep. SAND2016-0207C. Sandia National Laboratories, 2016 (cit. on p. 335).

[13] Gregory Bunting. *Strong and Weak Scaling of the Sierra/SD Eigenvector Problem to a Billion Degrees of Freedom*. Tech. rep. SAND2019-1217. Sandia National Laboratories, 2019 (cit. on p. 35).

[14] Gregory Bunting et al. "Parallel Ellipsoidal Perfectly Matched Layers for Acoustic Helmholtz Problems on Exterior Domains". In: *Journal of Computational Acoustics* (2018) (cit. on p. 385).

[15] M. Christon. "The influence of the mass matrix on the dispersive nature of the semi-discrete, second-order wave equation". In: *Computer Methods in Applied Mechanics and Engineering* 173.1 (1999), pp. 147–166 (cit. on p. 91).

[16] J. Chung and G. M. Hulbert. "A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation - The Generalized Alpha Method". In: *Journal of Applied Mechanics* 60.2 (1993). genalpha, pp. 371–375 (cit. on pp. 235, 238).

[17] R. D. Cook and M. E. Plesha D. S. Malkus. *Concepts and Applications of Finite Element Analysis*. 3rd. John Wiley & Sons, 1989 (cit. on pp. 238, 319).

[18] J. M. Dickens, J. M. Nagawa, and M. J. Wittbrodt. "A critique of mode acceleration and modal truncation augmentation methods for modal response analysis". In: *Computers and Structures* 62.6 (1997), pp. 985–998 (cit. on p. 187).

[19] Clark R. Dohrmann and Olof B. Widlund. "Hybrid domain decomposition algorithms for compressible and almost incompressible elasticity". In: *International Journal for Numerical Methods in Engineering* 82 (2010), pp. 157–183 (cit. on p. 90).

[20] U. S. FAA. *MIL-HDBK-5J. Metallic Materials and Elements for Aerospace Vehicle Structures*. Tech. rep. Department of Defence, 2003 (cit. on pp. 262–265).

[21] Charbel Farhat, Crivelli, and M. Géradin. "Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory". In: *International Journal for Numerical Methods in Engineering* 41 (1998), pp. 675–696 (cit. on p. 238).

[22] C. A. Felippa. *The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation*. Tech. rep. CU-CAS-02-03. Univ. Colo. at Boulder, 2002 (cit. on p. 307).

[23] J. D. Ferry. "Viscoelastic Properties of Polymers". In: Wiley, 1980. Chap. 1-19, pp. 1–590 (cit. on p. 257).

[24] F. Fuentes et al. "Orientation embedded high order shape functions for the exact sequence elements of all shapes". In: *Computers and Mathematics with Applications* 70.1 (2015), pp. 353–458 (cit. on p. 22).

[25] Gregory D. Sjaardema. *A Collection of Exodus Utilities: Exodiff, Epu, Ejoin, and Conjoin*. Tech. rep. SAND2011-5715. epu. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2011 (cit. on pp. 37, 38).

[26] M. F. Hamilton and D. T. Blackstock. *Nonlinear Acoustics*. Academic Press, 1998 (cit. on p. 256).

[27] Yuli Huang, Richard Sturt, and Michael Willford. “A damping model for nonlinear dynamic analysis providing uniform damping over a frequency range”. In: *Computers and Structures* 212 (2019), pp. 101–109 (cit. on p. 288).

[28] Thomas J. R. Hughes. *The Finite Element Method–Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Inc, 1987 (cit. on pp. 222, 238).

[29] G. M. Hulbert and T. J. R. Hughes. “An error analysis of truncated starting conditions in step-by-step time integration: Consequences for structural dynamics”. In: *Earthquake Engineering & Structural Dynamics* 15.7 (1987), pp. 901–910 (cit. on pp. 71, 240).

[30] A. Ibrahimbegovic and E. L. Wilson. “A Modified Method of Incompatible Modes”. In: *Communications in Applied Numerical Methods* 7 (1991). wilson2, pp. 187–194 (cit. on p. 291).

[31] Conor D. Johnson, David A. Kienholz, and Lynn C. Rogers. “Finite element prediction of damping in beams with constrained viscoelastic layers”. In: *AIAA Journal* 20.9 (1982), pp. 1284–1290 (cit. on p. 161).

[32] Kinsler et al. *Fundamentals of Acoustics*. John Wiley & Sons, 1982 (cit. on pp. 218, 296).

[33] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users’ Guide*. Philadelphia, PA, USA: SIAM, 1998 (cit. on pp. 189, 190).

[34] B.T. Lester et al. *Library of Advanced Materials for Engineering (LAMÉ)*. Tech. rep. SAND2021-16079. Sandia National Laboratories, 2021 (cit. on p. 226).

[35] R. H. MacNeal. *Finite Elements: Their Design and Performance*. Marcel Dekker, 1994 (cit. on p. 291).

[36] Ch Michler et al. “Improving the performance of Perfectly Matched Layers by means of hp-adaptivity”. In: *Numerical Methods for Partial Differential Equations* 23.4 (2007), pp. 832–858 (cit. on p. 385).

[37] O. O. Ochoa and J. N. Reddy. *Finite Element Analysis of Composite Laminates*. Kluwer Academic Publishers, 1992 (cit. on p. 298).

[38] J. T. Oden. “Calculation of geometric stiffness matrices for complex structures”. In: *AIAA Journal* 4.8 (1966), pp. 1480–1482 (cit. on p. 317).

[39] Michael A. Puso. “A 3D mortar method for solid mechanics”. In: *International Journal for Numerical Methods in Engineering* 59 (2004), pp. 315–336 (cit. on p. 71).

[40] J. N. Reddy. *An Introduction to the Finite Element Method*. 1st ed. McGraw-Hill, 1984 (cit. on p. 298).

[41] Garth Reese, Rich Field, and Daniel J. Segalman. “A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments”. In: *Shock and Vibration. Digest* 32.6 (2000) (cit. on p. 206).

[42] Brett A. Robertson et al. *Dispersion Analysis of Acoustic Elements in Sierra/SD*. Tech. rep. SAND2014-3870P. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2006 (cit. on p. 91).

[43] S D Team. *Sierra SD Design Manual*. Tech. rep. SAND2021-4312. Sandia National Laboratories, 2021 (cit. on p. 485).

[44] S D Team. *Sierra SD Example Problems Manual*. Tech. rep. SAND2024-02113. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories (cit. on pp. 103, 106, 169, 194, 231, 354, 356, 363, 417, 485).

[45] S D Team. *Sierra Structural Dynamics - Theory Manual*. Tech. rep. SAND2024-01791. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratory, 2024 (cit. on pp. 92, 104, 110, 216, 250, 285, 286, 307, 311, 358, 407, 409, 493).

[46] S D Team. *Sierra Structural Dynamics Verification*. Tech. rep. SAND2023-03618. Sandia National Laboratories, 2023 (cit. on pp. 218, 242, 307, 310, 359, 362, 485).

[47] S M Team. *Sierra Solid Mechanics 4.56 User's Guide*. Tech. rep. SAND2020-5362. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2020 (cit. on pp. 255, 348, 498, 501, 510).

[48] Larry A. Schoof and Victor R. Yarberry. *EXODUS II: A Finite Element Data Model*. Tech. rep. SAND92-2137. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 1994 (cit. on p. 53).

[49] Daniel J. Segalman. "A Four-Parameter Iwan Model for Lap-Type Joints". In: *Journal of Applied Mechanics* 72 (Sept. 2005). SAND2002-3828, pp. 752–760 (cit. on p. 329).

[50] Daniel J. Segalman. *An Initial Overview of Iwan Modeling for Mechanical Joints*. Tech. rep. SAND2001-0811. joint2002. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2001 (cit. on p. 331).

[51] Daniel J. Segalman and Michael J. Starr. *Relationships Among Certain Joint Constitutuve Models*. Tech. rep. SAND2004-4321. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2004 (cit. on p. 331).

[52] Daniel J. Segalman et al. "An Efficient Method for Calculating RMS Von Mises Stress in a Random Vibration Environment". In: *Journal of Sound and Vibration* 230.2 (2000). jsv-2000-v230, pp. 393–410 (cit. on pp. 206, 462).

[53] Daniel J. Segalman et al. "Estimating the Probability Distribution of von Mises Stress for Structures Undergoing Random Excitation". In: *Transactions of the ASME* 122 (Jan. 2000). ProbStress2000 (cit. on p. 208).

[54] Daniel J. Segalman et al. *Handbook on Dynamics of Jointed Structures*. Tech. rep. SAND2009-4164. segalmanhandbook. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2009 (cit. on p. 331).

[55] R. P. Shaw and A. M. Agelastos. *Guide to Using Sierra*. Accessed: 2022-05-30. URL: `https://compsim.sandia.gov/compsim/Docs/Sierra/5.6/GeneralRelease/index.html` (cit. on p. 35).

[56] Sierra Toolkit Development Team. *Sierra Toolkit Manual Version 5.1.4*. Tech. rep. SAND2018-2856. PO Box 5800, Albuquerque, NM 87185-5800: Sandia National Laboratories, 2018 (cit. on p. 131).

[57] Gregory D. Sjaardema. *APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses*. Tech. rep. SAND92-2291. Sandia National Laboratories, 1992 (cit. on pp. 34, 52, 53).

[58] Gregory D. Sjaardema. *GROPE: A GENESIS/EXODUS Database Examination Program*. Tech. rep. SAND92-2289. Sandia National Laboratories, 1992 (cit. on p. 38).

[59] Gregory D. Sjaardema. *SEACAS Assemblies Wiki*. URL: https://github.com/gsjaardema/seacas/wiki/Assemblies (cit. on p. 56).

[60] David O. Smallwood. "Improved Recursive Formula for Calculating Shock Response Spectra". In: *Shock and Vibration Bulletin* 51.2 (1980), pp. 211–217 (cit. on p. 235).

[61] David O. Smallwood. "The Shock Response Spectrum at Low Frequencies". In: *Shock and Vibration Bulletin* 56.1 (1986). ramp invariant filter, pp. 279–288 (cit. on p. 235).

[62] C. J. Stimpson et al. *Verdict Library Reference Manual*. Tech. rep. SAND2007-2853P. Sandia National Laboratories, 2007 (cit. on pp. 293, 478).

[63] G. Strang and G. Fix. *An Analysis of the Finite Element Method*. Wellesley-Cambridge Press, 2008. ISBN: 9780980232707. URL: https://books.google.com/books?id=K5MAOwAACAAJ (cit. on p. 42).

[64] R. L. Taylor, P. J. Beresford, and E. L. Wilson. "A Nonconforming Element for Stress Analysis". In: *International Journal for Numerical Methods in Engineering* 10 (6 1976). wilson1, pp. 1211–1219 (cit. on p. 291).

[65] T. F. Walsh et al. "Finite element methods for structural acoustics on mismatched meshes". In: *Journal of Computational Acoustics* 17.3 (2009), pp. 247–275 (cit. on p. 493).

[66] Timothy F. Walsh, Garth M. Reese, and Ulrich L. Hetmaniuk. "Explicit A Posteriori Error Estimates for Eigenvalue Analysis of Heterogeneous Elastic Structures". In: *Computer Methods in Applied Mechanics and Engineering* 196.37 (2007), pp. 3614–3623 (cit. on p. 471).

[67] Malcolm L Williams, Robert F Landel, and John D Ferry. "The temperature dependence of relaxation mechanisms in amorphous polymers and other glass-forming liquids". In: *Journal of the American Chemical society* 77.14 (1955), pp. 3701–3707 (cit. on p. 257).

[68] Paul H. Wirsching and Mark C. Light. "Fatigue under wide band random stresses". In: *Journal of the Structural Division, ASCE* 106.7 (1980), pp. 1593–1607 (cit. on p. 205).

[69] Paul H. Wirsching, Thomas L. Paez, and Keith Ortiz. *Random Vibrations: Theory and Practice*. Courier Corporation, 2006 (cit. on pp. 205, 262).

This page intentionally left blank.

INDEX

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	Technical Library	1911	0845

This page intentionally left blank.

Sandia
National
Laboratories