SANDIA REPORT
SAND2021-11328
Printed September 15, 2021

Sandia National Laboratories

Sierra/SD – How To Manual – 5.2

Sierra Structural Dynamics Development Team

Latest Software Release:
5.2-Release 2021-09-14

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

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

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

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

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

The How To Manual supplements the User's Manual and the Theory Manual. The goal of the How To Manual is to reduce learning time for complex end to end analyses. These documents are intended to be used together. See the User's Manual for a complete list of the options for a solution case. All the examples are part of the **Sierra/SD** test suite. Each runs as is.

The organization is similar to the other documents: How to run, Commands, Solution cases, Materials, Elements, Boundary conditions, and then Contact. The table of contents and index are indispensible.

The Geometric Rigid Body Modes section is shared with the Users Manual.

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 and support of code capabilities. This team includes Gregory Bunting, Mark Chen, Nathan Crane, David Day, Clark Dohrmann, Sean Hardesty, Dagny Joffe, Payton Lindsay, Rob Flicek, Lynn Munday, and Brian Stevens.

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, Brett Clark, Murthy Guddati, Chandler Smith, Benjamin Treweek, and Timothy Walsh.

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 regular provide code capabilities as well as help review and verify code capabilities and documentation. Individuals directly contributing to the **Sierra/SD** documentation or code base during the last year include Samuel Browne, Victor Brunini, Pete Coffin, Jonathan Clausen, Gabriel de Frias, Mike Glass, Jacob Koester, Mark Merewether, Scott Miller, Kendall Pierson, Vincent Pericoli, Julia Plews, Timothy Smith, and Alan Williams.

Historically dozens of other Sandia staff, students, and external collaborators have also contributed to the **Sierra/SD** product 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[1] 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.

The analyst community has begun their own website to address means of working various solutions in **Sierra/SD**. See the SNL-wiki. The documentation is dynamic, but has details on eigenvalue extraction, modal based solutions, and reading sideset data.

1. Training Problems

The sections of a **Sierra/SD** input file are described in the Sierra SD Users' Guide. An input file has seven required sections: solution, file (**Exodus** mesh), load(s), outputs, echo, block (one per element block in the input **Exodus** file) and material (one per unique material). In the file section, the string FILEPATH must be replaced by the name of the input **Exodus** mesh file.

The input file for the statics solution method, 1, provided in the Appendix has the required sections, and three optional sections: parameters, boundary and GDSW. The parameter Wtmass, typically $1/(32.2ft/s^2\ 12in/ft)$, is used so that for example densities may be specified in units of lbs/in^3, as described in the Users' Guide. Boundary conditions on a side set, or in this case a node set, are specified in the boundary section. The GDSW section indicates that the threshold on the relative residual norm be decreased from the default 1.e-6 if using the GDSW linear solver.

The input file 2 for the eigen solution method requests that the twelve lowest frequency modes be computed. The eigen norm parameter indicates that the mode shapes will be normalized in a way that is convenient for visualization. The default normalization uses the mass matrix. Here `solver_tol` has been further reduced to $1.e-10$.

The transient simulation input file 3 uses the default Newmark method and has the total simulation time of 1/100 seconds. The load specified by a tabulated Haversine pulse. The history section indicates that the output quantities at each time step and at the specified node sets only will be written to a different Exodus output file with the suffix h. In this case the history file name is `fixture-out.h`. The history file is $20,000$ times smaller than the ordinary output file. Finally the restart option in the solution section means that the file fixture-out.rslt_trans will be written. It is possible to restart the simulation using this restart file, as described in the Users' Guide.

In a modal transient simulation, the transient problem is projected onto the subspace spanned by the mode shapes of a user specified number of the lowest frequency modes. Modal transient simulations are often convenient when a system has to be modeling using many different loads. The `transient` keyword has been replaced by the `modaltransient` keyword. Also, a single input file is used for both the initial eigenvalue problem (20 modes), and the following modal transient solution. This is called a *multicase* solution. Another difference is that the plural `loads` section has been replaced by a a numbered `load` block; this is always the case with a multicase simulation.

Returning to the first solution case in the modal transient simulation, the eigenvalue problem, a shift is set to $-1e+6$. The default shift is -1. Here the first eigenvalue is $1e+8$. The eigenvalue problem is solved more efficiently and accurately if the shift is approximately -1 times the lowest nonzero eigenvalue (flexible mode).

The frequency response function is used for example to confirm engineering assumptions about the frequency content of the accelerations. The `modalfrf` solution case showing in

the input file 5 concerns the frequency response function

$$\hat{u}(\omega) = (K + i\omega C - \omega^2 M)^{-1} \hat{f}(\omega), i = \sqrt{-1}.$$

Modal frequency response refers to using the mode shapes to diagonalize the transfer function. A linear solver is not used to evaluate the transfer function, but is used in solving the eigenvalue problem. The function here describes the frequency dependent load, the Fourier transform of the temporal load. The `damping` section supplies the coefficient for mass proportional damping, $C = \gamma M$. The frequency block sets the spatial location and frequency range of the load.

In the modal frequency response problem note that there is both a history section and a frequency section. The input file is for a multicase simulation. The history file section applies to the solution of the eigenvalue problem, and is ignored during the solution of the frequency response problem. And the frequency response section is ignored during the solution of the eigenvalue problem, and applies only to the frequency response problem.

The last input file 6 will be discussed in the next section.

2. 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** doesn't 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".

2.1. Debugging Threading Approaches

Choosing an ideal set of parameters for a threaded run can be complex. There are many options to choose from, and availability can vary. As such, it is frequently useful to obtain information about exactly what your chosen set of parameters is doing on a given system.

There are several stand-alone codes that will accomplish this goal, but in **Sierra/SD** we have incorporated a summary table that includes information about MPI ranks, threads, the physical core on which each thread is running, and the core affinity of each thread. This table will be output in a typical run if the "timing_summary" or "threading_summary" options are requested in the echo block. Alternatively, you can output this table directly with the --threading command line option, i.e.

```
mpirun -n 4 salinas -nt 2 --threading
```

This option allows you to check your run command and the effect of any environment variables without invoking a full **Sierra/SD** run. Additionally, we will always issue a warning if we detect any over-subscribed cores.

2.2. Batch Submission / Optimal Parameters for KNL

If you are not familiar with using a queued system, proceed with caution. A project can easily go behind schedule. Due diligence (as described in the Users Manual) is necessary to minimize the possibility of an input file error. Also, there may be more than one queue. It is also important to do as much work as possible in the fastest queue.

Batch submission scripts for threaded runs must be tailored to the system you're running on. In the following example, we will focus on the Knights Landing (KNL) processor, as found on the *Mutrino-KNL* platform.

There are 68 CPUs on a KNL node. Each has 4 hyperthreads. They are numbered 0 - 271.

- Cores 0 - 67 are the first hypertask on CPUs 0 - 67
- Cores 68 - 135 are the second hypertask on CPUs 0 - 67
- Cores 136 - 203 are the third hypertask on CPUs 0 - 67
- Cores 204 - 271 are the fourth hypertask on CPUs 0 - 67

The following will like the "Sierra" script set up a run on Mutrino/KNL including the number of nodes needed.

```
#!/bin/bash
module load sierra/release.knl
export PATH=path_to_salinas:$PATH

#sbatch settings
accountNumber="your_WC_ID"
time="04:00:00"

# input/decomp settings
```

```
inputFile="myExampleProblem.inp"
numRanks=51             # number of MPI procs/ranks
numThreadsPerRank=4     # number of threads per proc/rank

# machine-specific information... obtain using lscpu
numSocketsPerNode=1     # number of sockets per node
                        # for KNL, this is 1
numCoresPerSocket=64    # number of cores per socket
                        # for KNL, this is 68, but we don't want to use them all
                        # -> say 64
numThreadsPerCore=4     # number of CPUs/threads per core (hyperthreads)
                        # for KNL, this is 4

######################## USER INPUT SECTION FINISHED ######################

# Determine the number of sockets/nodes needed for the procs*threads requested
# NOT USING HYPERTHREADS
maxNumRanksPerSocketNeeded=$[numCoresPerSocket/numThreadsPerRank];
minNumThreadsPerRankNeeded=\
$[numCoresPerSocket*numThreadsPerCore/maxNumRanksPerSocketNeeded];

minNumSocketsNeeded=$[numRanks/maxNumRanksPerSocketNeeded]
remainder=$[numRanks%maxNumRanksPerSocketNeeded]
if [ $remainder -gt 0 ]; then
    minNumSocketsNeeded=$[minNumSocketsNeeded+1]
fi

minNumNodesNeeded=$[minNumSocketsNeeded/numSocketsPerNode]
remainder=$[minNumSocketsNeeded%numSocketsPerNode]
if [ $remainder -gt 0 ]; then
    minNumNodesNeeded=$[minNumNodesNeeded+1]
fi

echo
echo "Machine info..."
echo -n "# sockets per node = ${numSocketsPerNode}, "
echo -n "# cores per socket = ${numCoresPerSocket}, "
echo "# threads per core = ${numThreadsPerCore}"
echo
echo -n "Requested ${numThreadsPerRank} threads "
echo "and ${numRanks} mpi ranks"
echo
echo -n "Allocated ${minNumThreadsPerRankNeeded} threads per rank, "
echo -n "${maxNumRanksPerSocketNeeded} ranks per socket, "
echo -n "${minNumSocketsNeeded} sockets, "
```

```
echo "and ${minNumNodesNeeded} nodes"
echo


echo "Generating sbatch submission script \"sbatchScript.sh\""
echo

echo "#!/bin/bash" > sbatchScript.sh
echo "#SBATCH --account=${accountNumber}" >> sbatchScript.sh
echo "#SBATCH --nodes=${minNumNodesNeeded}" >> sbatchScript.sh
echo "#SBATCH --time=${time}" >> sbatchScript.sh


# MUTRINO... use these to get to the KNL partition
echo "#SBATCH -p knl" >> sbatchScript.sh
echo "#SBATCH -C knl,compute,quad,cache" >> sbatchScript.sh

echo "export OMP_NUM_THREADS=${numThreadsPerRank}" >> sbatchScript.sh
echo "export OMP_PROC_BIND=true" >> sbatchScript.sh

echo "srun --cpu-bind=threads --cpus-per-task=${minNumThreadsPerRankNeeded} \
-n ${numRanks} \
salinas -nt ${numThreadsPerRank} -i ${inputFile}" >> sbatchScript.sh

sbatch sbatchScript.sh
```

The command "source scriptName" runs the script. The script was designed to be adaptable to systems other than Mutrino/KNL. In the following section it is modified to run on HPC Clusters. Users are strongly advised to verify that their approach behaves as expected. After writing a script, but before running it, reduce the duration to say one minute and replace "`-i ${inputFile}`" with "`-threading`". It should run quickly. And it shows where threads are allocated. Also, run the script provided here without the actual sbatch command. It should generate "sbatchScript.sh".

2.3. Parameters for Running on HPC Clusters

Every HPC platform will have different chip architectures, necessitating changes to the environment variables used for launching applications that use both mpi and threading. In this section, we provide the required modifications to the above Mutrino script needed for optimally running threaded applications on an Institutional Cluster. These script modifications include different environment variables that account for the differences in chip architectures as well as changes to account for differences in the executables needed for launching MPI jobs. **Sierra/SD** is traditionally run as a MPI only process on an Institutional Cluster. However, in some instances cores on each compute node are left idle,

and it therefore makes sense to put those idle cores to use by using threads. This is often the case for memory bound problems, in which case the use of threads is an attractive solution as it does not lead to an increase the in memory. As such, we have modified the Mutrino/KNL script from the previous section with settings that we have found to work for the cluster, and which may be useful when setting up a run on other machines using threading.

```
#!/bin/bash
module load sierra-devel/intel-17.0.1-openmpi-1.10
export PATH=path_to_openmp=on_salinas_build:$PATH

#sbatch settings
accountNumber="your_WC_ID"
time="04:00:00"

# input/decomp settings
inputFile="myExampleProblem.inp"
numRanks=51             # number of MPI procs/ranks
numThreadsPerRank=4     # number of threads per proc/rank

# machine-specific information... obtain using lscpu
numSocketsPerNode=2     # number of sockets per node
                        # for chama, this is 2
numCoresPerSocket=8     # number of cores per socket
                        # for chama, this is 8
numThreadsPerCore=1     # number of CPUs/threads per core (hyperthreads)
                        # for chama, this is 1

######################## USER INPUT SECTION FINISHED ######################

# Determine the number of sockets/nodes needed for the procs*threads requested
# NOT USING HYPERTHREADS
maxNumRanksPerSocketNeeded=$[numCoresPerSocket/numThreadsPerRank];
minNumThreadsPerRankNeeded=\
$[numCoresPerSocket*numThreadsPerCore/maxNumRanksPerSocketNeeded];

minNumSocketsNeeded=$[numRanks/maxNumRanksPerSocketNeeded]
remainder=$[numRanks%maxNumRanksPerSocketNeeded]
if [ $remainder -gt 0 ]; then
    minNumSocketsNeeded=$[minNumSocketsNeeded+1]
fi

minNumNodesNeeded=$[minNumSocketsNeeded/numSocketsPerNode]
remainder=$[minNumSocketsNeeded%numSocketsPerNode]
if [ $remainder -gt 0 ]; then
```

```
    minNumNodesNeeded=$[minNumNodesNeeded+1]
fi

echo
echo "Machine info..."
echo -n "# sockets per node = ${numSocketsPerNode}, "
echo -n "# cores per socket = ${numCoresPerSocket}, "
echo "# threads per core = ${numThreadsPerCore}"
echo
echo -n "Requested ${numThreadsPerRank} threads "
echo "and ${numRanks} mpi ranks"
echo
echo -n "Allocated ${minNumThreadsPerRankNeeded} threads per rank, "
echo -n "${maxNumRanksPerSocketNeeded} ranks per socket, "
echo -n "${minNumSocketsNeeded} sockets, "
echo "and ${minNumNodesNeeded} nodes"
echo


echo "Generating sbatch submission script \"sbatchFile\""
echo

echo "#!/bin/bash" > sbatchFile
echo "#SBATCH --account=${accountNumber}" >> sbatchFile
echo "#SBATCH --nodes=${minNumNodesNeeded}" >> sbatchFile
echo "#SBATCH --time=${time}" >> sbatchFile


# use this to run in the much faster "short" queue (16 nodes, 4 hours max)
echo "#SBATCH -p short,batch" >> sbatchFile

echo "export OMP_NUM_THREADS=${numThreadsPerRank}" >> sbatchFile
echo "export OMP_PROC_BIND=true" >> sbatchFile

echo "mpiexec --map-by socket:pe=${minNumThreadsPerRankNeeded} \
-n ${numRanks} \
salinas -nt ${numThreadsPerRank} -i ${inputFile}" >> sbatchScript.sh

sbatch sbatchFile
```

Note that this script will only work with a maximum number of threads equal to the number of cores per socket. Note also that there is currently threads are disabled. To use threads a special version of **Sierra/SD** needs to be build with *openmp=on*.

3. Coupled Sierra/SM- Sierra/SD Analysis

In this section, we describe how to run a modal analysis using **Sierra/SM**, also known as Adagio. This is a multi-step approach that uses **Sierra/SM** and **Sierra/SD** separately, and couples the data using **receive_sierra_data**. A nonlinear preload is computed in **Sierra/SM**, followed by a modal analysis in **Sierra/SD**. In this approach, the modal analysis is performed about the nonlinear state that is computed in **Sierra/SM**. An updated Lagrangian approach is used - this implies that the nodal coordinates in **Sierra/SD** are computed as the sum of the initial coordinates, plus the final set of displacements computed in **Sierra/SM**. This is the most convenient approach, since the modes about the deformed state are typically of most interest. [1]

Handoff between **Sierra/SM** and **Sierra/SD** uses a handoff of files between separate runs. In this approach, **Sierra/SM** writes the necessary data to the output **Exodus** file, and then **Sierra/SD** reads this data in and executes the modal analysis. This approach has been found to be useful for running large models, since it breaks the overall computation into two phases, and thus acts as a restart.

A file transfer approach that couples **Sierra/SM** and **Sierra/SD** proceeds as follows. We assume that the **Sierra/SM** input file is named 'sierra.i', and the **Sierra/SD** input file is named 'salinas.inp'. The model consists of four layers of material, with a membrane layer between the bottom layers, as shown in Figure 3-1.

1. Construct input decks for both **Sierra/SM** (sierra.i) and **Sierra/SD** (salinas.inp). There are modifications to the standard that are required for both input decks, and these will be described below.
2. Execute **Sierra/SM**
3. The output **Exodus** file that is assigned in **Sierra/SM** is used as the geometry file for the **Sierra/SD** analysis. This step will need to be inserted manually by the user.
4. Execute **Sierra/SD**. Some of the steps of the analysis are indicated in Figure 3-2.
5. The eigenvalues and modal frequencies are listed in the `salinas.rslt` file, and both the modal frequencies and mode shapes are in the file `salinas-two.exo`, assuming that the second case in the **Sierra/SD** input deck has the name `two` (see example below).

[1] Receive_sierra_data has 3 primary use cases.

a. Preload from **Sierra/SM**, where displacements and stresses are passed, **Sierra/SD** reads those in, adjusts the tangent stiffness matrix, and computes modes.

b. Preload from SM, where displacements and stresses are passed, **Sierra/SD** reads those in, adjusts the tangent stiffness matrix and equilibration forces, and then computes a transient response to some load that is specified in SD input.

c. Implicit or Explicit transient analysis in SM, followed by a `handoff` to an implicit transient in **Sierra/SD**. By default, **Sierra/SD** will start at the end time of the SM handoff analysis.

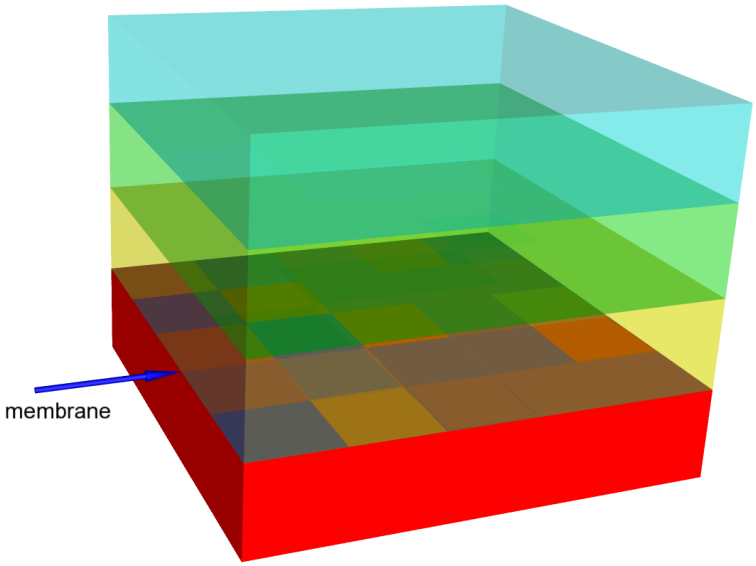

Figure 3-1. – SM/SD Transfer Model Geometry

1. Read displacements, stresses and some material parameters from previous SM analysis. These are found in the **Exodus** output from SM.
2. Move coordinates, $X' = X_o + U$.
3. Compute element stiffness matrices from material properties.
4. Adjust stiffness matrices for stress preload.
5. Generate constraints.
6. Assemble system level matrices and compute eigen problem.

Figure 3-2. – Steps in **Sierra/SD** Coupled Analysis. Most properties and element matrices are recomputed in SD.

The **Sierra/SM** input specifications for how to create this output **Exodus** file is in a later section (3.2). A verification test is available in the **Sierra/SD** verification section in the test repository, where a prescribed displacement is applied in the X-direction in **Sierra/SM**, and then a modal analysis is performed in **Sierra/SD**.

For tire models, the `thickness scale factor` variable in the **Sierra/SM** input deck is required for a coupled **Sierra/SM-Sierra/SD** analysis. This parameter is the same as the fiber diameter, which is specified in the material section for the membranes. This parameter is needed for the coupled analysis, because it is used to determine the modulus of the fibers, which are in turn used in **Sierra/SD** to construct the stiffness matrix.

3.1. Sierra/SD *parameters for file transfer with* Sierra/SM

Next, we consider a typical **Sierra/SD** text input in Figure 3-3. There are several sections of interest in this input deck. The syntax involved in a **Sierra/SD** input is described in *User's Manual*. Here, we assemble an example that collects the inputs in different sections, and highlight those areas that are different due to the SM coupling.

The `solution` section contains two cases. Case `one` uses the `receive_sierra_data` solution procedure to receive the necessary data from **Sierra/SM**. This is where all of the data transfer takes place, including data such as stresses, displacements, and analysis time. The second case instructs **Sierra/SD** to perform a modal (or eigen) analysis about the nonlinear state that was received from **Sierra/SM**. In this case, 10 modes are requested. Also, the GDSW solver is requested in this case. This solver is usually needed in problems with large numbers of constraint equations that are generated with tied data pairs.

The `boundary` section specifies nodesets and sidesets that are to be fixed. The four edges of the plate, which are denoted by sidesets 21, 22, 11, and 20, are fixed. These boundary conditions diff from those used in the **Sierra/SM** analysis. The boundary conditions in the **Sierra/SD** modal analysis may be the same or different than those used in the **Sierra/SM** analysis, depending on the goals of the analysis. Only one Boundary section is applied in the **Sierra/SD** input, and it applies to the entire SD analysis.

Since there are no loads associated with a modal analysis, the `loads` section is empty.

In this example, Block 11 is a block of membrane elements. These elements are currently setup as layered elements in **Sierra/SD**, even though there is only one layer. Thus, the `layer 1` specification is needed. Following this, the material name, layer thickness and fiber orientation are needed. In this case, the latter two come in as data from the Sierra transfer, and `from_transfer` is used in lieu of a numerical value. If this were an uncoupled analysis, numeric values would be used instead. The remaining four blocks are hex elements, and thus the only required input syntax is the specification of the material identifier.

For the 'orthotropic_layer' material many element attributes may be specified as `from_transfer`. Table 3-1 summarizes some of the quantities that may be exchanged in the `read_sierra_data` process using the `from_transfer` option for the 'orthotropic_layer' material.

```
SOLUTION
 case one
   receive_sierra_data
     scale=no
     lumped
 case two
   eigen
   nmodes 10
   solver=gdsw
END

FILE
    geometry_file    'mesh.g'
    NUMRAID 1
END

Boundary
    sideset 21
      fixed
    sideset 22
      fixed
    sideset 11
      fixed
    sideset 20
      fixed
end

LOADS
END

OUTPUTS
    disp
END

ECHO
  INPUT
END

BLOCK 11
  QuadM
    layer 1
      material ply
      thickness = from_transfer
      fiber orientation = from_transfer
END

BLOCK 1
        material 2
END

BLOCK 2
        material 2
END

BLOCK 6
        material 2
END

BLOCK 5
        material 2
END

MATERIAL 2
  name "steel"
  E 30.0e6
  nu 0.0
  density 2.61e-4
END

MATERIAL ply
  name "ply"
  density  = from_transfer
  orthotropic_layer
  E1 = from_transfer
  E2 = from_transfer
  nu12 = from_transfer
  G12 = from_transfer
  percent_continuum 0.005
END

Tied Data
  surface 2 1
  search tolerance 0.2
  edge tolerance 1.0e-8
  method = inconsistent
END
```

Figure 3-3. – Salinas input deck for coupled Adagio-Salinas analysis

Quantity	Descriptor
element attribute	Nearly any element attribute that can be stored in the **Exodus** file may also be exchanged using `from_transfer`.
fiber orientation	2D fiber orientation in a layer
thickness	2D shell layer thickness
E1	material modulus
E2	material modulus
nu12	material modulus
G12	material modulus
density	material density

Table 3-1. – Orthotropic_layer Quantities Available for FROM_TRANSFER Exchange in Read_Sierra_Data

Next, the two material blocks are specified. Material 2 is specified to be an isotropic elastic material, with Young's modulus, Poisson ratio, and material density. These values can be different than those used in the **Sierra/SM** analysis, and in fact they typically are.

Material 'ply' corresponds to the membranes. Here, there are several parameters that are required to fully specify the orthotropic material properties, and density. However, all of these properties are transferred from **Sierra/SM**, and thus the `from_transfer` option is used in lieu of numerical values.

Next, we consider the Tied Data block. As mentioned earlier, most of the parameters in this section are described more fully in *User's Manual*. Here, we only focus on the special considerations needed for coupled **Sierra/SM**-**Sierra/SD** analysis. In particular, the search tolerance parameter is used by **Sierra/SD** to determine which nodes are in contact. We recommend using a very small search tolerance in **Sierra/SD**, since at the end of the **Sierra/SM** analysis the surfaces will already be in close contact.

Finally, although the previous example did not involve pressure forces, we note that when pressure loads are used in the **Sierra/SM** preload analysis, a follower stiffness term needs to be activated in **Sierra/SD**. For example, if a pressure of 0.2 was applied to sideset 1 in the **Sierra/SM** input deck, then the following block would be needed in the **Sierra/SD** input deck

```
LOADS
   sideset 1
      pressure = 0.2
      follower = y
END
```

Although no error will be reported if this block is left out, the results may be inaccurate.

3.2. Sierra/SM *output parameters for* Sierra/SD *modal analysis using file transfer*

When running **Sierra/SM** a list of quantities needs to be written to file for the subsequent **Sierra/SD** modal analysis. The *names* of the variables must be set properly in the **Sierra/SM** output or **Sierra/SD** will not read the data.

For a typical analysis, the only parameters needed are the stress and displacement.

```
begin Results Output output_adagio
Database Name = stage1.e
Database Type = Exodus
At Step 0, Increment = 1
nodal Variables = displacement as displ
element Variables = rotated_stress as stress
end results output output_adagio
```

The following parameters need to be added to the **Sierra/SM** output section for a tire model.

```
begin Results Output output_adagio
Database Name = stage1.e
Database Type = Exodus
At Step 0, Increment = 1
component separator character = ""
nodal Variables = displacement as displ
element variables = cord_modulus as fibmod
element variables = memb_stress as memstr
element Variables = rotated_stress as stress
element Variables = density as fiberdensity
element Variables = element_thickness as fiberthickness
element variables = cord_ax as ax
element variables = cord_ay as ay
global Variables = timestep as timestep
end results output output_adagio
```

This will then create the stage1.e file, which will contain all of the data necessary for **Sierra/SD**.

The next step is to decompose this file into the number of required partitions for the modal analysis. For example, if the modal analysis expects to use 10 processors, then one could use the commands

```
unix> mpirun -np 10 stk_balance stage1.e split
```

These commands will create the 10 partitioned files. Then, the **Sierra/SD** file section should be modified as follows,

```
FILE
   geometry_file './split/stage1.par'
END
```

Sierra/SD can then be executed using the command

```
unix> mpirun -np 10 ./salinas salinas.inp
```

After the **Sierra/SD** analysis is complete, it is typically desirable to join the parallel results files into a single file. This is typically done with the tool `epu`. Although some of joining tools allow the user to specify a base **Exodus** file, for these coupled nonlinear-modal analyses, the incoming **Exodus** file for **Sierra/SD** has both initial coordinates and displacements. The current configuration for the modal analysis is determined by adding the original coordinates and the displacements to obtain the "deformed" configuration. Thus, for the joining process, it is necessary to use a base **Exodus** file that has the displacements added to the original model coordinates. Otherwise, the final joined file will not have the correct updated coordinates. The safest and easiest approach is to not specify a base **Exodus** file in the joining process. With `epu` the base file is optional.

3.2.1. Syntax differences and design tips

Key syntax differences and design tips between **Sierra/SM** and Sierra SD

1. The search tolerance for the tied contacts in **Sierra/SD** must be set carefully in order to ensure that nodes that are in contact in **Sierra/SM** are in contact in Sierra SD and vice versa.
   a) Use very small search tolerance, in the range of one to two orders of magnitude smaller than the capture tolerance in **Sierra/SM** should be sufficient.
   b) Ideally, nodal contact information should be passed directly from **Sierra/SM** to **Sierra/SD** (not currently available).
2. The sidesets used to define the tied contacts in **Sierra/SD** must be defined in the input **Exodus** file used by **Sierra/SM**, even if they are not used in **Sierra/SM**.
3. The material properties for each element are not passed from **Sierra/SM** to **Sierra/SD**. This is important with nonlinear models.

3.2.2. Modifications for modal analysis following SST analysis

When **Sierra/SM** is used for an SST analysis, the same procedure for computing modes in **Sierra/SD** can be followed, with minor modifications.

For each block, a `rotational_type` has to be specified to indicate the rotational state of that block. For a rotating body, this quantity would either be set to `Eulerian` or `none`, indicating that the block is rotating in an Eulerian framework or not at all. For the blocks

on the rotating body, use `Eulerian`, whereas for the stationary block, use `none`. An example is given below.

```
BLOCK 1
   material 1
   hex8u
   sd_factor=1
   rotational_type eulerian
END
```

The rotational speed of the rotating body is specified in the `loads` section. This speed is specified as a vector with three components, each one giving the rotational speed about the corresponding global coordinate axis. An example is given below

```
LOADS
   body
   rotation = 0 2.95 0
END
```

For SST analysis, a torque is typically specified in Eagle, and a corresponding rotational velocity is computed. Since this speed is not known beforehand, it must be output from the Eagle run as a global variable. Then, that variable can be input into **Sierra/SD**. The following output section will write the steady-state rotational velocity to a global variable called `sstrvel`.

```
begin results output output_1
database name = model.e
database type = exodusII
component separator character = ""
at step 0 increment = 1
global variables = TOTAL_ITER AS ITOTAL
global variables = sstrvel_1 as sstrvel
end results output output_1
```

Once `sstrvel` is known, it can then be copied to the **Sierra/SD** input deck.

The projection method to approximate the eigenvalues of rolling bodies. The syntax for this in the `solution` section is given below.

```
SOLUTION
    solver=gdsw
    case one
       receive_sierra_data
       lumped
    case two
       qevp
```

```
          method=projection_eigen
          lumped
          nmodes=20
          shift = -100.0
END
```

3.3. Rigid Rims, Coupling with Concentrated Masses, and Superelements

In some cases, it is desirable to treat parts of the mesh as rigid bodies during the modal analysis. This can be accomplished in **Sierra/SD** using the rigid set capability. For example, if sidesets 901 and 902 surround two pieces of the mesh, then the following command block will make the surfaces rigid. Although, in theory, these parts would be free to deform, the resulting modes would be very high frequency and thus out of range of the normal interest.

```
rigidset set1
  sideset 901
  sideset 902
END
```

It is also often effective to add the mass properties of a rigid body onto its centroid. This can be accomplished by coupling to a concentrated mass. For this, a sphere element needs to be added to the mesh file. This can be done with a tool to manipulate the mesh such as Patran (with gjoin) or Cubit. The sphere can be added to the **Sierra/SM** input file, and it will be inactive for the first stage analysis. For the **Sierra/SD** portion, the following blocks would connect the concentrated mass to the rigid body

```
rigidset set1
  sideset 901
  sideset 902
  centernode tied to node 28539 block 20
END
BLOCK 20
  coordinate 1
  Joint2G
  kx=elastic 1.0e+10
  ky=elastic 1.0e+10
  kz=elastic 1.0e+10
  krx=elastic 1.0e+10
  kry=elastic 1.0e+10
  krz=elastic 1.0e+10
END
```

```
BEGIN RECTANGLAR COORDINATE SYSTEM 1
  origin 0 0 0
  z point 0 0 1
  xz point 1 0 1
END
BLOCK 17
  conMass
  mass 1.0e1
  Ixx 1.0e1
  Iyy 1.0e1
  Izz 1.0e1
  Ixy 0.0
  Ixz 0.0
  Iyz 0.0
  offset 0 0 0
END
```

In this example, block 17 is the concentrated mass, and contains both the mass and inertial properties of the rigid body. Thus, the actual rigid body would be given zero density. Block 17 is also node 28539, and is connected to the reference node of the rigidset through block 20 via the statement the statement `"centernode tiedto node 28539 block 20"`. The reference node of the rigidset is chosen to be the node in the rigidset that is closest to its geometric centroid (which is computed by averaging the coordinates of the nodes in the rigidset). Since that node will most likely not be at the same location as the concentrated mass node, block 20 will usually have a non-zero length.

We also note that in the statement `"centernode tiedto node 28539 block 20",` Node 28539 must be connected to a virtual Joint2G block, in this case block 20. That is, block 20 is not part of the mesh file in **Exodus**, but instead is created internally in **Sierra/SD** during execution of the code. It is necessary that block 20 be a virtual Joint2G block, otherwise the code will die with a fatal error message. This element provides 6 components of elastic resistance (3 translations and 3 rotations) between the concentrated mass and the reference node of the rigid body. As these elastic stiffnesses increase, the effect converges to a rigid bar between the pair of nodes.

This same approach can be used to couple to a superelement in the case where the superelement has a single interface node. In that case, the superelement is also represented in the mesh with a sphere element, and the coupling between the superelement and the reference node of the rigid body is specified in exactly the same manner. In this case, however, block 17 is defined to be a superelement rather than a concentrated mass, and is given a corresponding netcdf file that contains the reduced mass and stiffness matrices of the superelement.

```
rigidset set1
   sideset 901
```

```
   sideset 902
   centernode tiedto node 28539 block 20
END
BLOCK 20
   coordinate 1
   Joint2G
   kx=elastic 1.0e+10
   ky=elastic 1.0e+10
   kz=elastic 1.0e+10
   krx=elastic 1.0e+10
   kry=elastic 1.0e+10
   krz=elastic 1.0e+10
END
BEGIN RECTANGULAR COORDINATE SYSTEM 1
   origin 0 0 0
   z point 0 0 1
   xz point 1 0 1
END
BLOCK 17
   superelement
   file='superelem.ncf'
   map
   // local grid id cid
     1     1
     1     2
     1     3
     1     4
     1     5
     1     6
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
END
```

4. Encore Transfers

In this section, we will describe how to transfer velocity data to an acoustic mesh using Encore. The purpose is to transfer velocity data written by Presto to data that can be used by **Sierra/SD** to generate an acoustic analysis . We will present an example of a

script that would accomplish this transfer. In order to execute this script, the following command would be used.

```
sierra encore -i gw_transfer.i -j 1
```

Encore maps the data from a standard Presto output (.e) file to a new (.e) file that is subsequently read in by **Sierra/SD** and used for the acoustic analysis. The new data file is now ready to be called by **Sierra/SD** for acoustic analysis.

In order to understand how the Encore Procedure works, a step-by-step tutorial of the input deck (gw_transfer .i) is provided. The code begins as follows to call the Encore procedure:

```
Begin Sierra Encore
```

4.1. Define Solid Mesh

Before executing Encore, it is necessary to define the various files that will be read and written to. In this case, a `gw.e` file that contains the velocity data required to perform the Encore function is given the identifier `solid_mesh`. Whenever called, this keyword will direct the script to the `gw.e` file. Note that the keyword can be any name the user chooses.

```
Begin Finite Element Model solid_mesh
   Database Name = gw.e
   Component separator character is NONE
End Finite Element Model
```

Similarly, the next block defines a keyword,`acoustic_mesh`, which points Encore to the file containing acoustic mesh data.

```
Begin Finite Element Model acoustic_mesh
   Database Name = deform_acoustic_mesh.exo
End Finite Element Model
```

4.2. Encore Transfer Procedure

The following is the Encore Procedure that uses the Sierra input defined above to complete a mapping from the solid mesh to the acoustic mesh. Note that there are two regions listed in this block that have not been defined yet, but will be defined later in the code.

```
Begin Encore Procedure encore_transfer_proc

   Begin Solution Control Description
     Use System main
     Begin System main
        Begin Transient encore_trans
           Advance solid_region
           Transfer solid_to_acoustic
                       Event Reinitialize solid_to_acoustic
           Advance acoustic_region
        End
```

There are four basic steps that make up the Encore procedure. They are:

1. `Advance solid_region`

   At the start of the loop, the solid mesh region will advance one time step.

2. `Transfer solid_to_acoustic`

   A transfer will take place from the solid to the acoustic mesh at the time step defined in the previous step. This process will match nodes from the solid region with nearby nodes in the acoustic region. Keep in mind that the tire is rolling while the acoustic mesh is stationary, and therefore the nodes on the boundary of the tire that contain velocity data are constantly changing with respect to the fixed acoustic mesh.

3. `Event Reinitialize solid_to_acoustic`

   Due to the movement of the nodes in the solid region, the paired nodes will separate beyond a specified tolerance. This causes a need for re-initialization for the next time step. The data obtained just before the re-initialization will map to a time step in the acoustic region, and will create the desired acoustic data.

4. `Advance acoustic_region`

   The acoustic region will then advance one time step and the loop will repeat.

4.3. Simulation Time

The next block determines how much of the Presto mesh will be transferred and then ends the transfer procedure. Considering that the Presto mesh has data corresponding to time 0 through time x, the Simulation Termination Time is set to some arbitrary time much larger than x to ensure that all the data is transferred. If only a portion of the data is desired, the Simulation Start Time and the Simulation Termination Time can be set to values in between 0 and x, but these capabilities have not been extensively tested.

```
    Simulation Start Time = 0
    Simulation Termination Time = 100000000000
    Simulation Max Global Iterations = 3106 #Arbitrarily Large
  End
End
```

4.4. Encore Transfer Definition

This block defines how Encore will interpolate data from one region to another. There are three surfaces in the solid_region, `surface_1000`, `surface_1001`, and `surface_1002`, which will map to a single surface, `surface_3`, in the acoustic_region. The data to be transferred is identified as velocity data, which is specific to this particular problem. Options other than velocity mapping can be used and include displacement, acceleration. Consider the already mentioned example of a rotating tire. The solid_region is the outer surface of the tire that contains *sending elements*. The acoustic_region wraps around this outer surface and contains *receiving points* to gather information from the sending elements. In between the sending element nodes and the receiving point nodes exist very small gaps. A contact search is done to find out which nodes are near enough to one another to be considered neighbors. There are two tolerances used to define how far these elements can be apart before they are no longer candidates for data transfer. The first is a `surface gap tolerance` which checks to see if the receiving point intersects a bounding box volume around the sending element plus some small amount. This small amount is the surface gap and in this example can not exceed 1mm. The second tolerance specified is a `geometric tolerance` that will check to see if the receiving point intersects the element volume plus a small amount. For this example, this gap must also be within 1.0mm. Thus, the surface gap tolerance narrows the search to a small number of elements while the geometric tolerance searches through that smaller number of elements to find the "best" sending element to map from.

```
begin Transfer solid_to_acoustic
  interpolate surface nodes from solid_region to acoustic_region
  send block surface_1000 surface_1001 surface_1002 to surface_3
  Send field vel State New To vel State New
  search surface gap tolerance = 1.0
```

```
  search geometric tolerance = 1.0 ## 1.e-6
end
```

4.5. Input/Output Data

The next four blocks of code describe the nature of the input information and where output data from Encore can be found. Take notice that both blocks contain the line `At Step 0 Increment = 1` which prescribes that all time steps are to be processed. If `Increment = 2`, Encore should process every other step, but this capability has not been tested.

In the next block, data is obtained from the Presto .e file. The description `Model Coordinates Are displaced` tells Encore that the `solid_mesh` is a moving mesh that outputs different coordinates at every time step. Continuing, the call `Import Field vel as Nodal Field vel` reads in the velocity data from the `solid_region`. This data is then read back out to another .e file that can be used for debugging purposes. This file can be directed to the location of the user's choosing using the `Database Name` command.

```
Begin Encore Region solid_region
  Use Finite Element Model solid_mesh Model Coordinates Are displaced #mode\
l_coordinates
  Process Initial Condition
  Import Field vel as Nodal Field vel # of Type VECTOR_3D
  Begin Results Output Label solid_region_results
    Database Name = /var/scratch/tfwalsh/gw2.e
    Database Type = exodusII
    Title Grosch Wheel Solid Debug Output
    At Step 0 Increment = 1
    Nodal Variables = vel
    Nodal Variables = displaced
  End Results Output
End
```

While the `solid_region` experiences constant motion, the `acoustic_region`, on the other hand, is fixed. This is denoted by the line `Model Coordinates Are model_coordinates` and means that the nodes associated with the acoustic mesh remain still and collect data regarding their position in relation to the moving solid mesh nodes. The data collected is output with the velocity data to a .e file in the location of the users choosing, and is then used in a subsequent acoustic analysis. The separator characters can be controlled with the keyword `component separator character = NONE`. When set to `none`, velocities will be written as `velx`, `vely`, etc., without the underscores.

```
Begin Encore Region acoustic_region
  Use Finite Element Model acoustic_mesh Model Coordinates Are model_coordi\
```

```
nates
      Process Initial Condition
      Create Nodal Field vel Of Type VECTOR_3D
      Disable Compute Timestep
      Begin Results Output Label encore_results
        component separator character = NONE
        Database Name = /var/scratch/tfwalsh/gw_transfer_new.e
        Database Type = exodusII
        Title Grosch Wheel Solid To Acoustic Transfer
        At Step 0 Increment = 1
        Nodal Variables = vel
      End Results Output
    End Encore Region
  End Encore Procedure
End Sierra
```

The Encore Transfer is now complete!

5. Linear Solvers

Many solution methods rely on reliable and efficient linear solvers. However, there are features in models that may either impede convergence or degrade accuracy. In this section, common issues are tabulated and an example with before and after configurations is reviewed.

1. Some problems occur only for models with lots of constraint equations, due to large surfaces that are tied together (e.g. one large sideset constrained to another with many nodes). A way to confirm that this is the issue is the check whether or not the problem is mitigated if tied contact over large surfaces is turned off.
2. Decreasing the time step (e.g. halving) can mitigate convergence issues.
3. Suppose there are accuracy issues. Note that the tolerance on the residual is always larger than the uncertainty in the solution vector. A linear system has a condition number, which is always greater than 1. The uncertainty in the solution vector is the product of the condition number and the tolerance on the residual.
4. There are alternative to `GDSW`. **Sierra/SD** provides serial sparse linear solvers, `sparsepak` for symmetric positive definite systems, and `SuperLU` for other systems. In addition, `Pardiso` is a general-purpose sparse solver that is available on Intel platforms. These solvers are at least as robust as the iterative methods. It can be enlightening to try to use the appropriate serial sparse linear solver as problem size permits.

Consider, for example, the following user provided configuration of the GDSW linear solver.

```
GDSW
  prt_summary = 3
  solver_tol = 1.0e-5
  max_iter = 5000
  orthog = 200
  overlap = 1
  diag_scaling = diagonal
  scale_option = 1
END
```

The options are generally intuitive. If the solver diverges, then trying a larger `solver_tol` or a larger `max_iter` is recommended. If the solver converges, and accuracy issues arise, then trying a smaller `solver_tol`, and a larger `max_iter` is recommended.

If the solver diverges, a larger `orthog` is recommended. However, there are memory usage limitations. If there is an immediate error that could be related to running out of memory, then try a smaller value of `orthog` or use more processors. See the discussion of reducing memory usage in the training documents for details.

There is a hidden constraint on these options. With some Krylov methods, e.g. the default of `krylov_method` = 1 (GMRES), it turns out that `orthog` $\geq$ `max_iter`. For this reason, when divergence is a problem, users often switch to `gmresClassic`, which allows `orthog` $<$ `max_iter`.

In this example, `overlap = 1` is a small value for overlap. If you are running out of memory with a higher value, then this might be a great idea. If the linear solver is diverging, you might try a larger value (the default is 2).

The `diag_scaling = diagonal` option can be used either to find a convergent solver, or to find a more accurate solver. On the other hand, there are cases in which selecting the option decreases accurate.

In this case study, the user ultimately changed the GDSW configuration to the following to address convergence issues.

```
GDSW
    solver_tol = 1e-12
    overlap = 2
    num_vectors_keep = 0
    orthog = 4000
    max_iter = 4000
    krylov_method = gmresClassic
END
```

The option `num_vectors_keep` can only be used with the classic version of GMRES (krylov_method gmresClassic). The parameter orthog controls how many search direction are stored. We store search directions in order to make the linear solver faster. Generally

more is better but not always. The point to understand is which search directions are stored. In this example, the first 4000 search directions are stored. On later solves, the first `num_vectors_keep` are saved and recycled. The default value of `num_vectors_keep` is `orthog`/2. In this case the solution has changed significantly and you don't want to use any of the old search directions. `num_vectors_keep = 0` tells GDSW to start afresh and remove all search directions every time the maximum is reached. Thus, the benefits of recycling are still retained, but the entire search space is periodically purged of older search directions.

6. Frequency response linear solver

This section is about using the Helmholtz linear solver. The reader is assumed to be familiar with all the other documentation. At this time using `solver_tol` below the default value is not recommended due to observed inconsistencies suggesting that the wrong answer can be returned to the user. Clarifying this issue has a low priority at this time.

6.1. Insufficient virtual memory problems

If insufficient memory problems arise, users must determine their cause and explain them. This is difficult.

Zeroing out `orthogH` conserves memory. Note that the Helmholtz linear solver is less mature than some other parts of GDSW. I have noticed in the past that setting `krylov_methodH` to 1 changed `orthogH` to 1000 (of course 1000 is the default value of `orthog` and 20 is the documented default value of `orthogH`). The **Sierra/SD** parser has default value 0 for `orthogH`. It is necessary to monitor the value reported for `orthogH` in `dd_solver.dat`.

Experiments with alternative mesh partitioners have been surprisingly productive for structures.

`precision_option_0` single conserves memory in theory, but in practice it has been problematic. It would help to use it with Flexible GMRES. Note that Flexible GMRES may interact with `orthogH` like `krylov_methodH`.

6.2. Divergence problems

Address divergence either by adjusting the preconditioner configuration parameters or by increasing the magnitude of the damping matrix. The former has the disadvantage that there are many parameters. Given time the variety of parameters exposed to the user will decrease. The latter has the disadvantage that it can change the solution.

Determining how much damping to use is beyond the scope of this note. If the response is independent of the damping, then there isn't too much damping. The case of slight increases in the response due to the damping are less clear.

Configuring the preconditioner may involve trial and error. One approach is `useParallelDirectSolver yes`. As long as there is enough memory available, the parallel direct solver will almost surely work.

The remainder of these notes concern the trial and error approach to configuring the preconditioner. Start by decreasing the preconditioner update frequency, despite the computational cost.

Increasing the number of levels of overlap may help, particularly with shell elements. There is a theoretical explanation for this.

`Structural_damping` and `viscous_damping` apply to the custom and the operator preconditioners. A formula for the dependence of the preconditioner on these parameters appears in the documentation. The code probably uses this formula. There are two important things to know here. First: these parameters have nothing to do with the damping matrix, and only change the preconditioner. The default values of the structural and viscous damping are respectively 12/100 and 0. Second: sometimes, changing (usually but not always increasing) the structural damping improves the preconditioner (decreases iterations and decreases overall time to solution).

The previous `max_previous_sols` solutions determine an initial guess for the current linear system. The default is zero. I do not know the default initial guess. Although I don't know the initial guess if max previous sols is positive either, it has a canonical value, and I assume that it (least squares) is used. This helps some.

The Krylov subspaces generated to solve the initial linear systems are applied to the remaining linear systems. Only the first `orthogH` Krylov vectors are used. In several studies, the value 100 has proved optimal.

`cull method eigen` is in theory the best way to refresh the Krylov vectors, but in my experience it has never helped.

`SC_optionH yes` helps less often than the default, no, but is worth trying. It is particularly important to type this option correctly. A similar option for other types of linear systems, `SC_option`, is silently ignored for direct frequency response problems.

Preconditioner effectiveness may vary with both input frequency and the number of MPI ranks. Subdomain diameter is inversely proportional to the cube root of the number of MPI ranks. Subdomain mode shape wavelength is proportional to subdomain diameter, and frequency is inversely proportional to wavelength. For these reasons increasing the number of MPI ranks can improve simulation reliability at higher frequencies. My observations are consistent with this prediction. For the same reason at a fixed low number of MPI ranks, as the frequency increases, the effectiveness of the coarse grid correction within the preconditioner may deteriorate. Such deterioration theoretically may be mitigated by setting the `coarse_option` to the non default value `none`. Due to software

defects, this strategy only became an option recently (9/2020). However, this strategy has not helped so far.

7. Comparing Sierra SM Explicit Transient to Direct and Modal FRF

FRFs are a matrix of relationships from forced input to either displacement, velocity, or acceleration output. Most commonly, analysts use acceleration to show the response of the system.

7.1. Frequency Response Functions

The transfer function $[H]$ relates the force input to the displacement between two points in the system. The transfer function is symmetric and is formed as a function of mass, damping, and stiffness. The transfer function is differentiable and the relationship of the force to the acceleration is shown using the following in matrix form:

$$\bar{A} = [\ddot{H}]\bar{F}$$

More information can be found in the theory manual.

7.2. Mesh

Figure 7-4 shows a sample mesh that was used both as an input for **Sierra/SD** Modal and Direct FRF as well as for the Sierra Solid Mechanics Code - Adagio. The Node where force is applied is connected to the beam using a network of rigid Rbars and the force is applied in the Z-direction.

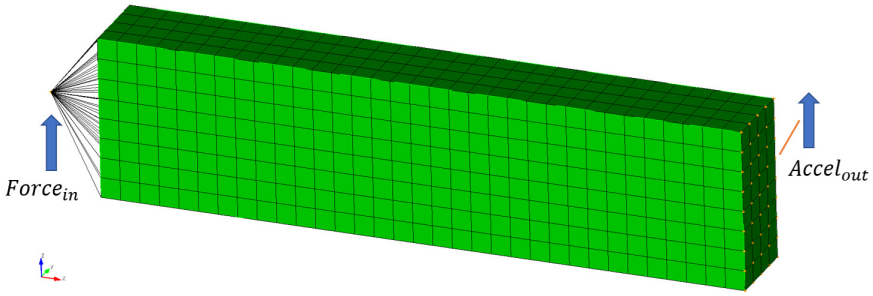

Figure 7-4. – Cantilever Beam FRF example problem. The Input to the system is the Force applied at the Node on the left and accelerations are output at nodes on the left. The input for the problem is provided in Appendix A.16.1-16.3.

7.3. Input File

Figure 7-5 shows the relevant portions of a direct FRF input file. The keyword alpha =5 sets the mass damping of the system. The FREQUENCY section has the frequency range from .1-50Hz at .1Hz increments. A general rule of thumb is that the load in the LOADS should be at least 1.5x the max frequency in the FREQUENCY section.

```
SOLUTION
   case d_frf
      directfrf
END

FILE
   geometry_file = 'beam_frf.e'
END

LOADS
   nodeset 500
   force = 0.0 0.0 1.0
   scale = 1
   function = 1
END

FUNCTION 1
   type LINEAR
   name "white noise"
   data 0.0 1.0
   data 200. 1.0
END

DAMPING
 alpha = 5
END

FREQUENCY
   freq_min = .1
   freq_step = .1
   freq_max = 50
   acceleration
   disp
   nodeset 2
END
```

Figure 7-5. – Relevant Portions of Direct FRF Input File

7.4. Results

Figure 7-6 shows the Z-axis response of the cantilever beam to forced input. The damping for the modal FRF and direct FRF are both same, and there are enough modes for the modal FRF to show nearly exact agreement to the direct frf results. Each of the frequencies used for the adagio input show reasonable agreement. The discrepancies seen are possibly due to the possibility that the alpha damping in adagio is not one-to-one related to the alpha damping in **Sierra/SD**.

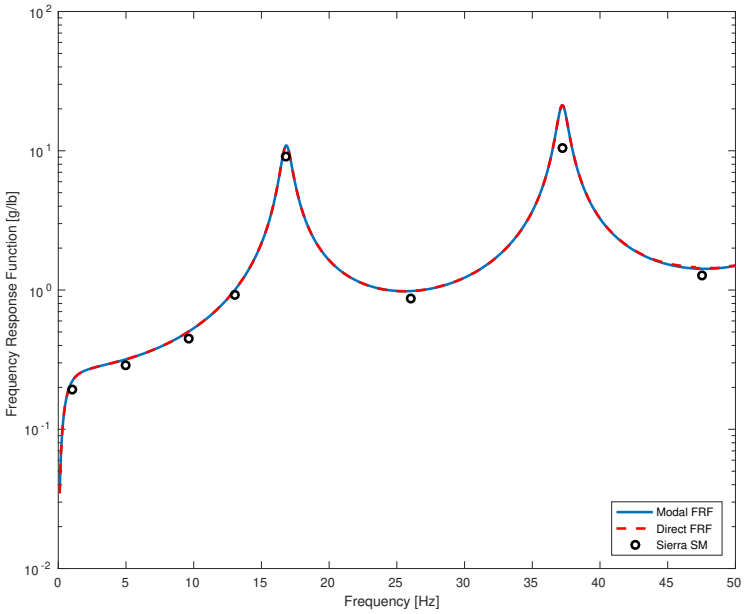

Figure 7-6. – Acceleration of end node in the Z-axis direction.

Table 7-2 shows the amount of time that each method uses. A caveat should be noted here that 10 cycles were used in the adagio input in order to ensure that the system reached steady state. Reducing the number of cycles reduces the compute time proportionally. In addition, with complex systems, it is possible that the amount of time to calculate the eigen solution added to the modal FRF solver time will approach the direct FRF solution time. It also should be noted that adagio run was performed with the knowledge of mode frequency locations. If it were not, it is possible that the frequencies needed to plot would be closer together and more numerous.

Table 7-2. – Solver Timer Comparison

Method	Solver Time (min:sec)
Modal FRF (20 modes)	00:09
Direct FRF	02:41
Sierra SM (8 freqs)	129:48

8. Craig-Bampton Reduction

CBR solution method makes a superelement as specified in the CBModel section of the text input file. The requirements for **Sierra/SD** to use this superelement are in the next section. This reduction is often called a Component Mode Synthesis (or `CMS`).

It can be advantageous to reduce a model to its interface degrees of freedom. This is important in modeling satellites, where the model of the satellite may be much larger than the model of the remainder of the missile. Reduction of the satellite model to a linearized, Craig-Bampton model makes it possible to share the dynamic properties of the model without requiring details of the interior.

A limitation of **Sierra/SD** is that the `CBR` solution reduces the entire structural model to its reduced system and transfer matrices. Other commercial codes can independently reduce different portions of a model to a variety of reduced models in a single run.

8.1. Definitions

There are two types of modes computed and discussed using a Craig-Bampton model reduction.

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.

8.2. Input Required

The following input is required to run the CBR solution.

8.2.1. Exodus Requirements

The only modification of the **Exodus** database is that a nodeset must be defined that identifies the interface degrees of freedom. This nodeset must be large enough to ensure that it constrains the model, i.e. if the node set is fixed, there will be no rigid body modes of the system.

In addition, the CBR model in **Sierra/SD** cannot be applied if any MPC or rigid links are directly applied to the interface. The model must be linearized to perform this reduction, so nonlinear elements and materials are not very meaningful.

8.2.2. Solution

The solution section must contain input for the number of modes. This is the number of fixed interface modes to compute. It must be entered, and will be different than the number of system modes desired. It must also contain shift to ensure that the matrices are not indefinite.

The CBR method has not been tested and verified after preloads or other solution cases that may modify the tangent stiffness matrix. We've only looked at cases where the CBR method is the only case in the solution block. The only exception to this is that an initial solution case computes the system eigenvalues.

8.2.3. CBModel

The `CBModel` defines most of the parameters for the solution. It defines the interface boundary nodes. Note that all degrees of freedom of each of these nodes are a part of the model. Either define all six degrees of freedom as interface dofs, or permit them to be reduced in this step. Interface nodes may be connected to any structural element (solids, shells or beams), but not to a constraint relation.

Selecting the `output_vector` option will output both the fixed interface modes, Φ, and the constraint modes Ψ, to the output **Exodus** file (provided that `disp` output has been selected in the OUTPUT section). These modes are not usually required as a part of the reduction process, but they will be necessary if you should desire to complete a full data recovery after using the reduced model in a subsequent analysis. These modes are the full complement of the displacement data written to the OTM.

Because there are no **Sierra/SD** tests that fail if the OTM is incorrect, CBR solutions that generate an OTM are beta capabilities. A CBR solution case generates the OTM if there is a history section in the text input file.

As a check on the consistency of the model, the eigenvalues of the reduced system can be computed. These eigenvalue and frequencies appear in the text result file, under the heading `eigenvalues of Reduced System`. It is strongly recommended that such analysis be compared with a full system eigen analysis where the interface nodes are fully clamped. This ensures that the model reduction process has not missed data important in the frequency of interest. Select this option with the `GlobalSolution` keyword.

Example,

```
CBModel
  nodeset=1
  format=mfile
  file=cbr.m
  output_vector
  GlobalSolution
end
```

However, finding the eigenvalues and frequencies of the original full system is potentially confusing. One way to determine the full system modes is to use a `multicase` solution with first case `eigen` and second case `CBR`. The eigenvalues of the full system appear in the text result file under the heading `eigenvalues`, as expected. The confusing point is that if the only solution case is `CBR`, then the Fixed Interface Modes appear in the result file under the `eigenvalues` heading.

8.2.4. Output

The output section is used to specify output quantities as well in the usual way. For the `CBR` solution case, the output is the shape functions of the fixed interface and constraint modes.

8.2.5. History

For the `CBR` solution case, the history file contains the Output Transfer Matrix (OTM). Only the following will be honored (others will be ignored).

- displacement
- strain
- stress

Note that transfer matrix for acceleration or velocity is obtained by differentiating the equation.

For MATLAB output, the meaning of the History section of the input text file is expropriated to simplify the testing of the OTM. The history file is used to specify the portion of the model that will be put in the output transfer matrix. The CBR method will use the specification to determine what to write to the OTM if MATLAB format is specified.

8.2.6. Wtmass and Units

The matrices stored in the reduced model are the matrices for the analysis. Thus, the mass matrix elements have been multiplied by the *Wtmass* parameter if applied. As a result, if the eigen analysis is performed on these matrices the resulting eigenvalues will be correct. When such a matrix is used as a superelement input (see Section 19), the matrix is not multiplied by the *Wtmass* parameter again. [2]

[2] For example, if the analyst has a model in inches and pounds, the Wtmass parameter should be 1/386.4 or about 0.00259. This same Wtmass parameter must be applied to both a model reduction step, and to a subsequent superelement insertion.

Since the matrices have an implicit unit associated with them, the analyst must ensure that the units used in the reduced model properly match the units used in the superelement system model.

8.3. Example

The geometry as shown in Figure 8-7 consists of a cone with a nodeset on the top and bottom edge. The model reduction consists in reducing the stiffness matrix from the 80 nodes in this model to the interface nodes (3 nodes on the base in nodeset 3). Thus, there are 18 constraint modes. We choose to retain 4 fixed interface modes for this example. The input is included in the chapter 12.

Running the model and examining the output, you will notice the following.

1. For this example there are two sets of eigenvalues (Ritz values) output to the screen. The first, a set of 10 modes, corresponds to the eigen problem of the unreduced model which includes 6 zero energy modes. The second set of modes is the fixed interface modes of the analysis. The first 4 modes in **CBR-CBR.exo** correspond to these fixed interface modes.

2. The result file, **CBR.rslt**, contains three sets of eigenvalues; the two mentioned above and the eigenvalues of the reduced system. No eigenvectors from the reduced system can be output since there is no geometry database associated with it. The last set of eigenvalues includes every eigenvalue of the reduced system.

   Notice also that the eigenvalues of the reduced system are not identical to the unreduced system. However, even with only four fixed interface modes, the first elastic mode agrees up to the 4th digit. General practice would ensure that the maximum frequency of the fixed interface modes is at least twice the frequency of interest.

3. The `cbmap` is found in both the result file and the reduced model output file. This map relates rows and columns of the reduced system with physical quantities. The first of the 3 nodes in the nodeset has global id 1 as shown in the figure. All 6 degrees of freedom are active at each node. And the `cbmap` has 18 rows.

4. The reduced system is 22 degrees of freedom, which consists of 4 fixed interface modes and 18 constraint modes (6 degrees of freedom associated with 3 nodes). The mass and stiffness matrices are almost full. Generally, the constraint modes contribute full matrix terms to both mass and stiffness.

5. Rerunning with `mfile` added to the output section creates many files that will not be described here including the Φ and Ψ matrices.

6. The output is written to the file `CBR.m`. Output 8.1 contains extracts from this file from which you note the following.

   a) All the data required for the model reduction is found in a single file.

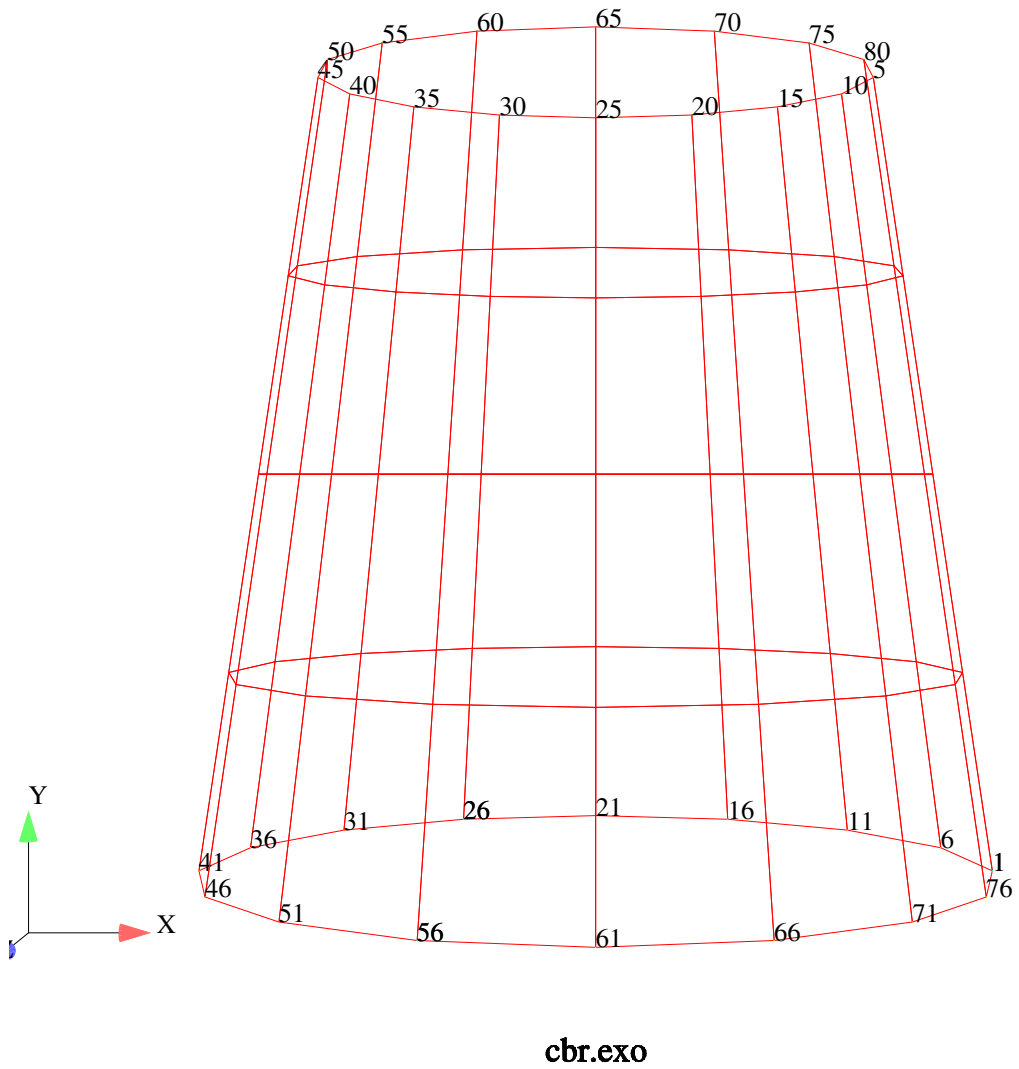

Figure 8-7. – Example CBR model.

b) The map of the reduced model is defined in `cbmap`. A map of the output transfer matrix rows is `OutMap`.

c) There are always 6 degrees of freedom per node in the OutMap. This example does not show this, but there may be fewer in the `cbmap`. Note that while `Kr` and `Mr` are reduced system matrices which must be nonsingular, `OTM` is a transfer matrix and can include inactive degrees of freedom.

```
NumC=18;
NumEig=4;
Kr=zeros(22,22);
Kr(1,1)=7.703363317234302e+04;
Kr(2,2)=9.043236930586677e+04;
...

Mr=zeros(22,22);
Mr(1,1)=1.000000000000000e+00;
Mr(1,5)=-9.545115933105166e-03;
...

% map of nodes in the output transfer matrix
% OutMap is the global node number
% There are exactly 6 outputs per node.
OutMap=zeros(1,32);
OutMap=[1 5 6 10 11 15 16 20 21 25 26 30 31 35 36 40 41 45 ...
OTM=zeros(192,22);
OTM(1,5)=1.000000000000000e+00;
OTM(2,6)=1.000000000000000e+00;
...

%cbmap(:,1) is global node id (1:n)
%cbmap(:,2) is coordinate (x=1, y=2, etc.)
%the first 4 dofs in the matrices are modes,
% while the last 18 dofs are interface dofs.
cbmap=[1 1
1 2
1 3
1 4
1 5
1 6
...
```

Output 8.1. Selected Reduced Model Output

8.4. Verification of the Model

The following are some things that can be done to ensure that the model has been properly developed.

8.4.1. Comparison of Reduced and Full Eigenvalues

It is a very good idea to compare the eigenvalues of the full and reduced system. It will approximately double the computational effort of the model reduction, but there is very little set up time. The example does this. All that is required is to compute the results in a multi-case approach. Begin by computing the eigenvalues of a full system. Then, in the next case compute the reduced order model. By including `GlobalSolution` in the `CBModel` section, the eigenvalues of the reduced system are also computed.

8.4.2. Comparison of Reduced and Full Displacements

It is significantly more complicated to compare the displacements of the two models because there is no automatic upstream data recovery. Manual data recovery will have to be done in MATLAB. We illustrate the method with a small transient run, but it could also be done for a eigen analysis (or statics if the model is statically determinant).

Consider a calculation of 2000 time steps each of 10^{-5} seconds. We impulsively load the structure on the interface (nodeset 3) with a force in the y direction only. The load begins at zero, ramps to 10^6 at $10\,\mu s$, and then ramps back to zero at $20\,\mu s$. Output will be examined on nodesets 1 and 2. This example is found in `CBR_trans.inp`.

Following the calculation, data from any of the output nodes can be evaluated using the history file. The following commands evaluate the x displacement on node 70.

```
unix% exo2mat CBR-tran.h
unix% matlab
load CBR-tran
k=find(node_num_map==70);
plot(time,nvar01(k,:));
```

The reduced model can be used to perform the same calculation. The MATLAB commands to do this work once `CBR.m` has been read into MATLAB are included here.

```
nsteps=2000;
ff=zeros(1,nsteps);
ff(2)=1;
neq=max(size(Kr));
force=zeros(neq,1);
rows=NumEig + find(cbmap(:,2)==2);
```

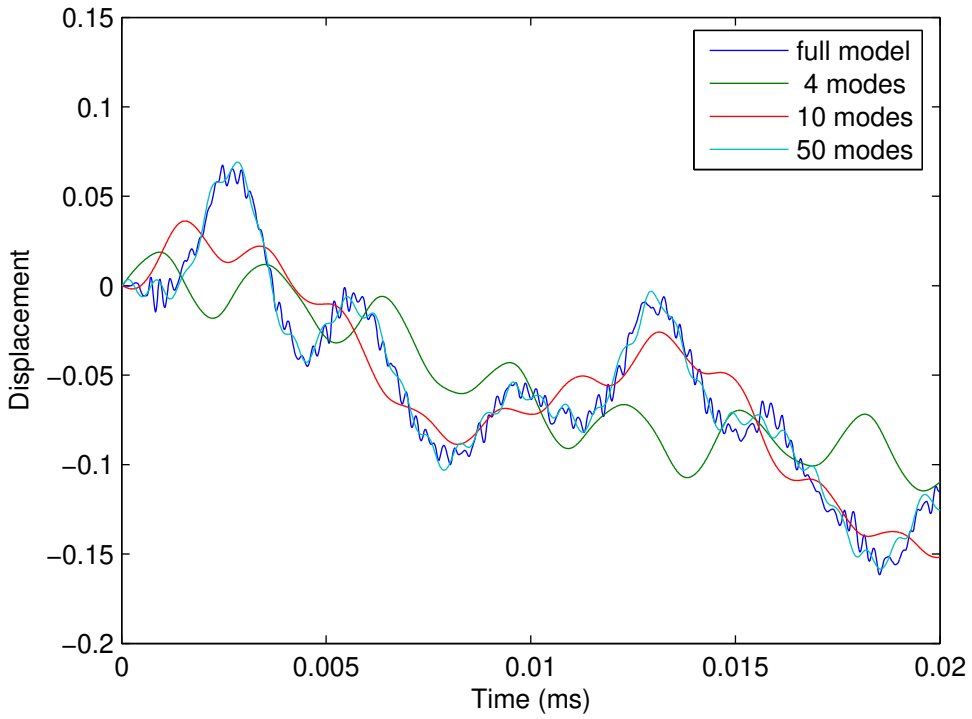

Figure 8-8. – Example CBR transient computations.

```
force(rows)=1e6;
dt=1e-5;
u=CBRint(Kr,Mr,force,ff,dt);
time=(1:nsteps)*dt;
k=find(OutMap==70);
orow=(k-1)*6+1;  % x component of node 70
U70x=OTM(orow,:)*u;
```

The time integration is a standard Newmark integration performed using `CBRint.m`, which is available in the test directory.

Finally we can compare the results, which are shown in Figure 8-8. The data in the figure is obtained by running the CBR reduction with a varying number of fixed interface modes. Note that 4 modes, and even 10 modes are not sufficient to capture the gross response of the structure at node 70. Even at 50 modes there is high frequency data that has been lost. This is as expected since the reduced model is designed to capture only the low frequency response of the structure. The first elastic mode at 21 Hz has a period of 48 ms.

8.5. What to do with the Results

8.5.1. solving the system

The reduced mass and stiffness matrices contain the dynamics of the system. These could be solved in an eigen analysis for example in MATLAB.

```
[evalue,evect]=eig(Kr,Mr);
```

The eigenvalues, `evalue`, represent the system natural frequencies. The eigenvectors are a mix of generalized and physical degrees of freedom. The OTM is used to compute the response on the physical degrees of freedom on the nodesets in the history file.

```
Out=OTM*evect;
```

To find the response on a specific degree of freedom use the OutMap. For example, to find the Z degree of freedom on node 25 of the model.

```
index = find(OutMap==25);
k = (index-1)*6 + 3;
for i=1:size(Out,2)
  fprintf('Mode %d, Z value on node 25 = %g\n',i,Out(k,i))
end
```

When this document was written no process was available to take these results back into an Exodus database so the resulting displacement mode shapes can be plotted on the original model.

8.5.2. Incorporate the reduced model into another system model

This is one of the more important reasons for doing a model reduction. The approach depends on the format of the new model. The following are options.

MATLAB. The model can be combined with other models in MATLAB. The trick is to use the cbmap to tie together different degrees of freedom. I have not done this, but others have expressed interest.

NASTRAN. NASTRAN can do this.

Sierra/SD. As of release 2.5, **Sierra/SD** can input a CBR model in netcdf format as a superelement. See Section 19.

8.6. Limitations

CBR should work with any element. However, none of the interface degrees of freedom should be part of either an SPC or an MPC of any kind.

9. Inverse Problems

Inverse problems are class of problems where some portion of the solution to an analysis is known, but the inputs to the problem are not. Inverse methods solve an optimization problem where the inputs are optimized to match the solution. The current types of input problems supported are Load ID (transient or FRF) and Material ID (Eigen and FRF).

9.1. Experimental Data

For inverse problems, experimental data is typically gathered in a lab. In acoustics, microphones are used to measure acoustic pressure For the transient case, these are measured over a period of time. For the FRF case, these are measured over a series of frequencies. Introducing additional measurements at new data points generally improves the fidelity of the computed solution. On the other hand, the computational difficulty of solving the inverse problem increases too. For this demonstration, synthetic data is generated by solving a forward acoustic problem.

9.2. Inverse Problems - Load-ID

9.2.1. Experimental Model

The experimental model is shown in Figure 9-9. The *Football Model* is an ellipsoidal acoustic mesh, with a cylindrical hole in the middle. 70 sidesets are placed around the exterior of the football, allowing for different loading on each sideset.

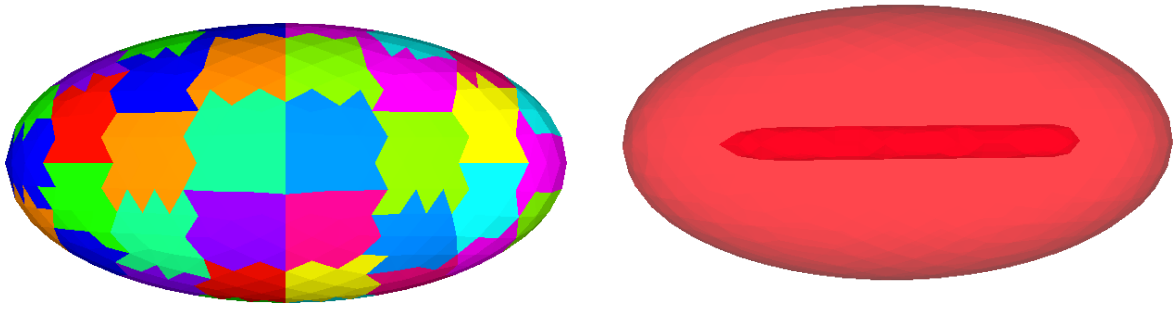

Figure 9-9. – Inverse Football Problem Geometry. On the left, the sideset definitions on the surface. On the right, the interior of the problem.

9.2.2. Forward Problem

To generate the experimental data, the model is solved with the solution method `direct-frf`. A set of human generated loads is used to generate "experimental" pressures at a select set of nodes. Any set of loads can be used. The Matlab function `inputDataProcDynFreqAcoustic_Exo.m` is used to generate the experimental data files: `ttable.txt`, `dataReal.txt`, and `dataImag.txt`. This Matlab function requires the results from the forward run, and a nodeset containing the nodes of interest. In practice, this data is generated experimentally, with the measured acoustic pressures being inserted in dataReal.txt and dataImag.txt.

```
solution
    directfrf
end

loads
    sideset 4
    acoustic_accel = 1.0
    scale = 2
    function = 10001
    inverse_load_type = spatially_constant \\ ignored
end
```

Inverse keywords are ignored when running a forward problem.

9.2.3. Inverse Problem with known loads

Next, the experimental model is solved with solution method `directfrf-inverse`. and `design_variable = load`. For the first run of the inverse problem, the synthetic loads were left in place, as the "initial guess". The inverse problem converges on the first iteration, as the initial guess is the exact solution to the inverse problem. This is an easy way to make sure the input file are correct. The relative tolerance is shown in the first column of ROL_Messages.txt, and the absolute tolerance is shown in the second column of ROL_Messages.txt. If the exact loading is used as the initial guess, the relative error norm should be on the order of machine precision.

```
solution
   directfrf-inverse
end

optimization
   check_grad = no
   optimization_package = ROL_lib
   LSstep = Newton-krylov // recommended
   LS_curvature_condition = null
   max_iter_Krylov = 50 // tolerance on gradient
   opt_tolerance = 1e-8 // of objective function
```

```
    // with respect to parameters
    objective_tolerance = 1e-4 // tolerance on
    // objective function value
    opt_iterations = 50 // before stopping
end

inverse-problem
    design_variable = load
    data_truth_table = ttable.txt
    real_data_file = dataReal.txt
    imaginary_data_file = dataImag.txt
end
```

9.2.4. Inverse Problem with unknown loads

Next, the synthetic loads are removed, and the initial guess for the loading is set to be 0 at all time steps. The inverse problem converged in four iterations, with an objective tolerance of 10^{-4}. The objective norm is a relative measure, and any objective norm of 10^{-6} or smaller is considered more than sufficient. Alternatively **opt_tolerance** can be used to set the absolute tolerance. Recommended values for **opt_tolerance** are problem dependent.

9.2.5. Verification

Finally, the loading output from the inverse run, `force_function_data.txt`, is used to run the forward problem again. This file is designed so that it can replace the function file with no changes. The problem is verified by checking the pressures at the selected nodes against the initial run. Though the loading may not be exactly the same between the initial forward run and the verification forward run, the inverse problem has been solved successfully, as the objective function has been solved to the selected tolerance. To generate loading closer to the initial loading, more nodal data can be added or tolerances can be tightened.

9.3. Inverse Problems - Material-ID

9.3.1. Experimental Model

The experimental model is a solid assembly of two steel blocks joined by a region of viscoelastic foam material. Figure 9-10 shows the geometry of the test model.

As shown in Figure 9-10, the model assembly consists of two equally-sized steel blocks, depicted in yellow and green, joined by a region of viscoelastic foam material, shown in red. The model was discretized with a finite element mesh of Hex-8 elements. A periodic point load with a frequency of 500 Hz was applied to the yellow block, also as shown in the figure. It was desired to calculate the frequency-dependent viscoelastic material properties of the foam block, including complex values for the bulk (K) and shear (G) moduli.

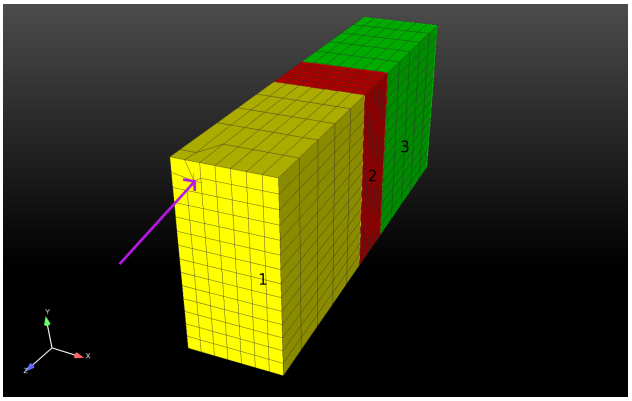

Figure 9-10. – Foam block model with finite element mesh and force location

9.3.2. Inverse Problem input format

The relevant sections of the input used for this example are shown below, followed by some notes about each section.

```
solution
  directfrf-inverse
end
inverse-problem
  design_variable = material
  data_truth_table = ttable.txt
  real_data_file =  data.txt
  imaginary_data_file =  data_im.txt
end
optimization
  optimization_package = ROL_lib
  ROLmethod = linesearch
  LSstep = secant
  LS_curvature_condition = null
  opt_tolerance = 1e-13
  opt_iterations = 100
end

...

block 1
  inverse_material_type = homogeneous
  material 4
  hex8f
end

...
```

```
material 4
  isotropic_viscoelastic_complex
  Greal_bounds  -1000 100000
  Kreal_bounds  -1000 100000
  Gim_bounds -1000000 1000000
  Kim_bounds -1000000 1000000

  Greal = function 2
  Gim   = function 3
  Kreal = function 4
  Kim   = function 5
  density=0.010804
end
```

- **solution section:** defines the type of solution (inverse DirectFRF).
- **inverse-problem section:** specifies the design variable (material) and connects externally defined data describing the test.
  - The **data truth table file** contains the global node numbers where the experimental data measurements are given. For example, the input below gives the number of nodal locations (1) in the first line, followed by the single node id (212) showing all structural dof active (1 1 1), and an inactive acoustic dof (assumed 5th column = 0).

    ```
    1
    212    1 1 1
    ```

  - The `real_data_file` and `imaginary_data_file` contain the real and imaginary parts of the measurement data at each frequency. For example, the input below (from a real data file) gives the number of nodes (3) and frequencies (2), followed by the data at each node. Each frequency requires a separate column of data.

    ```
    3  2
    -4.385640897908e-02     -3.985611576838e-02
    2.898761003889e-02     3.478175302036e-03
    -1.167004279970e-01     -8.319040683879e-02
    ```

- **optimization section:** provides options to control the optimization strategy. See the users manual for more information on available optimization options.
- **block section:** provides an information on the material of the block. Options include,

  known - The material parameters of the block will not be varied in the inverse solution.

homogeneous - Material properties are uniform within the block, and are varied to arrive at the best fit for the data.

heterogeneous - Material properties vary element by element within the block, and are varied to arrive at the best fit.

- **material section:** provides additional options to control the optimization strategy.

9.3.3. Running the Inverse Problem

Next, the experimental model is solved as indicated above. A good choice for the first run of the inverse problem is to use the actual material data used as the "initial guess". This causes our problem to converge much faster than with a general guess, and is a good verification step for problems where the material data is known *a priori*. Next, initial guesses for the material data is set to be something other than the actual value to represent a typical initial guess. Convergence data can be found in the file `ROL_Messages.txt`. The objective norm is a relative measure, and any objective norm of 10^{-6} or smaller is sufficient. Alternatively **opt_tolerance** can be used to set the absolute tolerance.

9.3.4. Verification

For the problem presented here, the following material data is obtained from running the inverse problem (taken from the `name_0.rslt` file):

```
...

Block 1 Viscoelastic Material Properties
  Real Part of K:    40000.002012
  Imaginary Part of K:    -0.005544
  Real Part of G:    15999.999313
  Imaginary Part of G:    5000.000812

...
```

This gives us values that we can then use as part of an input for a forward problem, and see if we obtain the same values given in the input data from the truth table and data files. Though the displacements may not be exactly the same between the initial forward run used to generate the inverse data files and the verification forward run, the inverse problem has been solved successfully, as the objective function has been solved to the selected tolerance. To generate more exact results, more nodal data can be added or tolerances can be tightened.

10. Accuracy in Linear and Eigenvalue Problems

Modal solutions form the basis of much of the analysis performed in **Sierra/SD**. It is essential that we understand the accuracy of the solution computed. eigen pairs may have errors for a variety of reasons, the most common is that the linear solvers all have tolerances, and errors in these solutions feed directly into errors in eigenpairs. It is well known that errors in eigenvectors are typically significantly larger than errors in eigenvalues. If the relative error in an eigenvalue is ε, the relative error in the eigenvector is of the order of $\sqrt{\varepsilon}$.

10.1. Linear Solver Accuracy

Linear solver errors are especially troublesome when the condition of the dynamic matrix is high. This can be caused by various sources.

- Singular mass matrices.
- Lack of a large shift for floating structures.
- Some complex constraint systems.
- Connection of very stiff and very compliant materials.
- Large concentrated masses.
- Poor decomposition, which affect the preconditioner and convergence rate.
- Redundant and/or conflicting constraints.

Any of these items can impact the linear solver sufficient to cause solution failure.

When using the GDSW solver, information on solver accuracy is readily obtained from `dd_solver.dat`, which is written by default. Figure 10-11 provides an example of a portion of this file. The top portion of the file contains information about the general solution. The operator diagonal magnitudes provide a lower bound on the condition of the matrix, in this case 448463. Condition numbers up to 1.e14 are solvable. Higher condition numbers are rarely solvable. The condition numbers are determined *after* application of the MPCs.

The default name of this file can be overridden by the **dd_solver_output_file** option in the GDSW section. Likewise, the default name of the Krylov solver output file ("krylov_solver.dat") can be overridden with the **krylov_solver_output_file** option.

Rigid body norms are then reported. Each row is the product, $|AR_j|$, where R_j is the geometrically determined rigid body vector, and A is the dynamic matrix[3]. Low values for these norms may indicate singularity.

The lower portion of the file provides information about each linear solve. The "recursive relative residual" is computed indirectly as part of the solution. It is used to control the

[3]For eigenvalue problems, $A = K - \sigma M$, where σ is the shift.

solution. At the end of the solution, an "actual relative residual" is computed, $r_a = |Ax - b|/|b|$. Large differences between relative and actual residuals are a concern that the solution may lack accuracy.

The solver is designed to reduce the relative residual to a low tolerance. This residual relates to the error in force in a statics problem. The error in displacement, δx, may be more important for many applications. This error in the displacement depends on κ, the condition of A, and the relative residual. It is not directly computed nor reported.

$$\frac{\delta x}{|x|} \leq \kappa r_a$$

```
-- domain decomposition solver summary --
preconditioner                 = GDSW
Krylov method                  = Right GMRES
solver option                  = Esmond
number of processors           = 1
...
solver tolerance               = 1e-09
maximum number of iterations   = 11
maximum number of restarts     = 1
maximum stored directions      = 0
solving scaled problem         = no
operator diagonal magnitudes --
min                            = 31145.6
max                            = 1.39676e+10
max/min                        = 448463
Rigid Body Norm for Mode 1     = 0.0123875
Rigid Body Norm for Mode 2     = 8.43938e-07
Rigid Body Norm for Mode 3     = 0.012616
Rigid Body Norm for Mode 4     = 0.00206949
Rigid Body Norm for Mode 5     = 0.000878705
Rigid Body Norm for Mode 6     = 0.00423774
coarse space type              = large
number of coarse levels        = 0
solver initialization time     = 0.0306559 seconds

                          Recursive       Actual
                           Relative     Relative
 Solve  Iter  Total  Avg     Residual     Residual    CPU (s)    Total (s)    Avg (s)
     1     1      1    1  7.22136e-12  1.16949e-11  0.00170898  0.00170898  0.00170898
     2     1      2    1  4.55332e-12   1.7662e-11  0.00142002  0.00312901   0.0015645
     3     1      3    1   8.1699e-13  7.89586e-13  0.00141907  0.00454807  0.00151602
     4     1      4    1  5.69584e-14  5.92117e-14  0.00142908  0.00597715  0.00149429
   ...
    39     1     39    1  2.51249e-14  2.34535e-14  0.00145912   0.0559211  0.00143387
    40     1     40    1  2.08119e-14  2.18612e-14  0.00142503   0.0573461  0.00143365
total time for overlap preconditioner (seconds) = 0.0491779
```

Figure 10-11. – dd_solver.dat output from GDSW

10.2. Eigen Solver Accuracy

At the conclusion of an modal analysis, the **Sierra/SD** application reports the eigenvalues and associated error estimates. Figure 10-12 provides an example of this output. The first column of data is the eigenvalue, $\lambda_j = (2\pi f_j)^2$, where f_j is the frequency of the mode. The second column is an estimate of the error bound on the eigenvalue, $\epsilon_j = |(K - \lambda_j M)\phi_j|_2$. Generally, except for zero energy modes, the error bound should be significantly smaller than the eigenvalue itself.

```
Ritz values (Real, Imag) and direct residuals
-----------------------
           Col 1          Col 2
Row    1:  -2.16338D-06   7.34617D-07
Row    2:  2.07696D+07    2.25677D-06
Row    3:  2.07858D+07    8.73909D-07
Row    4:  3.56376D+08    1.48725D-06
Row    5:  4.84777D+08    1.69662D-06
Row    6:  4.84906D+08    5.01020D-06
Row    7:  9.59039D+08    6.06316D-06
Row    8:  1.11917D+09    1.22741D-05
Row    9:  1.11917D+09    3.30643D-06
```

Figure 10-12. – Output of eigenvalues and Associated Error Bounds

11. Wet Modes

Wet modes is a solution procedure that computes the normal modes for a structure partially submerged in a fluid. In appropriate approximations, this may be analyzed as a real Eigen problem of the structure with added mass on the wetted surface.

11.1. Mesh

Figure 11-13 shows a sample mesh for a wet modes problem. The structural mesh is a cylinder composed of four-noded NQUAD shell elements, and the fluid mesh is composed of four-noded tetrahedral elements. The wet mode solution case can be run either with a conforming mesh, or using tied-data with a nonconforming mesh.

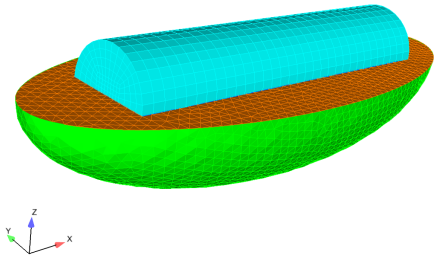

Figure 11-13. – Wet Modes Sample Problem. The structural mesh is shown in blue, and the acoustic/fluid mesh is shown in orange and green. The input for the problem is provided in Appendix A.11.

11.2. Input File

Figure 11-14 shows the relevant portions of a Wet Modes input file. The keyword fluidloading=yes enables the wet-modes solution case. The parameter num_rigid_mode 6 removes the null space for the structural problem. A boundary section is required to set the pressure on the outside of the acoustic mesh to zero. Both structural and acoustic elements are required for a wet mode analysis.

```
SOLUTION
    eigen
        nmodes 20
        fluidloading=yes
END

PARAMETERS
     num_rigid_mode 6
END

MATERIAL fluid
    acoustic
    density 3.46822e-003 // artifically high to demonstrate wet mode capability
    c0 22878
END

MATERIAL steel
    e = 3.0e7
    density = 7.324e-4
    nu = 0.3
END

BOUNDARY
    sideset 1
      p=0
END
```

Figure 11-14. – Relevant Portions of Wet Modes Input File

11.3. Results

Table 11-3 shows the results for the floating cylinder. Note that the density of the acoustic material is artificially high to increase difference between the wet and dry solutions. Adding the fluid mass to the structure reduces the natural frequency of the cylinder.

Figure 11-15 shows the results from the wet mode solution case. Note that much of the symmetry that would normally be found in the dry case is missing. The location of the waterline (located at the midpoint of Figure 11-15) can often discerned from the mode shapes.

Figure 11-15. – Wet Modes Results. The mode shapes from wet modes can be visualized like any other Eigen solution case.

Table 11-3. – Wet Mode Floating Cylinder Results

Mode	Dry	Wet
1	79.82	18.07
5	177.994	46.72
10	207.878	70.11
15	307.325	91.70
20	367.93	117.266

12. Linear Buckling

Several code errors were discovered and fixed in the buckling solution method during May of 2020. This section has not been updated to document the current behavior.

12.1. Shifted Eigenvalue

A challenging part of buckling analysis is determination of the shift parameter. The shift parameter provides a convergence point for the solution, so it should be chosen to be *near* the final solution, but not so near that the solver will fail due to a singularity. The eigen problem of the following system is to be solved is,

$$A = K_m - \lambda K_g$$

Where K_m is the material stiffness and K_g depends on the loading. The problem is solved using a shift invert strategy using ARPACK, where the operator is defined as,

$$A = (K_m - \sigma K_g)^{-1} K_m$$

The buckling load must be multiplied by $-\lambda$ to determine the critical buckling load.

Estimating a shift is easy if the solution has been found, but it is difficult until the loading is determined. Iteration may be necessary in many cases. First, note that the shift, σ, will typically be a negative number for a structure in compression.

Figure 12-17 illustrates data for the ring model shown in 12-16 as a function of the shift parameter, σ. As the shift value approaches the eigenvalue, the solution is found more readily. However, too large a shift results in an incorrect solution. [4]

[4] The input for this example is found in Appendix A.15.

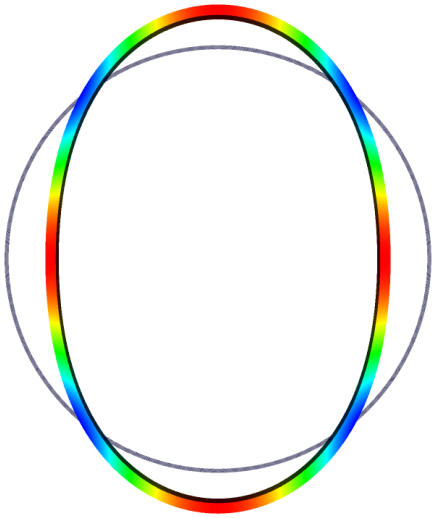

Figure 12-16. – Ring Model for Buckling and Associated Deformation

Shift	Eigenvalue	Time
-1000	-890.381	35
-500	-396.23	35
-400	-396.23	35
-396.23	-396.23	34
-380	-396.23	33
-200	-396.23	34
-100	-396.23	34
-50	-396.23	35
-10	-396.23	35
-1	-396.23	34
1	fail	56
10	fail	53

Shift	Eigenvalue	Time
-1.0000	-396.232	35
-0.1000	-396.232	35
-0.0100	-396.242	35
-0.0010	-394.775	35
-0.0001	-99.8013	35
-1.0e-5	-12.8036	35
-1.0e-6	-0.9631	35
-1.0e-8	-0.0105	35

Figure 12-17. – Solution Dependence on Shift. A shift larger than the computed eigenvalue may generate solver issues (the matrix is negative), while shifts near zero have round off issues.

Recommendations

1. Get the sign of the shift correct. Objects in compression will require a negative shift.
2. If the magnitude of the eigenvalue is greater than the shift, reduce the shift to less than the eigenvalue.
3. You may want to evaluate a shift that is significantly smaller than the eigenvalue. Generally, the eigenvalue should not be sensitive to the value of the shift.
4. The shift selected may impact the convergence of the linear solver. Generally a shift close to the eigenvalue leads to nearly singular linear system and may make the linear solver fail. A shift further from the solution may be easier on the linear solver, but may result in a poor convergence of the eigen solver.

12.2. Buckling Case Study

The pressure load at which the structure buckles is the buckling eigenvalue. This case study shows how to build confidence in a buckling result.

The critical eigenvalue is the mode of smallest magnitude. I prefer to compute 10 modes to check that I have computed the right mode. For example a model with symmetry has multiple mode shapes at the critical eigenvalue. Small eigenvalue residual norms boost my confidence in a result. The residual norms are shown in stdout. Improving the shift or reducing the linear solver tolerance may reduce the residual.

Suppose that initially pressure$= -1$, shift$= -100$ and solvertol$= 1.e-6$. The eigenvalue is $6.1637e4$ and it has multiplicity two. The smaller of the two residual norms is 0.046. The observation $6e4 > .046$ suggests that the eigenvalue might be correct. Given the eigenvalue we can improve the shift. The magnitude of shift should be of about the same as the magnitude as the eigenvalue but not too much larger. Shifting by $-1.e4$ does not change the eigenvalue and decreases the residual to 0.0036. This gives me some confidence in the eigenvalue.

On the other hand if the initial pressure is $-1.e-4$ with the same initial shift -100 and solvertol $1.e-6$ then the eigenvalue is $-2.84241e8$ and the residual$= 3300$ with product pressure eigenvalue $= 2.84241e4$. As this is the initial result nothing yet suggests that it's wrong.

The first hint of a problem is that the smallest magnitude eigenvalue appears in the middle of the table of residual norms in row 6. It would be more encouraging for the smallest magnitude eigenvalue to be either at the top or the bottom of the table.

Here I will exercise my option to try a new shift instead of reducing the linear solver tolerance. The eigenvalue suggests the shift $-1e8$.

With this shift the smallest eigenvalue is at the top of the table. The eigenvalue is $6.16374e8$, the residual norm is .0017, and the product pressure eigenvalue$= -6.16374e4$. I

have no confidence in the results due to the change in the product. However, the new shift reduced the residual by a factor of $2e6$ lending credence to the new eigenvalue $-6e4$. Decreasing the linear solver tolerance to $1.e-8$ leads to similar conclusions.

It is a good practice in this case to try a different initial pressure. The predicted eigenvalue corresponding to pressure $-1e4$ is -6, suggesting the shift -1. The smallest eigenvalue is in the first two rows. This is encouraging. It is also encouraging that the smallest residual is 0.0029. The product pressure eigenvalue $= -6.16374e4$ has been reproduced.

Every simulation that I tried with shift $-1/|pressure|$ reproduced the product pressure eigenvalue $-6.16374e4$. The pressure load that will buckle the structure is the buckling eigenvalue $6.16374e4$.

13. Geometric Rigid Body Modes

This section assumes that the reader is familiar with the parameter `num_rigid_mode`. In **Sierra/SD**, it's possible to use the geometric rigid body modes. There are three examples here. The first example just brings in the rigid modes. The second example uses the modes in solving an eigenvalue problem. The third example uses the modes in a modal transient simulation to deflate out the rotations. An example input is found in the Appendix, (A.10). Rigid body modes are requested in the `SOLUTION` block.

```
SOLUTION
   geometric_rigid_body_modes
END
PARAMETERS
   num_rigid_mode 6
END
```

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

```
SOLUTION
   case out
       geometric_rigid_body_modes
   case flexibleModes
       eigen
       nmodes 10
       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 a total of 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
```

14. Modal Transient

Standard **Sierra/SD** has a fine set of modal based solutions, including a modal transient integrator. However, **Sierra/SD** is designed to focus on massively parallel solutions. It is not uncommon for an analyst to generate a small modal solution, and to use the modal solution as part of a small transient run. Since in modal space, the solution is diagonal, this completely uncouples the modes and allows for an independent solution of each modal amplitude, q_i.

Sierra/SD uses these solutions, but it assumes that the full solution on all output degrees of freedom is required. In other words, the quantity $q_i(t)$ is easily computed, but to transform back to physical space, a fair amount of calculation must be performed, and it is

performed on the full system model. For transient dynamics, **Sierra/SD** performs the following operations.

1. Compute $q_i(t)$ for all modes, i, at time t.
2. Expand to physical space. $x(t) = \phi q(t)$.

   This requires participation of all processors that were involved in the calculation of the modes.
3. Contract to a reduced physical space, if history output is requested.

   This requires communication between processors.

In cases where the analyst requires only a subset of the data, this process can be streamlined by performing the integration outside of **Sierra/SD**. The calculation is fast, and can be performed in serial.

14.1. Process for serial integration

14.1.1. Compute modes of the system model

Modes are extracted in the usual way, i.e. perform a standard eigen extraction on the full system model. Output a reduced order model by extracting a small portion of the eigenvectors to the history file. Element variables of stress and strain may also be output.

```
History
  nodeset 1
  block 12
  displacement
  stress
END
```

14.1.2. Extract Modal force, $\tilde{F}(t)$

The modal force can be written by specifying 'mfile' in the OUTPUT section of the **Sierra/SD** input. The file is named "ModalFv.m". The file contains a matrix of size *nvect* x *nmodes*, where *nmodes* is the number of normal modes computed, and *nvect* is the number of spatial load vectors.

Recall that **Sierra/SD** defines time dependent loads as a sum of products of spatial and temporal functions. For example, consider this example loads section.

```
LOADS
  nodeset 111
    force 1 0 0
```

```
    function 111
  sideset 22
    pressure 1.0
    function 2
END
```

This example time dependent force could be written as follows.

$$F(x,t) = N_{111}(x)F_{111}(t) + N_{22}(x)F_2(t)$$

where the $N(x)$ represents a function of space only, and $F(t)$ is a function of time only. In this example, there are two spatially varying functions, and $nvect = 2$.

We assume that the analyst has access to the time varying functions, $F(t)$, since they are part of the input. Each of the spatial terms is multiplied by the eigenvectors to arrive it the modal contribution.

$$ModalFv = (\Phi^T N_j)^T$$

The total generalized force is then,

$$\tilde{f}_j(t) = \sum_i ModalFv_{ji} F_i(t)$$

14.1.3. Perform Time Integration of Modal Space

Time integration can be performed in MATLAB or other suitable integrator. The file, "modal_int.m" provides an example time integrator using the standard trapezoidal rule. [5]

The result is $q_j(t_i)$, for each mode j in the system, and for each time value t_i.

[5] This is the Newmark-Beta integrator with $\beta = 1/4$, and $\gamma = 1/2$.

We can think of the integration as the solution of three equations in three unknowns.

$$\begin{aligned} \tilde{k}q_{n+1} + \tilde{c}\dot{q}_{n+1} + \ddot{q}_{n+1} &= \tilde{f}(t) \\ q_{n+1} &= q_n + \frac{1}{2}(\dot{q}_n + \dot{q}_{n+1}) \\ \dot{q}_{n+1} &= \dot{q}_n + \frac{1}{2}(\ddot{q}_n + \ddot{q}_{n+1}) \end{aligned}$$

The latter two equations are used to eliminate the $\dot{q}_{n+1}$ and $\ddot{q}_{n+1}$ terms, resulting in the algebraic equation for q_{n+1}.

$$\left[\tilde{k} + \frac{2}{\Delta t}\tilde{c} + \frac{4}{\Delta t^2}\right] q_{n+1} = \tilde{f} + \tilde{c}(\dot{q}_n + \frac{2}{\Delta t}q_n) + \left(\ddot{q}_n + \frac{4}{\Delta t}\dot{q}_n + \frac{4}{\Delta t^2}q_n\right)$$

14.1.4. Expand to Physical Space

The integrated time values can be represented as a matrix Q, where each row of Q corresponds to a normal mode coordinate, and each column represents a time value. The physical space is represented by the product, $\tilde{\phi}Q$, where $\tilde{\phi}$ is the eigenvector in the reduced space.

Using `exo2mat` the eigenvectors are put into six variables. They can be reshaped into ϕ as follows.

```
phi = [ nvar01 nvar02 nvar03 nvar04 nvar05 nvar06];
phi = reshape(something)
```

The transformation to physical space is,

```
XXX = phi * Q;
XX  = reshape(XXX,n,6);
x = X(:,1);
y = X(:,2);
z = X(:,3);
```

Determining the element variables is not much different. A set of element results "eigenvectors" is obtained using evarXX in place of nvarXX. The result is the product ψQ.

14.2. How to Use Results

The results from this calculation cannot be easily visualized as an animated structure because there is typically insufficient data to reconstruct the model. However, time histories of nodal and element data can be examined and plotted.

Related Calculations

Similar calculations are possible with other modal based solutions. For example, a modal frequency response calculation is performed in the same way except that the modal amplitude is given by the following.

$$q_i(\omega) = \frac{\tilde{f}_i(\omega)}{\omega^2 - \omega_i^2 + 2\gamma_i \omega \omega_i}$$

where $\tilde{f}_i(\omega) = \phi F(\omega)$ is given by $Modalfv$ as before. The modal amplitude in this problem is complex of course.

14.3. Limitations

The entire modal must fit in memory. Since this is a linear superposition model, only linear results can be used. Further, while natural stresses can be computed, von Mises and other principle stresses cannot be directly computed, as they are not linear functions of displacement.

The modal superposition method has significant limitations, independent of the particular solution methodology. In particular, the method may be slow to converge spatially if the loading is not well represented by a low frequency mode. Other methods such as the Craig-Bampton reduction can be much better in these cases, though they suffer from having a coupled system of equations.

14.4. Verification

The simplest verification is to run a portion of the time history through the standard **Sierra/SD** modal transient, and compare the results with the results from the reduced order model.

15. Modal Random Vibration

Random vibration is a complex phenomenon. A random input with defined spectral characteristics is applied and the resulting power spectral response is computed. It may be complicated by having multiple inputs with statistically defined cross correlations. The `modalranvib` module in **Sierra/SD** performs this analysis using a linear superposition of normal modes. [6]

15.1. Input Required

The specification of the input for random vibration is complicated. The easiest way to perform this analysis is to copy an existing input specification and correct it for your specific model. The following sections will need attention.

15.1.1. Exodus Requirements

The **Exodus** geometry specification is similar to other solutions.

Random load are often specified as an acceleration PSD, however an enforced acceleration *cannot* be used in the solution method for **Sierra/SD**. Instead of an enforced acceleration, a large concentrated mass may be inserted at the load point, and a *Force* applied to the

[6] See Section 22 for a discussion of the loading for a random pressure loading applied on an extended surface. The `modalranvib` approach is more applicable to a loading on a handful of locations.

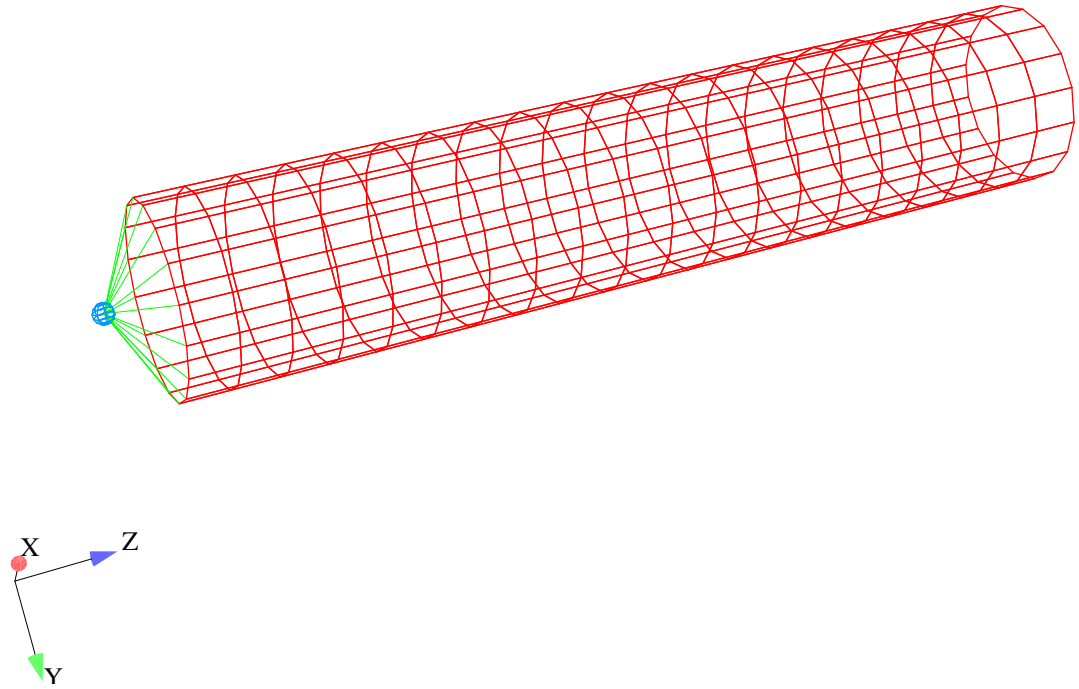

Figure 15-18. – Example Random Vibration Geometry

mass. The load is then distributed to the structure through rigid elements (Rbars) or other means.

A nodeset must be identified on the load point, and node or side sets should be identified on any output points of interest. Be careful of nodal distribution factors other than 1!

As an example, we use the geometry shown in Figure 15-18. The load is applied to the mass on the left of the long tube. We clamp all dofs except the Y at the load point.

15.1.2. Solution

The solution section is fairly straightforward, but note the following.

- While modalranvib can be performed in a single case solution it is strongly suggested that a multicase solution approach be used. Most of the computational effort for a large model is typically consumed in computation of the normal modes. These calculations can be saved using the "restart" option. The calculations of the random vibration results from the modes cannot be restarted.

  Using multicase simplifies keeping track of the output files.

- There are two methods for computing these modes.

  SVD. The default method is the more complete. It computes a vector representing the moment of the solution, and is recommended if detailed statistics on the statistical moments of von mises stress are required.

noSVD. In most cases, the details of the spectral moment are not required. The faster, simpler method is to be recommended. The performance differences only become important when many (hundreds) of modes are involved. The "noSVD" option also provides output of the stress moments, M_2 and M_4.

$$M_j = \int_{-\infty}^{\infty} \omega^j \sigma^2(\omega) d\omega$$

- Two parameters control culling of unwanted modes. The `lfcutoff` is used to control low frequency modes. It is important to set this to a large negative value if you wish to keep rigid body modes that may be important in the calculation of the autospectral response (see 15.3 below). On the other hand, these zero energy modes have no impact on stress, and are by default eliminated from the calculation.

  The `keepmodes` parameter can help reduce the number of modes used in the calculation. It truncates modes based on their activity for the given loads.

15.1.3. RanLoads

This section is the most complicated structure in **Sierra/SD** input files. This complexity is generated by the complex nature of a random input function, $S_F(x,\omega)$. This is an n_f x n_f Hermitian matrix of complex functions, which varies in both space and frequency. (Here n_f is the number of independent inputs). In the case of a single input this matrix reduces to a single real valued function. This is illustrated in the full example of Figure 15-20 (on page 68). The random loads section of a multiple input case is detailed in input 15.1.

Most loads in **Sierra/SD** are described as a sum of spatial and temporal functions. For Random loads this is required, but in addition, the random loads are limited to having the same spatial variation for each row of the matrix. Thus, in a 3 x 3 S_F matrix, only three spatial functions are required. The spatial functions from the example of input 15.1 are defined in nearly the same format as is used in a `loads` section. The balance of the definition is in the `Matrix-Function` section.

```
RANLOADS
  matrix=33   // defines a 3x3 matrix
  load=1      // associates next spatial distr. with row 1
    nodeset 11  // spatial distribution
      force= 0 1 0
  load=2      // associates next spatial distr. with row 2
    nodeset 22  // spatial distribution
      force= 1 1 0
  load=3
    sideset 3
      force=0 0 1
END
```

Input 15.1. RanLoads example for multiple input. In this case, loads are applied at three spatial locations as defined by the sideset and nodesets. The matrix-function determines the correlation of these loads. (See Figure 15-19).

```
MATRIX-FUNCTION 33
    symmetry=symmetric
    dimension=3x3
    data 1,1
        real function 1
    data 2,2
        real function 1
    data 3,3
        real function 3
END
```

```
MATRIX-FUNCTION 33
    symmetry=Hermitian
    dimension=3x3
    data 1,1
        real function 1
    data 1,2
        real function 120
        imaginary function 121
    data 1,3
        imaginary function 131
    data 2,2
        real function 1
    data 2,3
        real function 220
        imaginary function 221
    data 3,3
        real function 3
END
```

Figure 15-19. – Example Matrix-Function. The example is referenced from the `RanLoad` example of input 15.1. Both the left and right columns describe the spectral input to a three input system. On the left, the inputs are completely uncorrelated (as there are no cross terms). The right example provides correlation between the inputs.

15.1.4. Matrix-Function

This section defines the dimension of the input and the frequency functions that define the temporal loading. For random vibration analysis, it *must* be of type Hermitian. Matrix functions may be symmetric if there is no cross correlation, as in a single input system. The matrix function will refer to one or more `function` definitions for the frequency content of each function.

As an aid in model verification, you may want to add `nominalt` to echo the value of the matrix at a single frequency.

15.1.5. Function

The function definition is standard. Note that the "loglog" type function was provided to help in the cases where the function is uses straight line interpolation in the log(frequency) and log(amplitude) domain (which is very common for power input). The units of the

output of these functions is typically $1/Hz$. It represents the frequency variation of the spectral density input.

15.1.6. Frequency

The frequency section is important for these reasons.

1. It provides the frequency band and step size over which the functions will be integrated. This affects the accuracy of the RMS calculations. Note however, that there is little penalty for increasing this quantity since the frequency integral is performed only once.
2. It is used to specify the output of frequency dependent transfer functions. For example, the acceleration PSD is defined as,

   $$A(\omega) = H^{\dagger}(\omega) S_f(\omega) H(\omega).$$

   where H is the acceleration transfer function, and $H^{\dagger}$ is the complex conjugate transpose.

   $$H(\omega) = \sum_i \frac{-\omega^2}{\omega^2 - \omega_i^2 - 2j\gamma_i\omega\omega_i}$$

   Thus, the output specification of the frequency block determines which of these output quantities will be written. Note that there is little point in outputting both displacement and acceleration as they only differ by a factor of ω^4.

   A special consideration should be given to the low frequency end of the frequency block. Rigid body modes are usually undamped, so a singularity may be introduced if zero is included in the frequency band.

15.1.7. Damping

Damping is important to this type analysis. Don't forget it or leave it zero! All types of modal damping specifications are appropriate.

15.1.8. Output

Specification of `Vrms` is the only output specification that is honored for modal random vibration analysis. It triggers output of RMS values of stress, displacement and acceleration.

There are three values of RMS displacement – no results are output for rotational terms. The same is true for acceleration. Note that these quantities are *not* vectors. The RMS values indicate the most likely measurement of the square of the parameter, and includes the unique components of a Hermitian 3x3 tensor. It cannot be combined or transformed as vector.

15.1.9. Echo

The RMS values are typically written to the output **Exodus** file. They could also be written to the log file (or .rslt file) using the **Vrms** option. Some data is *only* available in the log file. If `input` is selected, then the log file will contain a list of those modes that were retained in the modal truncation together with the Γ_{qq} value for that mode. Modes for which the Γ_{qq} term are much smaller than other terms cannot contribute significantly to the total response.

15.2. Example Input

An example input for a single input random load is shown in Figure 15-20. Full detail is found in the Appendix, A.7.

```
SOLUTION
  case eig
    eigen nmodes=9
    shift=-1e5
  case rms
    modalranvib
    keepmodes=3   // force modal truncation
  title 'vtube test of random vibration'
END

RANLOADS
   matrix=1
   load=1
     nodeset 12
        force=0 1 0
        scale 1.00e3  // convert force to accel in g
END

MATRIX-FUNCTION 1
   symmetry=symmetric
   dimension=1x1
   data 1,1
     real function 1
END

FUNCTION 1
   Name='psd'
   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

Frequency
   freq_step=100
   freq_min=300
   freq_max=1e4
   BLOCK=1:2000
END

DAMPING
  gamma=0.01
END
```

```
PARAMETERS
 wtmass=0.00259
END

Boundary
  nodeset 12
  rotx=0 roty=0 rotz=0 x=0 z=0
end

OUTPUTS
   vrms
END

ECHO
    input
END

BLOCK 101
   material 101
  quadt
   thickness= 0.200000003E+00
END

BLOCK 102    // load mass
  ConMass
    Mass=1.00e3
    Ixx =0
    Ixy =0
    Iyy =0
    Ixz =0
    Iyz =0
    Izz =0
    Offset= 0 0 0
END

Block 1000
    RBar  // RBE type elements.  # links 16
END

MATERIAL 101
   density=0.1
   Isotropic
   E=1e+07
   nu=0.35
END
```

Figure 15-20. – Single Input, Random Vib Example.
Most of the input sections for the single input random vibration block corresponding to Figure 15-18 is included here.

- The **solution** block specifies that 9 modes will be computed, but only the 3 most important will be retained in the calculation of RMS quantities.
- The **ranloads** block specifies that the load will be applied only to nodeset 12 (the concentrated mass), and that the force applied will be scaled by 1000 (the load mass). It also points to the matrix function block for further input. The **matrix-function** section defines the load as a single input, and points to the PSD contained in *function 1*.

15.3. Verification of the Model

The obvious things come to mind in verifying the model for use in a random vibration analysis. First, ensure that the model is appropriate for eigen analysis. Mass properties and fundamental modes of vibration can be evaluated. Any rigid body modes should be near zero and not generate significant stress.

Second, the input PSD should be verified. Since the input cannot be provided as an enforced acceleration, it is typically specified as a load on a large mass. Examining the output acceleration at that degree of freedom should reproduce the input power spectrum. There are important issues that must be considered in evaluating the input PSD.

1. The rigid body modes of the system are critical to reproducing the input PSD. Typically, only one degree of freedom is left free on the load point, and that structure is loaded in that free direction. This corresponds to the action of a single axis shaker.

2. Rigid body modes are typically eliminated from the RMS stress calculation. This is done because these modes do not contribute to stress, and they may dominate the numerical solution, making it difficult to identify effects of other resonances. Further, one is often not interested in the rigid body mode contribution to the acceleration or displacement, except for the special case where the output PSD attempts to replicate the input. [7]

   Two factors can cause the rigid body modes to be removed from the calculation.

   - Rigid body modes are typically removed using a low frequency cutoff. This is easily managed using the **lfcutoff** parameter in the solution block. [8]
   - Any mode will be automatically eliminated if it is not a large contributor to the Γ_{qq} matrix. This is more difficult to manage, but is rare for rigid body modes.

3. As noted below, scaling can be a thorny issue.

A word about scale factors and the `Wtmass` parameter is in order. In order to obtain the correct acceleration, the applied force must be multiplied by a scale factor. Note that the spatial term will be squared for terms on the diagonal of S_F, so the units are still units of force (not those of force squared). For models with `Wtmass`=1, the input force is typically scaled by the product of the mass of the large mass times a factor of g to provide in input PSD in g^2/Hz. For English units, where the `Wtmass` parameter is used to scale the mass from lbm to lbf, that scale factor is already entered, and the force should be scaled only by the weight of the large mass. Some examples are provided in Figures 15-21, 15-22 and 15-23.

[7] Sometimes we want to retain the rigid body modes for validation with experiment. This depends on the boundary conditions applied during the test.

[8] The **lfcutoff** parameter *must* be used to retain the rigid body modes if you wish to replicate the input acceleration PSD. However, for numerical reasons, you should *not* normally retain rigid body modes when computing the RMS values of stress or displacement.

Scaling SI units: Consider a case using SI units and WTMASS=1. The acceleration of gravity is $9.8\,m/s$. Our nominal structure has a mass of 17 kg. To enforce acceleration, we add a 5000 kg mass and apply forces to it. We need to apply $1.5\,\mathrm{g}^2/\mathrm{Hz}$ over the band.
We establish the following.

- A PSD function that applies 1.5 at all frequencies.
- We determine that the force applied must be,

$$\begin{aligned} F &= M_{\mathrm{load}} A \\ &= 5000(9.8) \end{aligned}$$

Therefore, we set the scale factor to 49,000.

Figure 15-21. – Scale factors for SI units.

When the force is applied directly to the system, without a large test mass, verification is similar, but care must be exercised on two counts.

1. It is usually best to eliminate all but the rigid body modes from the input verification because system resonances can have a large (and confusing) impact on the results. This can be done by setting the number of modes in the eigen solution to match the number of anticipated rigid body modes.
2. When there is a single input, the product of the output acceleration spectra and the square of the mass should equal the input power spectrum, ($a^2m^2 = S$) provided that the force causes only a rigid body *translation* of the system. Rotations of the system confuse the verification. In other words, apply the load along the center of mass of the system or constrain out rotations in some manner.

Remember that the modal frequency response function can provide direct insight into the transfer functions.

A third verification is important for multiple inputs, where it can be easy to confuse the input to the S_F matrix. It can help to use the `nominalt` option in the solution block to provide an output of the matrix at some nominal frequency.

Scaling inches/pounds: In this case our model is built in inches, and mass is specified in pounds. To correct the mass to proper units, we must use a WTMASS=0.002588. Our nominal structure weighs 0.1 pounds, and to enforce acceleration, we add a 100 pound concentrated mass and apply forces to it. We have a complicated loading, with a maximum of $200g^2$/Hz at 1 KHz. Parameters used are the following.

- Our PSD function matches our complex loading. It has a maximum of 200 at frequency 1000.
- We determine the force to be applied.

$$\begin{aligned} F &= M_{load}A \\ &= (100 \cdot 0.002588)(386.4) \end{aligned}$$

Thus, our scale factor is set to 100.

Figure 15-22. – Example scale factors for inches and pounds.

Scaling English units: Our model is built in inches, and mass is specified in consistent units. We do not need to correct the mass units, so we have WTMASS=1. Our nominal structure has a mass of 258.8e-6 units, and to enforce acceleration, we add a 0.250 unit concentrated mass and apply forces to it. We have a complicated loading, with a maximum of $200g^2$/Hz at 1 KHz. Parameters used are the following.

- Our PSD function matches our complex loading. It has a maximum of 200 at frequency 1000.
- We determine the force to be applied.

$$\begin{aligned} F &= M_{load}A \\ &= (0.250)(386.4) \end{aligned}$$

Thus, our scale factor is set to 96.6.

Figure 15-23. – Example scale factors for English units.

15.4. What to do with the Results

The RMS values of displacement and acceleration can be very useful in determining what portions of the model may be experiencing large deformations or accelerations due to a random load. Unfortunately, RMS quantities are not vector quantities. They are difficult to display on a graphical representation of the data. One suggestion is that RMS displacement values be converted to an RMS radius, and spheres of that radius be plotted on the nodes of the structure.

Typically, RMS accelerations are not plotted on the structure. Such information may be useful for testing subcomponents. The full power spectra of acceleration is available at points specified as acceleration output in the *frequency* block, and may be used for test specification of subcomponents.

Root mean squared values of stress are more readily used, and may be displayed on the model any standard post-processor. Regions of high RMS stress indicate areas prone to failure either through instantaneously exceeding the yield stress, or through fatigue.

15.5. Limitations, Suggestions and Cautions

Must apply the loading directly to the model, you may not use enforced accelerations.

16. Fatigue

Sierra/SD supports two forms of high cycle fatigue analysis. We will use both in this example.

1. Modal Random Vibration, which we will refer to as the "Frequency Domain" solution.
2. Modal or Direct Transient, which we will refer to as the "Time Domain" solution.

Frequency domain fatigue requires three solution cases in the input deck, and the `Fatigue` keyword in the `OUTPUTS` section:

```
SOLUTION
  case eig
    eigen
    nmodes 36
    shift -1e6
  case rand
    modalRanVib
  case fat
    fatigue
END
```

```
OUTPUTS
  fatigue
END
```

Time domain fatigue only requires a transient solution and the `Fatigue` keyword in either `OUTPUTS` or `HISTORY`:

```
SOLUTION
  case trans
    transient
    nsteps 3.5e5
    time_step 1.25e-4
END

OUTPUTS
  fatigue
END
```

Time domain and frequency domain fatigue estimates are not expected to match for several reasons:

- Time domain estimates the total accumulated damage, while frequency domain estimates the damage per second.
- Time domain can represent endurance limits and mean stresses, while frequency domain cannot.
- Frequency domain estimates the expected damage of a random process, while time domain estimates the observed damage. Generating long enough time series for a statistically significant estimate can be costly.

From here on out, we will look at a specific example in detail.

16.1. Example Fatigue Model

16.1.1. Geometry

For this example we will be using a mock printed circuit board model (Figure 16-24) with all dimensions arbitrarily chosen for visual appeal. We will be driving the model with a random force on the underside of the structure while constraining all other translations and rotations to be zero at the drive point. Components are attached to each other using all-to-all contact. We will be focusing on the green electrical pins shown in Figure 16-25.

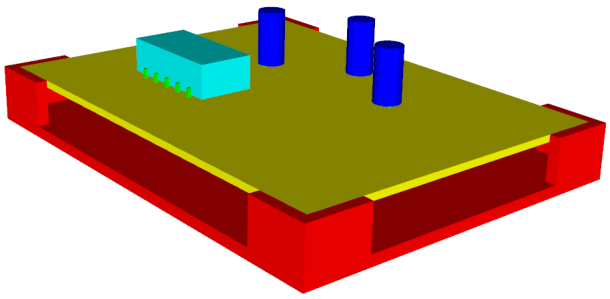

Figure 16-24. – Generic Circuit Board geometry

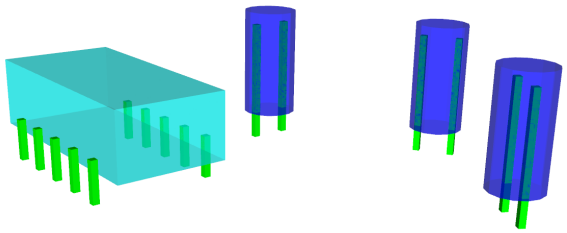

Figure 16-25. – Generic Circuit Board components

16.1.2. Materials

The material properties of the electrical pins are given in **Sierra/SD** syntax as:

```
MATERIAL al_withFatigue
    E = 1e7
    NU= 0.3
    Density = 0.1
    Fatigue_A1 = 20.68
    Fatigue_A2 = -9.84
    Fatigue_A3 =  0.63
    Fatigue_A4 =  0.0
    Fatigue_Stress_Scale = 0.001
END
```

The elastic properties are a rough approximation aluminum, while the fatigue properties are specific to an un-notched 6061-T6 aluminum alloy. The 5 fatigue parameters are:

1. Fatigue_A1, complicated units, strictly positive
2. Fatigue_A2, dimensionless, strictly negative
3. Fatigue_A3, dimensionless, defines the damage contribution from mean stress, strictly positive, 3 is large, 100 is not physical
4. Fatigue_A4, units of stress, defines an endurance limit below which no damage occurs, strictly positive
5. Fatigue_Stress_Scale, optional, conversion rate between model stress units and damage function stress units, e.g. convert *psi* to *ksi*

It is not necessary to define a Fatigue_Stress_Scale, but the option exists to prevent accidental translation errors. The conversion rate of Fatigue_A1 is given by:

$$A1_{new} = A1_{old} + A2 * \log_{10}(1/C), \quad A4_{new} = A4_{old} * C,$$

where C is the conversion rate from old units to new units. Note that **Sierra/SD** does not attempt these conversions directly. Instead, model stresses are converted to material units before being applied to the damage function.

All together, these parameters define the number of cycles to failure N given a stress cycle with peak S_{max} and valley S_{min}:

$$\log_{10}(N) = A_1 + A_2 \log_{10}(S_{max}(1-R)^{A_3} - A_4),$$

where $R = S_{min}/S_{max}$.

In the frequency domain, we are only able to evaluate damage functions which can be represented as:

$$N * S_{max}^m = A,$$

where m and A are material constants derived from $A1$ to $A4$. To reduce 4 material constants down to 2, we set $A4 = 0$, and assume $R = -1$ when doing frequency domain analyses. This limits the types of problems which can be represented accurately in the frequency domain. There will be more discussion of trade-offs later.

Since the geometry is arbitrary anyway, we don't pay much attention to the other components. The base structure and electrical components are modeled as aluminum. The circuit board material is slightly less dense, and significantly stiffer than the aluminum, but still arbitrary.

16.1.3. Loads

The loading for this model is a single-point random force between 10 Hz and 2000 Hz with the autocorrelation function shown in Figure 16-26, evaluated at 0.025 Hz intervals between 10 Hz and 4000 Hz.

By sampling this random function at intervals of 1.25e-4 seconds for 40 seconds (3.2e5 time steps), we are able to generate a very close approximation in the time domain. Figure 16-27 shows a small snapshot of the time domain load, and resulting Auto Spectral Density (ASD)

Figure 16-28 shows a histogram of the force levels seen in the time domain. Note that 4σ peaks exist in the data, and some values approach 5σ.

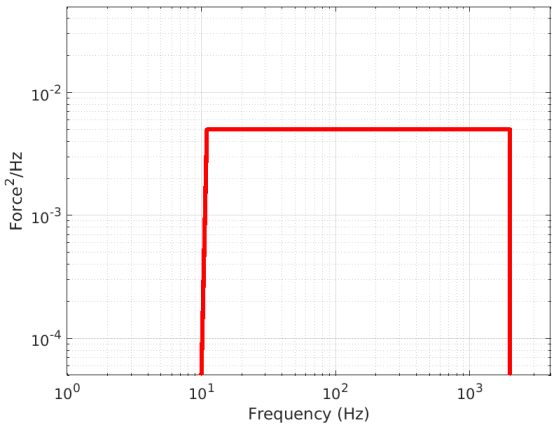

Figure 16-26. – Frequency Domain Loading ASD

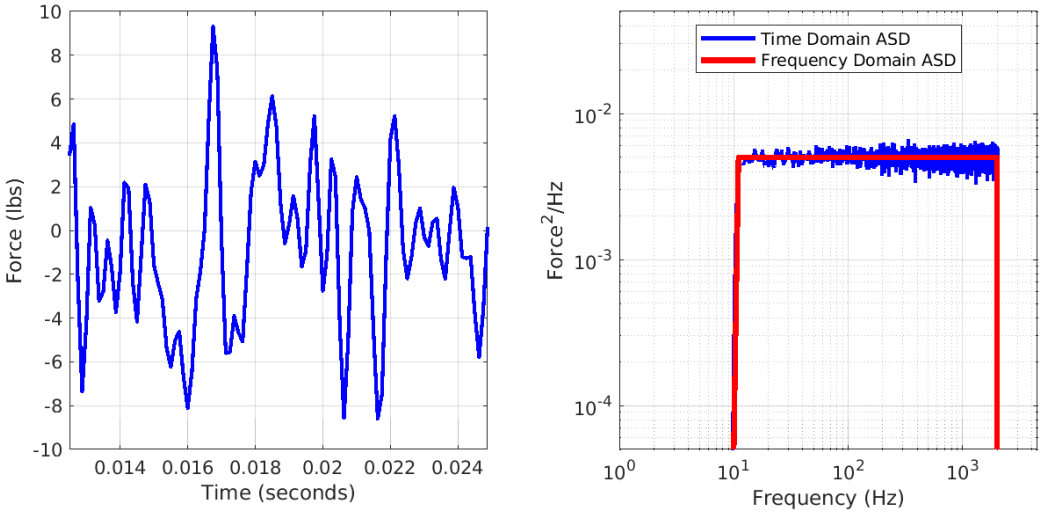

Figure 16-27. – Time Domain Load Snapshot (left), and ASD (right)

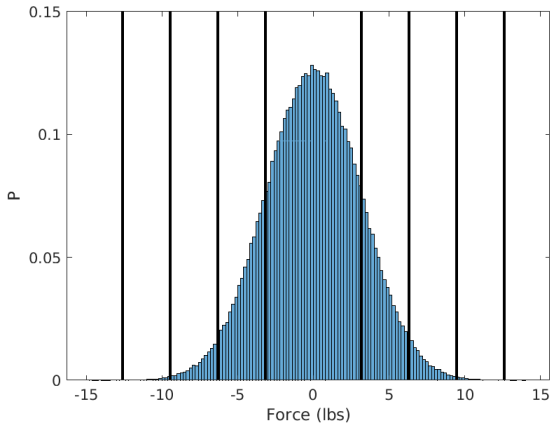

Figure 16-28. – Histogram of time domain loads with vertical bars at 1-sigma intervals

16.2. Results

16.2.1. Frequency Domain

Damage estimates in the frequency domain come in two flavors: "Narrow Band" and "Wirsching". Both are a damage *rate*, representing the damage per second seen by the element. "Narrow Band" damage is intended for solutions where the stress response is occurs at a narrow band of frequencies, while "Wirsching" damage includes a correction factor for wider frequency bands. Unfortunately, **Sierra/SD** does not support spectral density outputs for von Mises stress, and so we have no way of knowing which we should use in this case. Narrow band damage rates are always larger than Wirsching damage rates.

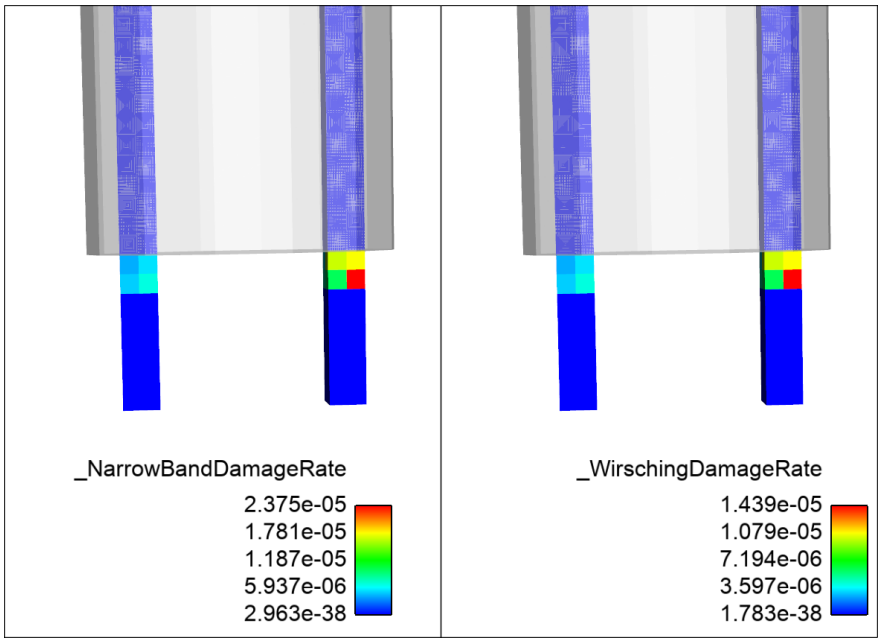

Figure 16-29. – Frequency Domain Damage Rate Estimates

16.2.2. Time Domain

Sierra/SD supports one fatigue damage estimate in the time domain: "Damage". This is an accumulated damage as a result of the transient environment, *not* a damage rate. In our case, the loading duration was 40 seconds, so the largest average damage rate is $3.19e^{-6}$. The average damage rate has been manually calculated in Figure 16-30 for comparison to frequency domain results.

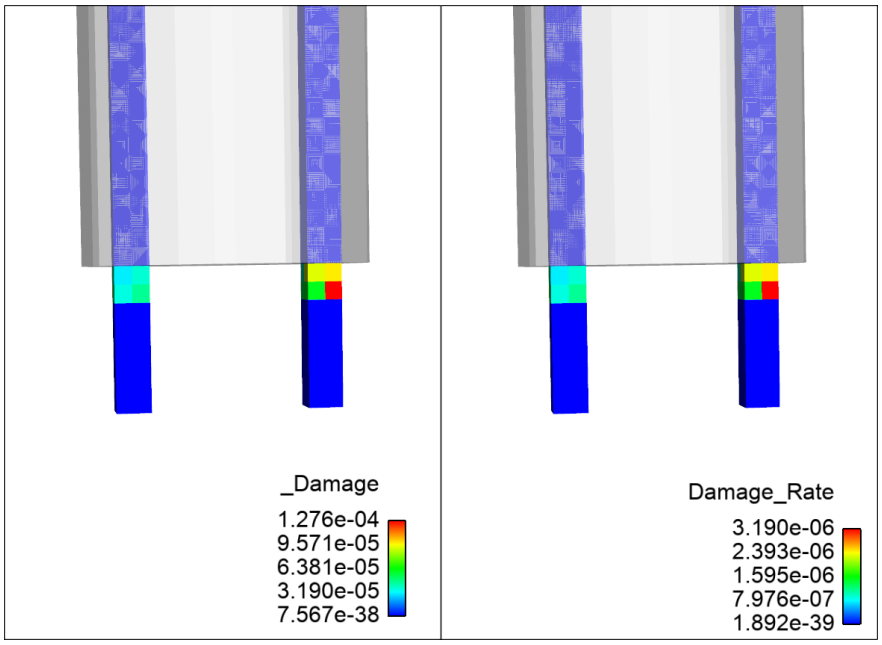

Figure 16-30. – Time Domain Damage Estimate

16.2.3. Comparison

The most obvious difference in these solutions is the cost. The modal transient solution took just over 3 hours to complete, while a modal random vibration solution took only 1 minute with fatigue outputs. Note: Requesting full acceleration and stress output on the pins also requires 3 hours, even in the frequency domain.

The solution quality suffers in the modal random vibration solution. In this example, we chose a material with no endurance limit so that we could make the closest comparison possible, but the frequency domain cannot account for mean stresses either. Together, these details significantly increase the predicted damage in the frequency domain. The peak time domain damage estimate was 4.5x lower than the Wirsching damage rate, and 7.4x lower than Narrow Band. This means the difference between surviving 3.6 days at these levels, and surviving 19 hours (12 for Narrow Band).

Note: The Wirsching damage estimate was not always conservative. One element in particular saw roughly 2x more damage in the time domain than the Wirsching estimate, and was in the 70th percentile of damaged elements. For that element, the Narrow Band estimate was a decent approximation of the time domain (only 13% error).

17. Coupled Electro-mechanical Physics

The term "piezoelectricity" refers to the production of electrical charges on a surface by the imposition of mechanical stress. **Sierra/SD** supports coupled electro-mechanical physics in order to simulate the electro-mechanical behavior of piezoelectric materials when

subjected to an electric field or mechanical stress. One common application of piezoelectrics is in experimental modal testing. Due to the electro-mechanical stiffness coupling, piezoelectrics provide a convenient means to conduct structural dynamics tests since structural vibrations can be converted to electric potentials (i.e. voltages) which can then be stored and processed.

This section demonstrates how to use **Sierra/SD** to simulate exciting and measuring structural vibrations using voltages and piezoelectrics. In this example, we generate a mechanical wave from a prescribed voltage time-history using one piezoelectric tile. The mechanical wave passes through the aluminum barrier and excites the second piezoelectric tile. The deforming piezoelectric tile induces a time-varying electric charge at its surface that we output in terms of voltage.

The demonstration model is shown in Figure 17-31. Symmetry faces indicated in Figure 17-31 mark the surfaces with symmetry boundary conditions. The voltage input and response surfaces are indicated. See Section 17 for the full input deck.

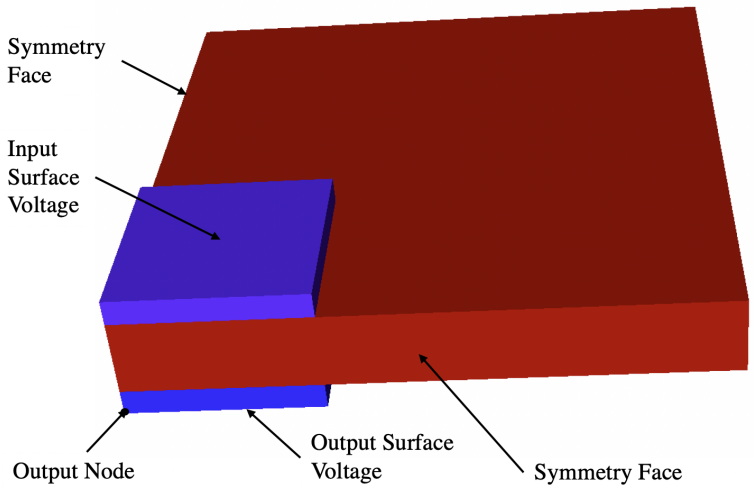

Figure 17-31. – The single patch bimorph model.

17.1. Piezoelectric Material Input

The piezoelectric material constitutive properties must include the orthotropic elasticity tensor (`Cij`), the permittivity tensor (`permittivity_ij`), and the piezoelectric coupling tensor (`e_ij`). Here is the material block for this input deck:

```
// scale = 1e9      // voltage unit scale

// ep = 8.85418782e-12 // permittivity of free space
// D11 = ep * 762.5 * scale * scale
```

```
// D33 = ep * 663.2 * scale * scale

// E11 = -5.20279 * scale
// E33 = 15.0804 * scale
// E15 = 12.7179 * scale

MATERIAL PIEZOELECTRIC
  ORTHOTROPIC_PIEZOELECTRIC

    Cij = 1.39e11   .78e11  .74e11
                  1.39e11  .74e11
                          1.15e11
                           .25e11
                           .25e11
                           .31e11

    permittivity_ij D11 0    0
                    0    D11 0
                    0    0   D33

     e_ij =  0      0     E11
             0      0     E11
             0      0     E33
             0      E15 0
             E15  0     0
             0      0     0

    density = 7500
END
```

Input 17.1. Piezoelectric Material

There are a few important details to note.

- Careful consideration for the coordinate system should be taken when specifying the coupling matrix. The material's poling direction is dependent on the coupling matrix, which should be specified with respect to the global coordinate system (unless a local coordinate system for that material block is specified). In this example, the piezoelectric material is poled in the global z-axis.
- Since the permittivity matrix has units, its entries should be scaled by the permittivity of free space. In this example, we define a variable *ep* for the permittivity of free space.

- We recommend changing the voltage units (volts V) to nano-volts (nV) where $1\ nV = 10^{-9}\ V$. This scaling will significantly improve the condition of the system's stiffness matrix and hence the convergence of the FE solver. See Section 17.4 for more details on solver issues related to piezoelectrics.

17.2. Boundary Conditions

The voltage signal used to excite the mechanical wave is a Gaussian pulse defined by the superposition of a 10 kHz and 43 kHz sinusoidal wave weighted by a Gaussian pulse function (Figure 17-32). The Gaussian pulse is applied to the surface labeled Input Surface Voltage. In this example, we define the voltage time history explicitly with a function. Grounded voltage conditions are prescribed on the barrier surfaces. The following presents the boundary input including the symmetry boundary conditions.

```
BOUNDARY
 sideset 5 //symmetry boundary condition
   x = 0
 sideset 4 // symmetry boundary condition
   y = 0
 sideset 6    // voltage input
   transV = 1
   function voltage_input
 sideset 7  // grounded voltage
   V = 0
END

FUNCTION voltage_input
       type linear
       #include CreateInputDeck/voltage_input.inp
END
```

Input 17.2. Boundary Conditions

In addition to prescribing voltage boundary conditions, we also apply a voltage rigid set to enforce an equipotential surface at the voltage output surface. The surface of the piezoelectric device where voltage is measured is often plated with a purely conductive material such as copper; this physically enforces an equipotential surface. The voltage rigid set simplifies our model by enforcing the equipotential surface without having to model a super thin conductive layer. The rigid set is specified in this problem as follows:

```
RIGIDSET set1
        voltage
        sideset 8
END
```

17.3. Transient Response Results

Figure 17-33 presents the voltage response at an arbitrary node located on the output surface. Since we used a rigid set, the voltage is equal at every node along that surface.

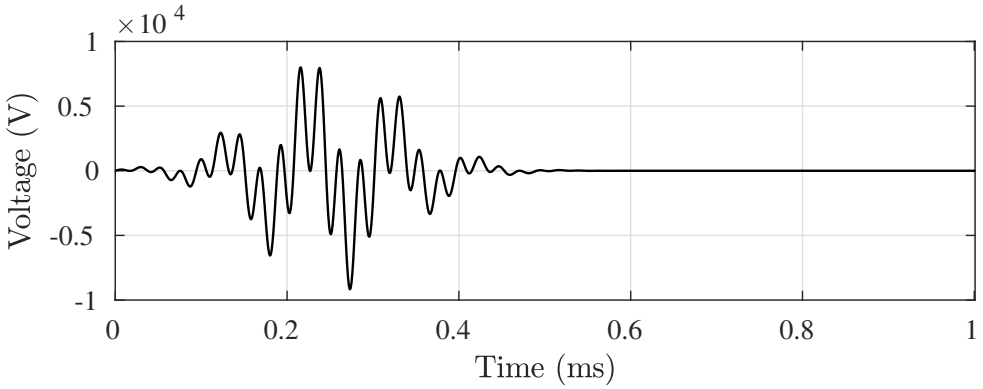

Figure 17-32. – Time history of voltage input (Gaussian pulse)

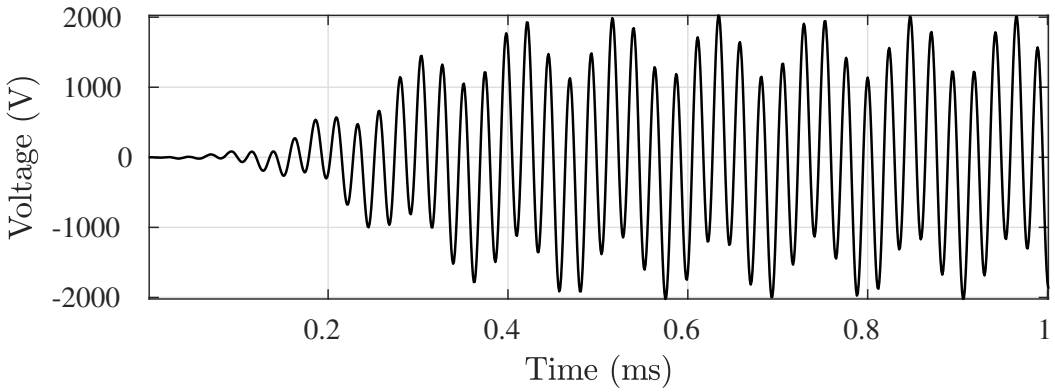

Figure 17-33. – Time history of voltage response

17.4. Linear System Solver Issues and Recommendations

Elastic and permittivity material properties can differ by tens of orders of magnitude, causing ill-conditioning of The coupled piezoelectric global stiffness matrix. In order to improve the matrix condition, we recommend scaling the voltage units (volts V) to nano-volts (nV) where 1 $nV = 10^{-9}$ V. In order to scale voltage to nano-volts, the piezoelectric coupling matrix e_{ij} should be multiplied by 10^9 and the permittivity matrix by 10^{18}.

Another option to account for ill-conditioning is to use the gdsw solver with diagonal scaling. These solver parameters can be specified in the input deck as shown below.

Diagonal scaling should be thought of as a band-aid. If the conditioning of the system can be fixed by scaling the underlying unit system, that is preferable to using diagonal scaling. The solver_tol option controls the tolerance to which the underlying linear system is solved. The default tolerance is $1e-6$, a tolerance of $1e-12$ will give a more accurate solution at the cost of increased computation time. One way to check the convergence of the solver is to see if changing the solver tolerance ($1e-6 \rightarrow 1e-7$) significantly changes the solution. If it does, a tighter solver tolerance is needed. We recommend contacting the **Sierra/SD** team (sierra-help@sandia.gov) in choosing an appropriate set of solver parameters.

```
SOLUTION
   directfrf      // solution method selected
   solver = gdsw  // solver specified
END

GDSW
   diag_scaling = diagonal // diagonal scaling turned on
   solver_tol = 1e-10      // convergence tolerance
END
```

Input 17.4. GDSW Solver Specification

18. System Level Matrices of Viscoelastic FEA Model [9]

In **Sierra/SD**, the constitutive model for an isotropic linear viscoelastic material uses a normalized Prony series to describe the time-dependent decay from the glassy moduli to the rubbery moduli. Following the theoretical development of the finite element formulation in the theory manual, the element stiffness matrices may be cast as:

$$K_{\nu,K} = (K_g - K_\infty) \int B^T D_K B dV \tag{18.1}$$

$$K_{\nu,G} = (G_g - G_\infty) \int B^T D_G B dV \tag{18.2}$$

$$K_e = K_\infty \int B^T D_K B dV + G_\infty \int B^T D_G B dV \tag{18.3}$$

The matrix B is the strain-displacement matrix that depends on the element shape function, while the scalar parameters K_∞, K_g, K_∞ and G_g represent the rubbery (subscript ∞) and glassy (subscript g) bulk and shear moduli. Both D_K and D_G are the constitutive matrices for the bulk and shear terms, respectively. These element stiffness matrices (along with the element mass matrix) are then assembled using standard finite

[9] This section prepared by Robert Kuether, org. 01556.

element techniques, resulting in the semi-discretized equations of motion for a structure with linear viscoelastic materials.

$$M\ddot{x} + \int_0^t K_{\nu,K}\zeta_K(t-\tau)\dot{x}(\tau)d\tau + \int_0^t K_{\nu,G}\zeta_G(t-\tau)\dot{x}(\tau)d\tau + K_e x = f(t) \tag{18.4}$$

This coupled integro-differential equation contains real, symmetric NxN system-level mass (M), viscoelastic bulk stiffness ($K_{\nu,K}$), viscoelastic bulk shear ($K_{\nu,K}$), and elastic stiffness (K_e) matrices. The Nx1 vectors x and $f(t)$ correspond to the physical displacements and externally applied forces, respectively, while the overdot represents the time derivative. The integral terms have a simple functional form, such that the kernel functions are a constant matrix multiplied by a series of normalized scalar exponential functions (Prony series).

One can extract the system level matrices (M, $K_{\nu,K}$, $K_{\nu,G}$, and K_e) directly from **Sierra/SD** by writing out the matrices of an *isotropic linear elastic* FEA model. The mass and stiffness matrices are written to MATLAB "*.m" files when using the "dump" solution type in the **Sierra/SD** input deck. The mass matrix extraction is straightforward since it only depends on the density; however, extracting the individual stiffness matrices is more complicated. A method for extracting the system-level bulk and shear stiffness matrices using the dump solution type is given in Table 18-4. Figure 18-34 provides an

Output Matrix in Eq. (18.4)	**Input Bulk Moduli**	**Input Shear Moduli**
K_e	K_∞	G_∞
$K_{\nu,K}$	$K_g - K_\infty$	0
$K_{\nu,G}$	0	$G_g - G_\infty$

Table 18-4. – Linear elastic material parameters to output system-level stiffness matrices using the dump solution type

example of the input required to extract the $K_{\nu,K}$ stiffness matrix.

```
SOLUTION
  case 'dump matrices'
    dump
END

FILE
  geometry_file 'plate_9by9inch.exo'
END

ECHO
  mass
END

BLOCK 1
  hex20
  material 1
END

//K_g = 9.8039e6
//K_inf = 7.0e6
//G_g = 3.7594e6
//G_inf = 2.5e6

MATERIAL 1
  Isotropic
  G= 1e-4          // essentially zero
  K= 2.8039e6      // = K_g - K_inf
  density=0.00024739
END
```

Figure 18-34. – Sample Input to determine Viscoelastic Matrices

19. Superelements

Superelements can greatly reduce the computational cost of large model. But they are hard to use. Recall from Section 8 that in **Sierra/SD** we have no automatic superelement capability. Superelements are usually used as follows. [10]

1. A full sized, complete model is generated.
2. Portions of the model are extracted, and a reduced CBR model is created from that extracted model.
3. The full model is modified by removing the extracted portions and replacing each with a superelement.
4. The modified model is analyzed.
5. The modified model is post processed.

This section describes each of these steps for a realistic example.

19.1. Superelement Example

The full model is shown in Figure 19-35. The model consists of a the following.

- A lower leg portion consisting of two solid blocks and several beam blocks for applying loads and tying the model together. This will become superelement 1.
- A central joint section representing the bolted joint. The joint is nonlinear, and is the primary interest in the study. It is a single, zero length beam that is attached to the upper and lower leg sections. This will not become a superelement.
- An upper leg section that is similar to the lower leg. This will become superelement 2.

The two superelements are attached in very different ways to illustrate the issues introduced by the connections. The lower model has only two interface nodes, at the centers of the networks 81 and 51. This makes a small structure that is easy to interface. However, because the interface nodes may not be part of an MPC, it also requires that these two networks be beams rather than the rigid Rbars that the analyst would prefer.

In contrast, the upper superelement uses Rbars, but they must be put in the residual structure. Thus, blocks 52 and 82 are not part of the superelement. The consequence is that there are many interface degrees of freedom which greatly complicates interfacing to the superelement, and significantly increases the computational cost of the model reduction.

The joint model (block 53) consists of a single **Joint2G** element. Topographically this is a 2 noded bar element which will be used to control the translations and rotations of the two

[10] This section was originally written 2005. The example described here is no longer available. Only the system mesh single_leg4.exo is archived.

Two Superelement (SE) Model					
Id	**# elems**	**type**	**SE**	**color**	**Description**
81	188	bar	1	blue	lower load spreading network
11	11072	Hex20	1	red	lower support block
12	2158	Hex20	1	pink	lower joint support
51	54	bar	1	cyan	joint connection network
53	1	bar	none	red	joint
52	124	bar	none	blue	joint connection network
21	15024	Hex20	2	yellow	upper joint support
22	2106	Hex20	2	green	upper support block
82	184	bar	none	purple	load spreading network
blocks 61, 62, 63, 71, 72, 72 are not shown and connect the Hex blocks					

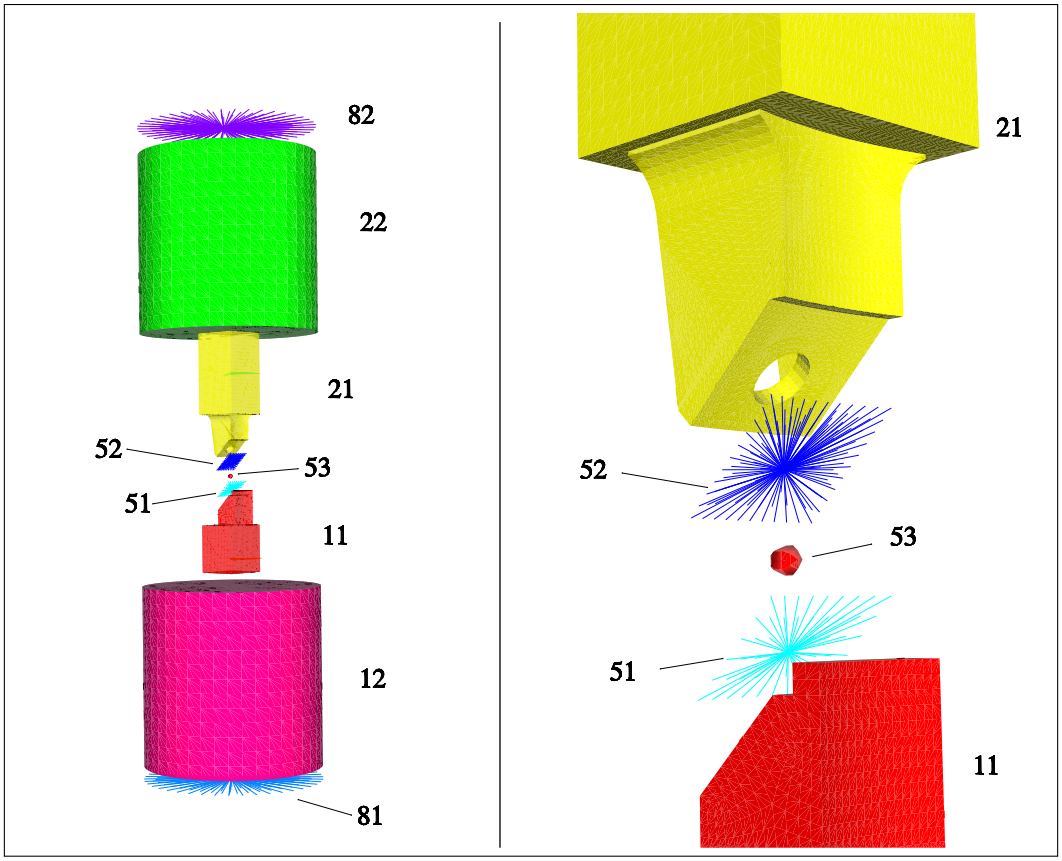

Figure 19-35. – Exploded view (left) of model and (right) zoom view of joint

points. Block 53 is connected to the centers of the two network blocks (51 and 52) which connect to the lower and upper joint supports respectively.

19.2. Submodel Model Extraction and Reduction

The two main ways of extracting a submodel from the original full model are to either 1) build up the submodel from scratch, or 2) pull the model out of the original model. When the model *interface* is complex, I would strongly recommend the second method. This is because there is a lot of book keeping required to assign the interface nodes to the revised model when the superelement is reinserted (see section 19.3). If the node number does not change between these two models, then this book keeping is minimized.

Extracting portions of a system model for CBR reduction may be done using the Grepos utility which preserves the node ordering.

```
$ grepos input.exo output.exo
GREPOS> delete block all
GREPOS> undelete block 1
GREPOS> exit
```

SE1: The lower structure with a small interface

For this model I went into Patran and removed all the elements except those in blocks 11, 12, 51, 61, 62, 63 and 81. [11] In hindsight removing blocks is easier with Grepos than Patran. To define the interface, I defined nodeset 1111 at the center of the networks in blocks 51 and 81. I removed all other nodeset and sidesets, and all empty block definitions. Nodeset 100 was created at random points for an OTM, and the elements were renumbered. No nodes were removed.

A "check" of this model in `explore` indicates that there are 77726 nodes that are not connected to any element. This is as expected, and there are no other errors reported.

The model is split into 10 regions using `stk_balance`, and model reduction is performed on our Linux cluster (liberty). Run times are shown below. Each processor required about 450MB of memory.

[11] Blocks 61, 62 and 63 contain RBar elements tying blocks 11 and 12. For simplicity they are not shown in the figure. Note that it is acceptable to have rigid elements inside the superelement, but not on the interface.

step	elapsed time	comment
matrix assembly	00:12	
CBR restructure	03:58	
fixed interface modes	20:44	computed 50 eigenvalues
constraint modes	25:43	computed 12 constraint modes
model reduction	25:43	
total (10 processors)	25:43	model size: 186 kB

SE2: The upper structure with a larger interface

Again, this model was developed by removing all elements that were not in the superelement blocks (21,22,71,72,73). All the nodes are included to enable using RBars to tie to the superelement. Nodeset 2222 is defined on the end points of all the bars in blocks 82 and 52. No OTM will be used because many nodes are in the interface, so no additional nodeset is created. As in SE1, empty or irrelevant blocks, nodesets and sidesets are removed, and the model generated. The node count did not change. The element count is about 25% higher for this superelement because the mesh of the original model is finer.

The model is split into 10 regions. Run times are shown below. Each processor required about 750MB of memory during the linear solve portion.

step	elapsed time	comment
matrix assembly	00:14	
CBR restructure	06:16	
fixed interface modes	25:30	computed 50 eigenvalues
constraint modes	1:47:39	computed 924 constraint modes
model reduction	1:49:23	
total (10 processors)	1:49:23	model size: 15 MB

19.3. Superelement Insertion

Again, the original model is taken and culled back to only the remaining blocks. We keep only blocks 52, 53 and 82. Sidesets are deleted, as they no longer point to valid elements. The node sets are left in. Empty blocks are removed and the elements renumbered. There are only 309 elements remaining in the model.

Superelements must be inserted into the model. For SE 1, this is easy since there are only two nodes in the superelement. We could use a superelement type, but choose to insert a truss element for later visualization. The nodes for the connectivity may be found in nodeset 1111 in the **Exodus** file.

Superelement 2 is more complicated because the interface is so much larger. *It is important that we maintain the order of the nodes so we have a consistent stiffness matrix.* Because we did not remove any of the nodes from the model in earlier steps the mapping from the superelement back to the new model is greatly simplified.

```
$ mksuper residual.exo
============================================================
| Sandia Tool:  mksuper
| Salinas Release 4.11.0.20090227173358
============================================================


Input Genesis file:  residual.exo
MKSUPER> add nodeset
Enter the nodeset ID.
2222
Adding 308 nodes to superelement.
MKSUPER> write 1leg_se1_and_2.exo
Wrote file '1leg_se1_and_2.exo' with 1 superelements.
MKSUPER> quit
```

Figure 19-36. – Inserting the superelement connectivity in the model.

Because superelement 2 has 308 nodes in the interface, no standard element can be used to represent it. A nonstandard "super" type element must be added to the **Exodus** file. This is done using the `mksuper` application.

There are several ways of defining the nodes for the superelement using `mksuper`. Because this is a large interface, we use the nodeset option. In the residual structure we define nodeset 2222 to apply to the same interface nodes as in the superelement model. We then use these nodes as the connectivity for the element using "mksuper". This step is illustrated in Figure 19-36. The mesh is completed in the file *1leg_se1_and_2.exo*.

The input file is different from the original. We have two blocks associated with the superelement, two blocks associated with the rigid links, and a single block for the joint. A sample is shown in input 19.1, with the map for the smaller superelement shown in input 19.2.

```
SOLUTION
  title 'AFF solid leg modal analysis using superelements'
  eigen nmodes=12 shift -1e6
END

FILE
  geometry_file '1leg_se1_and_2.exo'
END

Boundary
```

```
  nodeset 11 fixed
end

BLOCK 52
  rbar
END

BLOCK 53
  joint2g
  kx=elastic 1e6
  ky=elastic 1e6
  kz=elastic 1e6
  krx=elastic 1e6
  kry=elastic 1e6
  krz=elastic 1e6
END

BLOCK 82
  rbar
END

BLOCK 1001
  superelement
  file=cbrse1c.ncf
  diagnostic=1
  include map.se1
END

BLOCK 1002
  SUPERELEMENT
  file=cbrse2c.ncf
  include map.se2
END
```

Input 19.1. Superelement model input file

```
//    node cid
map   0    0
      0    0
      0    0
      0    0
      0    0
```

```
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
1    1
1    2
1    3
1    4
1    5
1    6
2    1
2    2
2    3
2    4
2    5
2    6
```

Input 19.2. DOF map for superelement 1

19.4. Units and Wtmass

It is critical that the analyst have the proper set of units when inserting a reduced model. See the discussion in Section 8.2.6.

19.5. Visualization

The output of the analysis in the previous section is an **Exodus** model. The structure is limited, but the portions of the model associated with each of the remaining blocks may be visualized. Figure 19-37 shows the response. More development is required for better

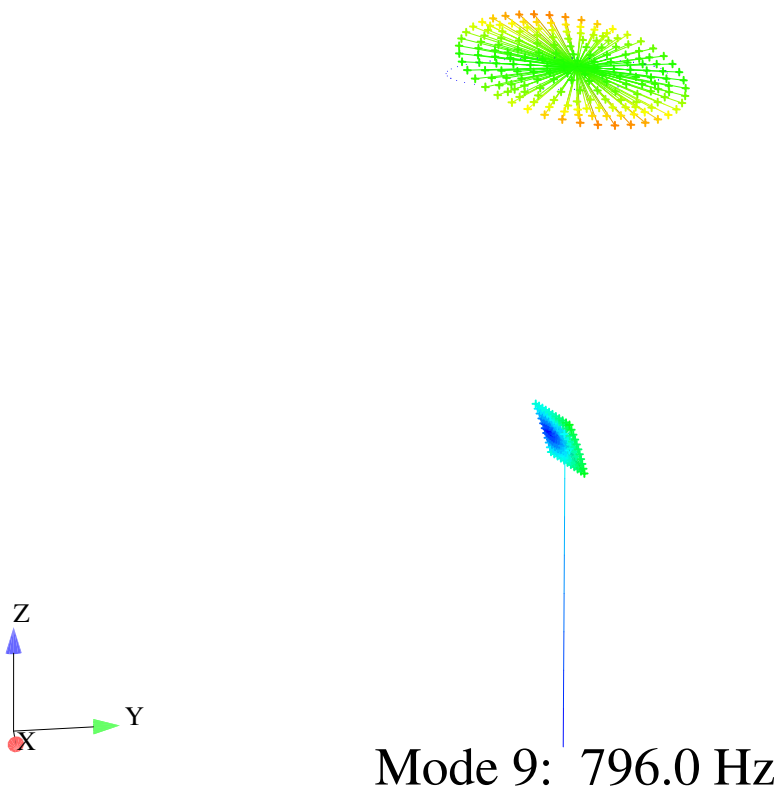

Figure 19-37. – Modal Response of the Superelement

visualization, but the displacements, etc. are available for visualization or for transfer to MATLAB or other plotting packages. [12] Display of the nodes and elements in the output transfer matrix of the superelement is under development.

20. Infinite Elements

In this section, we describe how to use infinite elements for acoustics. These elements serve as both high-order absorbing boundary conditions, as well as far-field calculators that allow the analyst to compute the solution at far-field points outside of the acoustic mesh. This latter step is a post processing step.

The infinite element specification begins with a sideset on the **Exodus** file of interest. Currently, that sideset has to be a spherical surface or part of a spherical surface. Thus, a full spherical surface, hemispherical surface, or a quarter of a sphere would all be acceptable. Note that the infinite element accuracy will degrade if the element surfaces on the spherical boundary do not adequately represent the spherical surface. The finite

[12] Unfortunately many of the visualization tools don't recognize the "superelement" type. For example, in versions before release 8 of ensight, the element and its nodes were not displayed.

element surfaces will be faceted, but enough elements on the boundary are needed to represent the spherical curvature.

Once a sideset is identified for the infinite element surface, the `boundary` section in the input deck would be modified as follows.

```
BOUNDARY
  sideset 1
  infinite_element
  use block 57
END

BLOCK 57
  infinite_element
  radial_poly = Legendre
    order = 5
    source = 0 0 0
    neglect_mass = yes
END
```

Where block 57 contains the infinite element parameters. The number 57 is arbitrary; the user can pick any number that is not assigned to a block in the input mesh **Exodus** file. The parameters are summarized in Table 20-5. Currently, only Legendre polynomials are

Parameter	Description	Options	default
radial_poly	the type of polynomial for radial expansion	Legendre	Legendre
order	the order of the radial basis	0-19	0
source	the location of the source point	any 3 real numbers	0 0 0
neglect_mass	indicates whether to neglect infinite element mass	yes or no	yes

Table 20-5. – Available parameters for the infinite element section

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 spherical surface from which the infinite elements originate. This would coincide with the origin of a spherical coordinate system that is anchored to the spherical surface of the infinite elements. The `neglect_mass` option indicates whether to neglect the mass matrix contributions from the infinite elements. Note that for a spherical surface, the mass matrix contributions from an infinite element are identically zero. However, when numerically generated, small entries will be present in the mass matrix, and thus an option is provided to include these terms in the analysis. It is recommended to neglect the mass in most cases, and thus it would typically be set to `yes`. By default, `neglect_mass` is set to yes.

Note that infinite elements only require a specification of a sideset on the surface of interest. No elements need be set up explicitly on this interface. Internally, **Sierra/SD** constructs virtual elements and virtual nodes that define the actual infinite elements, but the analyst need not build a layer of elements on the boundary of the sideset.

Currently, infinite elements are only set up to work in the time domain. We expect to provide the frequency domain version in an upcoming release.

The infinite element formulation in **Sierra/SD** uses a Petrov-Galerkin formulation, rather than a standard Galerkin formulation. As a result, nonsymmetric system matrices are encountered with infinite elements. This restricts the solver options to the GDSW solver. In addition, special options have to be selected in `GDSW` block to invoke the nonsymmetric solver for the linear solves. If infinite elements are specified, **Sierra/SD** automatically selects the GDSW solver and the correct options for the solve. This makes the process easier for the analyst. However, we list the GDSW options internally selected for completeness. A full input for infinite elements is found in the Appendix (A.8).

```
GDSW
  matrix_type   nonsymmetric
  krylov_method 1
  solver_tol    1.0e-9
  scale_option  1
  coarse_solver LDM
  I_solver      LDM
  O_solver      LDM
END
```

Note that the `nonsymmetric` option lets the solver know that it should be expecting a nonsymmetric matrix.

20.1. 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 period of time. The `linesample` capability may be used to write out the far-field data. This data may be written in a readable MATLAB format, which can easily be read in to create plots of the data.

Linesample is placed in the `linesample` section as follows.

```
linesample
    samples per line 10
    endpoint 0 0 0      1 0 0
    endpoint 0 0 0      0 1 0
    format mfile
end
```

This example creates two lines, with 10 samples per line. The first line runs from the origin for one unit in the X direction. The second line extends from the origin in the Y direction.

For example, the following section,

```
linesample
    samples per line 2
    endpoint 0 0 50     0 0 50.001
end
```

will instruct **Sierra/SD** to output the acoustic results at the 2 points $(0, 0, 50)$ and $(0, 0, 50.001)$. Since these 2 points are very close, the output will be almost the same. Thus, this is an example of using linesample to output the results at a fixed point in space.

The output will be written to a `Matlab` m-file with the name "linedata.m". One file is written per analysis (results are joined analogous to history file output). For example, reading this file in will create vectors `Time` and `Displacement`. In our case Displacement is just a placeholder for the acoustic pressure.

The infinite element output in the far-field is always given with respect to some time shift. This is due to the properties of the inverse Fourier transform. Details of this are given in the theory notes on infinite elements. The time shifts are included in the linesample output for the analyst to use. These will allow for plotting the time histories against the appropriate time vectors. For example, to apply the time shift to the first point in the linesample data, one could use the following MATLAB command.

```
shifted_time = time + TimeDelay(1);
```

One TimeDelay value is available for each sample point in the linesample output.

Once the time data is properly shifted, the following command in MATLAB will plot the pressure for the first sample point.

```
plot(shifted_time,Displacement(1,:))
```

21. Acoustic Scattering

Acoustic analysis often includes the concepts of a "scattering" solution. By this, we mean an analysis where it is relatively easy to specify the incident wave at all points in space, and we solve for the reflected wave. Such analysis is seldom done for elasticity because the input medium is not usually homogeneous and an *a priori* specification of the incident wave is a challenge. Such scattering solutions are useful in a variety of contexts. A submarine in the ocean may be struck by an incident "ping" from a neighboring ship. Such a ping is nearly a plane wave, and calculation of the outbound wave is the item of interest.

The total acoustic pressure (which is the sum of the incident and scattered components) may not be important. Because the incident wave is known, we do not need to model the vast region of space between the incident source and the scattering object. This greatly reduces the cost of the computation.

The theory manual details the formulation. There are several salient issues.

1. The same PDE is solved for scattering and full pressure solutions.
2. The acoustic scattering loads are applied analytically as a pressure on the wet surface of the structure.
3. A conjugate load is applied to the wet surface of acoustic medium. Thus, there are two loads applied: a pressure load, P, on the elastic medium, and a velocity load on the acoustic medium. For a plane wave, $v = \frac{P}{\rho c}$.
4. Because there are two such loads, we have designed a limited number of specialized functions for application of these loads. These functions ensure compatibility between the elastic and acoustic portions of the model.
5. The natural output quantity is the scattered pressure.
6. Typically, absorbing boundary conditions are applied to the exterior of the mesh to reduce reflection of the scattered wave.

Because scattering solutions use the same PDE as the full pressure calculation, the analyst could complete an analysis by applying these loads independently. Using the scattering loads and set up provides a more robust and simpler interface to scattering problems.

21.1. Scattering Sphere

The sample problem is an elastic sphere floating in an infinite acoustic medium. The meshes for the sphere and fluid do not match at the interface, so tied surface specifications must be used. The example problem is illustrated in Figure 21-38. A full example is listed in the Appendix (A.13).

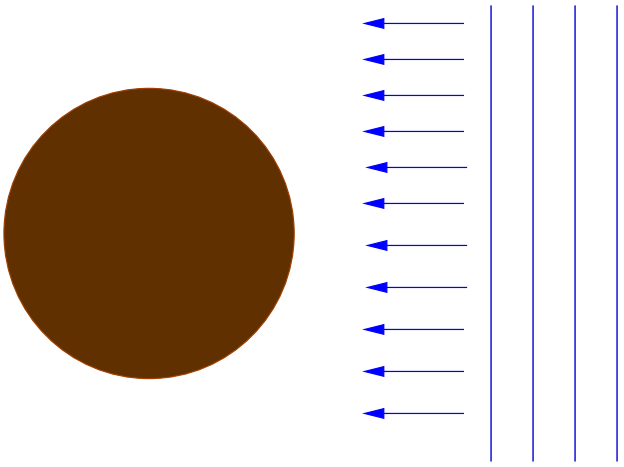

Figure 21-38. – Elastic Sphere in Fluid Example

Solution

Within the solution section of the input, we specify the "scattering" keyword. This informs the solver that consistency must be maintained between loads, and output pressures will be scattered pressures.

Loads

The loads section should have a load applied to both the elastic and the fluid portions of the model. In the example input of Figure 21-39, sideset 1 is the surface of the elastic material, and sideset 2 is the corresponding surface of the fluid. Note that there are no checks made on this loading. However, if the loads are not applied in pairs, the analysis is meaningless.

While other structural loads can be applied in a scattering problem, it is incorrect to apply acoustic loads other than scattering loads. This is because we are redefining the acoustic variables to apply to incident pressures. We cannot define the variable as "incident" in one portion of the analysis and "total pressure" in another portion.

Functions

The functions referred to in the loads section must be capable of applying different functional responses to the elastic and acoustic regions. Specification of the "scattering" keyword in the solution section permits us to check this for consistency.

Tied Data

Because the elastic and acoustic regions of the model are not compatibly meshed, the surfaces must be tied together with a tied surface specification. Sidesets 1 and 2 are again applied. It is not necessary for the scattering problem to use tied data sections if the regions have compatible meshes.

Outputs

Specification of "apressure" outputs the scattered pressure.

```
Solution
    case out
        transient
        scattering
        nsteps=1000
        time_step=0.001
end
Loads
    sideset 1
        pressure=1
        function=1
    sideset 2
        acoustic_vel=1.
        function=1
End
Function 1
    type=Plane_Wave
    material=water
    k0=450.
    direction -1 0 0
end
Tied Data
    name surface1-2
    surface 1,2
end
Outputs
    apressure
    displacement
end
material water
    c0 = 5000
    density = 1
end
```

Figure 21-39. – Example Scattering Input

22. Random Pressure Loads

In a previous section we discussed random vibration input (see section 15). That section addresses a loading where the frequency content (or power spectral density) of the loading is known for a few points on the structure. In contrast, for hypersonic vehicles a random loading may occur at thousands of points on the surface. Many aspects of the loading are the same, but the specification is different, and for performance reasons, the solutions are performed differently.

The starting point for this analysis requires the following.

1. A surface sideset where the loading will be applied.
2. A temporal correlation function to apply on the surface. The temporal correlation function is the inverse Fourier transform of a power spectral density (PSD).
3. A spatial correlation relation. Currently, that relation may only be specified as a pair of exponential decay constants.

Details of the problem setup may be found in the *User's Manual*. This section provides a simple example of the setup and an informal discussion of the sources of the data.

22.1. Example Problem Set-up

For our example, we consider a cylinder in a flow field as shown in Figure 22-40. The structure is a right circular cylinder of diameter 1 unit, and height 2 units. The flow is directed towards this cylinder in the X direction, and the PSD and corresponding temporal correlation function are shown in Figure 22-41. Input is found in the Appendix (A.9).

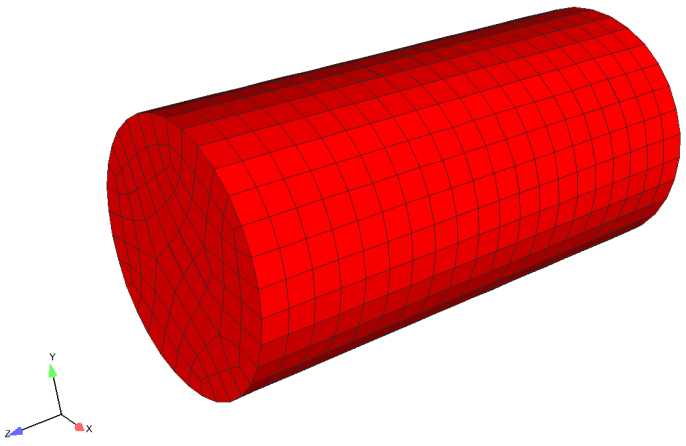

Figure 22-40. – Example Random Pressure Geometry

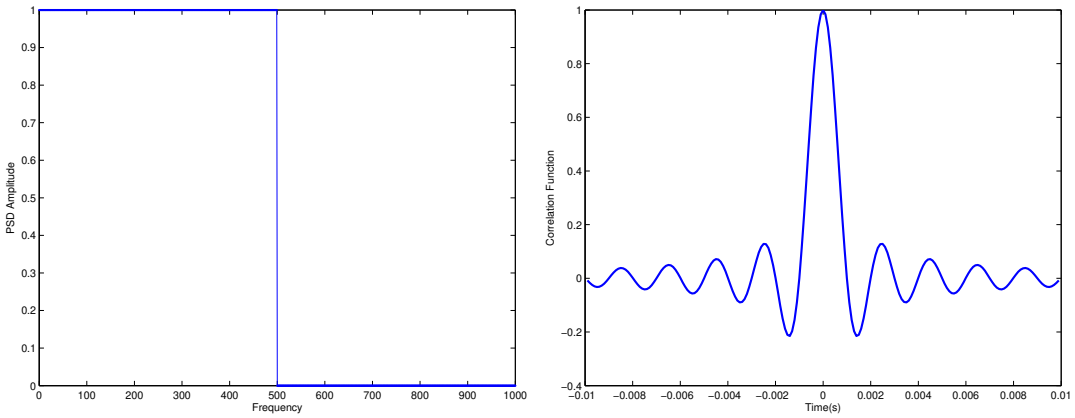

Figure 22-41. – Example Random Pressure PSD and Correlation Functions

We are interested in this example, in frequencies up to 500 Hz, so the cutoff frequency is 500 Hz. There is no point in adding energy above the desired cutoff frequency – it only complicates the procedure. [13] The PSD of the input thus controls much of the solution.

The spatial correlation is often more difficult to obtain. For our example, we require a decay constant of 2.0 units in the flow direction, and 5.0 perpendicular to the flow. One can think of corresponding decay distances of 0.5 and 0.2 respectively. Thus, down the flow, points more than about 1.5 units away will not be well correlated. [14] Perpendicular to the flow, correlations decay even faster.

One rarely has much experimental data about the spatial correlation. Some information is sometimes garnered from the temporal correlation. For example, if the correlation function has a characteristic time, τ, one would expect the spatial correlation length to be of the order of $\delta = v\tau$, where v is the flow velocity. For a structure in a fluid, the dimensions of the turbulent layer also provide a bound on the spatial correlation.

22.2. Example: Input Specifications

The physical quantities of the previous section can be interpreted and expressed as **Sierra/SD** input as follows.

- The temporal correlation function of Figure 22-41 can be digitized as a **Sierra/SD** function. In Figure 22-42 we use a triangular pulse for simplicity. The correlation function should by symmetric about the origin, and it should have the value of 1.0 at the origin. The `correlation_function` is used in the load section, as shown in Figure 22-43.
- In the loads section, we also define the following quantities.

[13] Although the physics has energy above 500 Hz, cutting off the PSD at 500 Hz. is required because a higher cutoff frequency narrows the correlation function with no added accuracy.

[14] correlation=$\exp(-3) = 0.0498$

cutoff_freq	= 500	
coordinate	= 1	to set flow direction
ntimes	= 5	varies from 3 to 20. Too small causes poor replication of the temporal correlation function. Too large results in ill conditioning and singularity.

Recall that the full correlation matrix is a tensor product of the spatial correlation with temporal components. The "NTimes" parameter controls the number of samples in the time domain.

All that remains is setting the spatial correlation decay constants in the loads section. The full text is shown in Figure 22-43.

correlation_length_z = 0.5
correlation_length_r = 0.2

```
FUNCTION 1
    type linear
       data  -0.001909859319285  0
       data  0                   1
       data  0.0019098593192856  0
END
```

Figure 22-42. – Random Pressure Correlation Function. The temporal correlation is digitized as a "time only" function. For purposes of illustration, we use a simple triangular function here.

```
LOADS
    sideset 1
            randompressure
            cutoff_freq = 500
            delta_t = 0.001
            correlation_function = 1
            ntimes = 5
            correlation_length_z = 0.5
            correlation_length_r = 0.2
            coordinate = 1
END

BEGIN RECTANGULAR COORDINATE SYSTEM 1
            origin 0 0 0
            z point 1 0 0
            xz point 1 0 1
END
```

Figure 22-43. – Random Pressure Load Section. Note that the "flow" direction is the Z coordinate direction of coordinate frame 1.

22.3. Example: Verifying the Load

This is a fairly complex input, and it is advisable to verify the generated loads to ensure consistency. We examine four quantities.

1. The average force on a node.
2. The variance of the force on a node.
3. The temporal force correlation on a single node.
4. The cross correlation of forces between nodes.

All of these quantities require output of the total input force, which is obtained by specifying "force" in the "outputs" section of the **Sierra/SD** input. We will use MATLAB tools to evaluate many of the results. Data can be read into MATLAB from the **Exodus** results using "`exo2mat`" or using other methods.

22.3.1. Average Nodal Force

The average nodal force may be determined either by evaluating the MATLAB results directly, or using the "statistics" output from **Sierra/SD**. The built in statistical output is easiest. Supply the "statistics" keyword to the "outputs" section, and results will be written to an additional **Exodus** file. This has the added benefit that these results may be easily visualized using Paraview or Ensight. See Figure 22-44.

For long time integration, the average value of the nodal force should approach zero. Shorter time samples will have greater variation. The random variables depend on "`cutoff_freq`". The number of random samples can be computed as,

$$N_{samples} = Time_{analysis} \cdot \text{cutoff\_freq}$$

The fractional mean of the force should be within about $3/\sqrt{N_{samples}}$. Or,

$$Error_{mean} = \left| \frac{F_{mean}}{F_o} \right| < \frac{3}{\sqrt{N_{samples}}}$$

Here F_o is the force applied for a correlation function of 1. It involves the scale factors of the function, the sideset distribution factors and the effective area for each node. [15] See the comments section, 22.4 for, discussion on the effective area.

For the example in Figure 22-44, mean forces are of the order of 1/1000. In this example, we took 10,000 time steps, with each of 0.1 ms for a total time $Time_{analysis} = 1\,s$. With $Delta\_T = 1/\text{cutoff\_freq} = 1\,ms$, the total number of random samples is $N_{samples} = 1000$.

For nodes in the center of the loading area, the effective area is about 0.0098 square units. Because the sideset distribution factors are all one, we have $F_o = 0.0098$. Then,

$$\begin{aligned} Error_{mean} &= \frac{0.001}{0.0098} \\ &= 0.1 \end{aligned}$$

which is greater than $\frac{3}{\sqrt{1000}} = 0.095$. A distribution of the mean is shown in Figure 22-45.

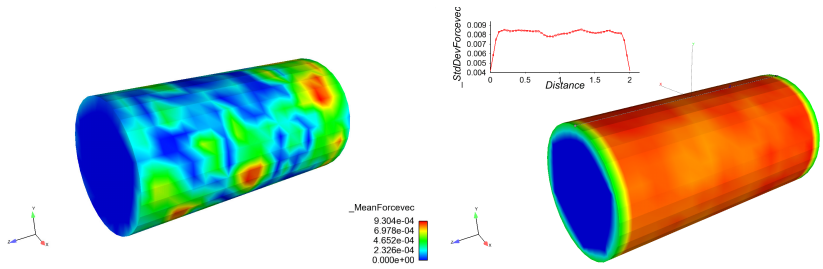

Figure 22-44. – Variation of Mean and Standard Deviation of Force Magnitude on the Surface

[15] A simple way to estimate F_o is to run a very short transient analysis after having converted the random pressure load to a constant unit pressure.

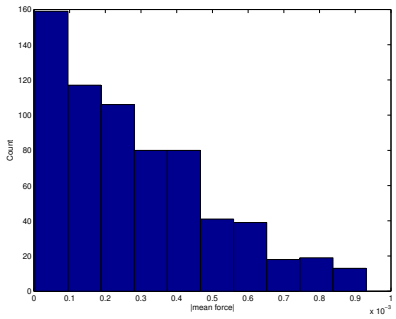

Figure 22-45. – Distribution of Mean Forces on Surface

22.3.2. Variance of Nodal Force

The standard deviation, which is the square root of the variance, is also available as an output from the analysis, and may be plotted on the structure using standard visualization tools. See Figure 22-44.

Again, the standard deviation is a statistical quantity, which is only meaningful for large numbers of samples. In the limit of large N, the standard deviation should approach F_o, as defined above, provided that the correlation function is 1 at time 0.

The plots show a value of $F_{std} \approx 0.0085$ which is under the expected value of 0.0098. Because the averaging process tends to round out the correlation function, the measured values of the standard deviation are typically somewhat less than F_o. The autocorrelation function analysis of the following section should make this more clear.

22.3.3. Temporal Nodal Force Autocorrelation

If we examine the statistics of a the loading on a single node, we should recover the temporal correlation that we initially input. Figure 22-46 shows the correlation function extracted from the raw time response data. The correlation function may be computed as,

$$f_c(n) = \frac{1}{F_o^2} \sum_i w_i w_{i-n}.$$

Where w_i is the force on a node at time t_i. This data can only be obtained using MATLAB or another external tool, i.e. it is not available as part of the statistical output. In MATLAB we get this with, `C = xcorr(f1,f1)`, where `f1` is the force time history on a node of the surface. We recover a correlation that is *similar* to the original triangle correlation in the input. Because of interpolation and finite sample length, we do not expect the same curves precisely.

The curves of Figure 22-46 should be considered "good enough" in a statistical sense. A temporal interpolation from multiples of `Delta_T` to the integration time step is being performed, which smooths the values of the correlation.

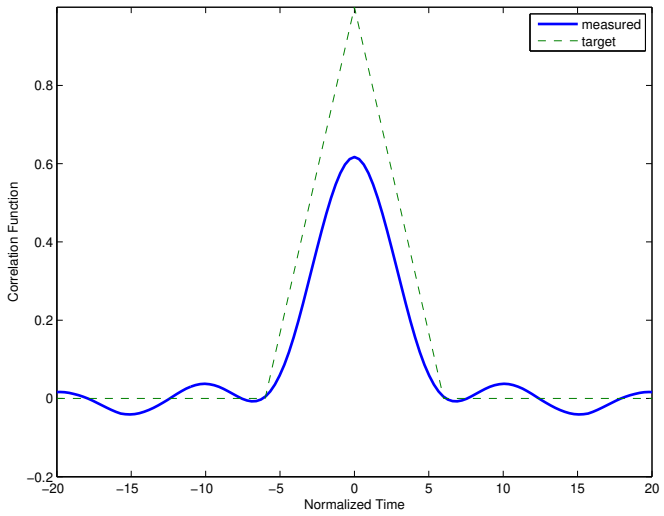

Figure 22-46. – Nodal Force Autocorrelation

22.3.4. Spatial Cross Correlation

The previous section discussed the autocorrelation function, i.e. the temporal correlation of signals on the same spatial location. We can also examine the cross correlation functions. We will only evaluate the functions at the peak.

This is more difficult. We use the MATLAB "find" method to get the indices of the nodes with $x = -0.5$, and $y = 0$. We loop through these nodes, and compute the "xcorr" function between the node at the center and the other nodes. The peak value of this solution is then plotted versus the distance in Figure 22-47.

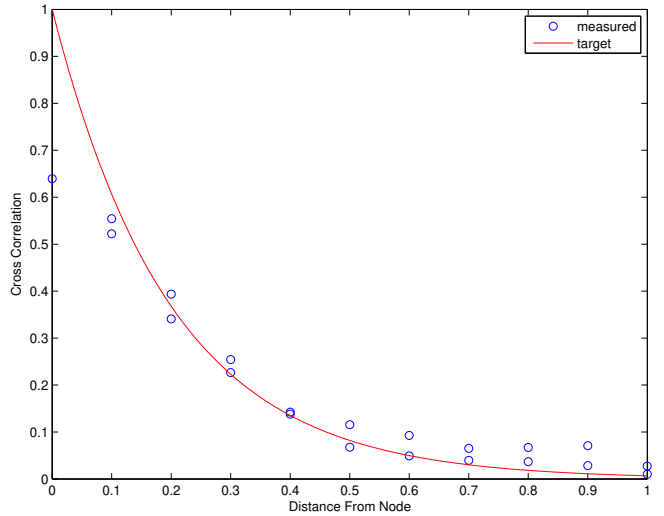

Figure 22-47. – Nodal Force Spatial Cross Correlation

There are obvious differences between the measured loads and the target. The correlations for close distances are lower. This is understood to be generated by the temporal

interpolation function. At large distances, the cross correlations never go to zero because of the finite length of the sample.

22.4. Random Pressure Comments

Effective Area

Random pressures are computed as force loads using a consistent pressure calculation. Pressures at the nodes are spread through the element shape functions to result in nodal forces. For a uniform mesh, this is similar to lumping the pressures from a fixed area onto the nodes with $F = P \cdot Area$. In Figure 22-48 an element based mesh is shown along a corresponding effective area for the nodes. For a uniform quadrilateral mesh like the example above, the nodal effective area is the same as the area of an element face.

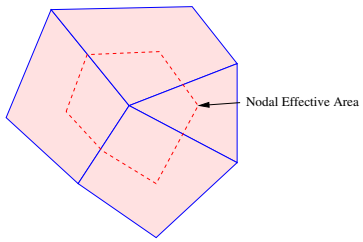

Figure 22-48. – Nodal Effective Area

Temporal Interpolation

To improve performance, the random pressure loading procedure computes random pressures at multiples of "Delta_T" and then interpolates to integration time steps. A piecewise linear interpolation introduces unacceptable errors; we use a $sin(x)/x$ type interpolation.

Interpolation can be avoided by choosing the integration and sampling times to be equal. In no case should the integration time be larger than the sampling time.

Singularities

To compute the proper temporal and spatial correlations for the forces, we need to perform a Cholesky factorization of the correlation matrix. This factor will fail if the matrix is singular. Remember that the correlation matrix that we factor is a tensor product of temporal and spatial components, $C = C_{spatial} \otimes C_{temporal}$. If either component is singular, the matrix C is singular. Several common issues can cause singularity of this matrix.

1. Taking NTimes too large or too small. For small Delta_T, NTimes must be large enough to sample the time correlation function. However, studies show that the condition number of the matrix grows exponentially with NTimes. The target value is 5. Values above 20 are not recommended; $C_{temporal}$ is numerically singular.
2. Spatial degeneracy, leading to $C_{spatial} = 0$. We have only one means of entering the spatial correlation parameters, viz. the `correlation_length` variables pair. If either of these quantities are so large that δ/correlation_length is very close to zero (with δ representing the distance from one node to another on the mesh), then the spatial portion of the matrix becomes singular. Effectively, these locations are no longer independent, but must apply the same load vector.
3. When Delta_T = 1/cutoff_freq and the default *sinc* function is used for a correlation function a $C_{temporal}$ singularity can be generated. [16] This is because we are now evaluating the correlation function at multiples of π, where it is always zero.

Time Step

The integration time step specified in the SOLUTION section must *always* be less than or equal to Delta_T.

Sinc Function

The *sinc* function defined as $\sin(x)/x$ is important in at least two places in the code. First, it is the only function available for the temporal interpolation function. Second, by default, we use the *sinc* function as the correlation function. In most cases, this use of the function should probably be replaced by another function. We use it as the default because it represents the Fourier transform of a flat PSD, which is the simplest loading.

22.5. Memory, Performance, Parallel and Anything Else of Interest

The matrices generated for these operations are all dense. The size of the matrix is $d \times d$ where, $d = n_{spatial} \cdot n_{temporal}$. Here $n_{spatial}$ is the number of points in the surface and $n_{temporal}$ =`NTimes`. Because memory requirements grow as the square of these variables, it is important to manage these carefully. Practically, models up to $d = 10^5$ are possible in parallel, but they take a lot of time.

The operation count for Cholesky factorization of a dense matrix is of order d^3. Thus, the computational cost increases much faster than model size. **The parallel solutions of the Cholesky system are not scalable**. In a scalable problem, doubling the size of the problem, and also doubling the number of processors should not change the solution time. The sparse linear solvers for FE solution are scalable, but the Cholesky factorization required to compute random pressure loads is not.

[16] In this example, we intentionally use the triangular function both for simplicity, and to avoid this problem.

The parallel Cholesky solution of dense matrices uses the ScaLAPACK library. The decomposition for this solve is completely different from the FEM decomposition, and is computed internally without user intervention. The user input for the parallel solution is exactly the same as the serial input. However, at this time, parallel solutions are limited to platforms built under the Intel compiler with MKL libraries. The solution will fail on other platforms.

23. Lighthill Tensor Loading

In this section we provide the steps for applying the Lighthill tensor as a load in a **Sierra/SD** acoustics simulation. The Lighthill tensor captures the noise generated by unsteady convection in flow in a fluids simulation. In this work, we use the Sierra/TF incompressible thermal fluids code Fuego to simulate a small chamber, shown in Figure 23-49, that undergoes a sinusoidal pumping motion in the x-direction. The air moving in and out of the chamber produces turbulence that is captured by the Lighthill tensor computed during the Fuego simulation. The divergence of the Lighthill tensor is handed off to **Sierra/SD** and is used as an acoustic source term for far-field acoustic noise modeling in the larger semi-circular domain shown in Figure 23-50.

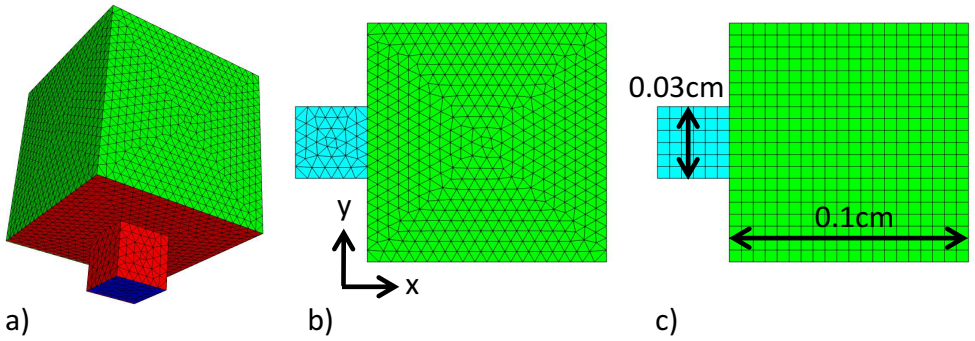

Figure 23-49. – a) Fuego mesh of fluids domain where sideset 2 (green) is absorbing, sideset 4 (blue) undergoes the pumping motion, and all other sides shown in red are fixed. Sideset 2 shown in green will be tied to the larger **Sierra/SD** domain shown in Figure 23-50. b) Fuego mesh shown on z-plane. c) Fuego interpolation mesh for output of the divergence of the Lighthill Tensor. Domain dimensions are also shown in c).

These simulations are part of the Sierra test suite and provide regression testing for both the **Sierra/SD** and Fuego parts of Lighthill noise modeling. Lighthill loading has also been verified in **Sierra/SD** for a 1-D waveguide with documentation provided in the **Sierra/SD** verification manual. The input for this example is provided in Appendix A.14.

Producing the Lighthill load and applying it in **Sierra/SD** is a 5 step process. The initial steps produce the divergence of the Lighthill tensor from a Fuego CFD simulation and are found in the test repo:
`fuego_rtest/fuego/mesh_deformation_file/`
Questions about these initial steps should be directed to the Sierra Thermal Fluids team.

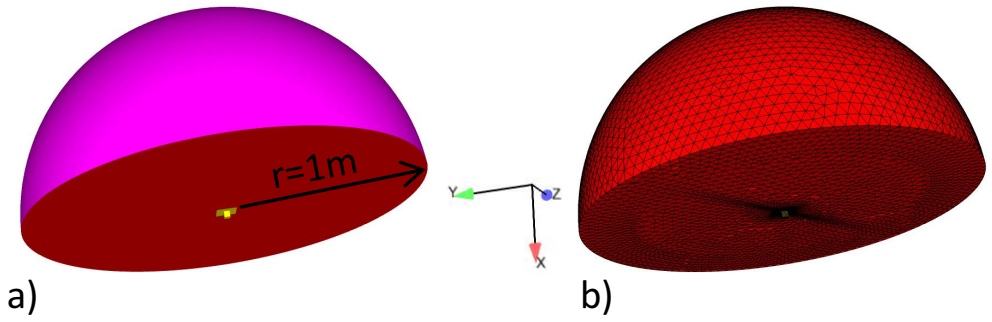

Figure 23-50. – a) **Sierra/SD** domain for acoustic noise propagation. The yellow block is the Fuego output domain containing divT and the red block is the additional domain for the **Sierra/SD** simulation. The pink sideset will interface with infinite elements. b) **Sierra/SD** tetrahedral mesh coarsened out from the Fuego mesh

The final steps involve preparing the Fuego output for use in **Sierra/SD** and then running the **Sierra/SD** simulation and are found in the test repo:
`Salinas_rtest/regression_tests/acoustics/lighthillFuegoSalinasHowTo`
Questions about the final steps should be directed to the **Sierra/SD** team.

23.1. Mesh Deformation For Fuego

This section describes the process of producing a deformation field used to drive the Fuego simulation. Questions about Aria should be directed to the Sierra/TF team. The files referenced in this section are found in the directory:
`fuego_rtest/fuego/mesh_deformation_file/`

In this example, Aria is used to produce the displacement field using the input file `generate_displ.i`. This file produces sinusoidal displacement in the x-direction on sideset 4, shown in blue in Figure 23-49a. The displacement of sideset 4 is given by

$$x(t) = a\sin(\pi\omega t) \tag{23.1}$$

where the ω=1000Hz, a=0.02m, and displacement in the y- and z-direction is fixed. The simulation is terminated at t=6e-3s. The Aria simulation is executed with the command:
`aria -i generate_displ.i`
which produces the file `displacements.e` that is used as input for Fuego. The Aria simulation is small and is run in serial.

23.2. Fuego Simulation

This section describes the process of running Fuego to produce the divergence of the Lighthill Tensor. Questions about Fuego should be directed to the Sierra/TF team. The files referenced in this section are found in the directory:
`fuego_rtest/fuego/mesh_deformation_file/`

The Fuego input file is `fluid.i` and is executed with the command:
`mpirun -np 8 fuego -i fluid.i`
The Fuego simulation is terminated at t=3e-3s. The Fuego simulation is discretized by the tetrahedral mesh shown in Figure 23-49b. The Fuego simulations writes the divergence of the Lighthill tensor out to the coarser hexahedral mesh shown in Figure 23-49c as nodal data. This data is written to `acoustic.e.8.[0-7]` and provides the loading for the **Sierra/SD** simulation.

23.3. Processing Fuego output for Sierra/SD

This section describes the steps required to run a **Sierra/SD** simulation using the Fuego output. Questions about this section should be directed to the **Sierra/SD** team. All steps in this section and the next are executed by the regression test located at:
`Salinas_rtest/regression_tests/acoustics/lighthillFuegoSalinasHowTo/`
The test file reproducing this work flow is `lighthill_howto.test`.

The first step is to join the partitioned Fuego files back together using the epu Seacas tool:
`epu -auto acoustic.e.8.0`
The above Fuego simulation writes the divergence of the Lighthill tensor out as nodal data with the variable names: `divT_x`, `divT_y`, `divT_z`. The Fuego domain is much smaller than than the **Sierra/SD** domain. If these two domains were joined together into a single **Exodus** file, nodal data of divT=0 would be created on the larger **Sierra/SD** domain. To circumvent this unnecessary storage of divT data on the **Sierra/SD** mesh, we convert the Fuego divT data to nodeset data using the ejoin Seacas tool:
`ejoin -output acoustic_ns.exo -convert_nodal_to_nodesets all acoustic.e`
which produces the output file `acoustic_ns.exo`.

23.4. Mesh for Sierra/SD

The **Sierra/SD** simulation will use the Fuego divT data as a source term to model noise propagation in a larger domain. For this example we join the smaller Fuego mesh containing the interpolated divT data to a larger semi-circular domain, see Figure 23-50a. A cubit journal file for creating the semi-circular mesh contained in `halfSphere.jou`. This mesh must contain sidesets (sideset 5 in the cubit journal file) that will be tied to sideset 2 in the Fuego output mesh, shown in green if Figure 23-49a. This mesh also contains sideset 6 on the exterior of the semicircular domain which will be used for applying absorbing boundary conditions via infinite elements. The two separate meshes are joined together with the ejoin Seacas tool:
`ejoin -output acoustic_distFactors.exo halfSphere.exo acoustic_ns.exo`
This produces the full meshed domain shown in Figure 23-50b for the **Sierra/SD** simulation. This mesh is then decomposed into four domains using `stk_balance`:
`mpirun -np 4 stk_balance acoustic_distFactors.exo temp1`

23.5. Sierra/SD *simulation*

This **Sierra/SD** simulation will be described in this section. Lighthill loading causes **Sierra/SD** to use the acceleration potential form of the acoustic equation. The **Sierra/SD** input file is included in Section 14. The **Sierra/SD** simulation is terminated after t=0.06s, which is twice as long as the Fuego simulation. For the final 0.03s of the simulation there will not be any available Fuego produced divT data to be read in for Lighthill Loading. For this case, the final divT data read in at t=0.03s will be applied for the remainder of the simulation, which produces a warning to this effect.

Some Lighthill specific portions of the attached **Sierra/SD** input file are:

1) The Lighthill loading is applied as a function load the LOADS section with the Function described in FUNCTION 1. Lighthill loading is described in *User's Manual* and the verification manual.

2) Tied data ties together the Fuego and Sierra-SD domains. Sidesets must be defined on these surfaces when they are created in Cubit. It is difficult to add a sideset to a mesh after it contains nodal data, i.e. The sidesets needed to tie the meshes together must be defined on the mesh used for Fuego output before the Fuego simulation is run.

3) Infinite elements are used on sideset 6 to absorb the pressure waves.

24. Tied Joints

The `Tied Joint` provides an interface to the whole joint models. Multiple connection methods are supported, including weighted constraint equations.

Separate shear and normal forces are supported. The separation also reduces requirements on the constraints. The whole surface is no longer required to have 6 rigid body modes. The normal tied interface keeps surfaces together. This relaxes the requirements for shear constraints. The Tied Joint permits constraints that look more like a collection of trusses, not a collection of beams.

Rotational DOFs are necessary for the structure to move as a rigid body. However, the adjacent elements may have no rotational stiffness. This introduces singularities. Avoiding the rotational DOFs is important.

Normal direction constraints are tied surfaces. Shear direction constraints are a truss network. For curved surfaces, constraints may be inconsistent.

24.1. Lap joint

A lap joint contains regions of "welded" contact, microslip, and macroslip as shown in Figure 24-51. An elastic spring approximates normal forces. Tied surfaces approximate shear behavior of the "welded" region. The macroslip region is free. The region of microslip depends on the loading. Microslip introduces loss into the structure. This region is well approximated by an Iwan element.

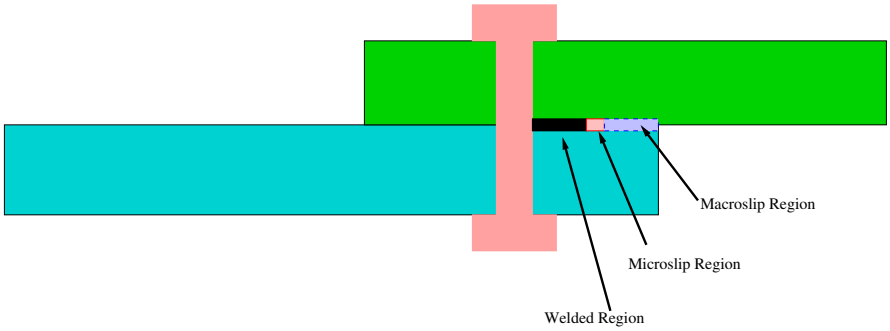

Figure 24-51. – Lap Joint with Contact Regions. The physics of bolted lap joints is complex. Tied Joints use a combination of constraints, springs and optionally Iwan elements to generate a reduced order model of the structure.

Without a Tied Joint, this lap joint can be modeled using a whole joint model. Each of the contact surfaces is rigidized (using a rigidset). A Joint2G connects the surfaces. The mesh is represented in Figure 24-52. Figure 24-53 illustrates the conventional means of connecting this structure. This method reduces all the behavior of the joint to a single Joint2G element. That element must be included as part of the mesh. Because the surfaces are allowed to translate and rotate independently, interpenetration can occur. Nevertheless, the method is effective in representing the energy loss that occurs in this structure.

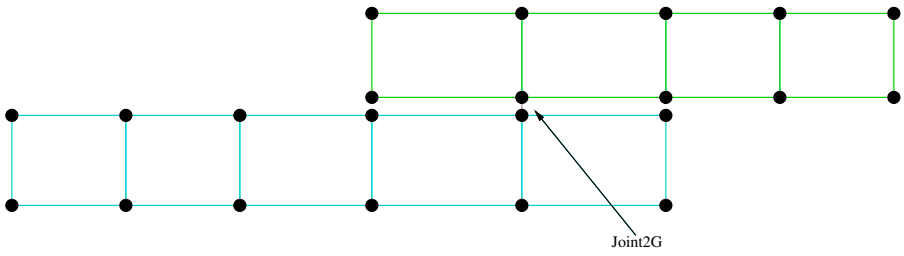

Figure 24-52. – Lap Joint Finite Element Mesh. The physical lap joint is represented by a reduced order model which uses disconnected meshes of the top and bottom material. These are shown separated in the cartoon but may have overlapping nodes. In a conventional connection the Joint2G which represents the bolt must be explicitly meshed. The Tied Joint approach generates that element internally.

The input included in Figure 24-54 represents the same physics. The normal definition is "none" because the normal stiffness is part of the Joint2G structure. The shear side definition is "rigid" corresponding to a rigid set definition on each of the surfaces. No mesh of block 3 is required.

```
Rigidset
        sideset 1
end
Rigidset
        sideset 2
end

Block 3
        Joint2G
                Kz = Elastic 1e6
                Kx = Iwan 1
                Ky = Iwan 1
                Krx = Elastic 1e9
                Kry = Elastic 1e9
                Krz = Elastic 1e9
END
```

Figure 24-53. – Conventional Input for Whole Lap Model

```
Tied Joint
        Normal Definition = none
                surface 1,2
        Shear Definition
                side = rigid
                connect to Block 3
end

Block 3
        Joint2G
                Kz = Elastic 1e6
                Kx = Iwan 1
                Ky = Iwan 1
                Krx = Elastic 1e9
                Kry = Elastic 1e9
                Krz = Elastic 1e9
END
```

Figure 24-54. – Tied Joint Input for Whole Lap Model

24.2. Joint with Slip

The whole joint model of section 24.1 can be modified to prevent penetration of the two surfaces. The models are shown in Figures 24-55 and 24-56 for the conventional and Tied Joints.

Sliding contact or *slip* keeps two surfaces in contact with no resistance to transverse motion. Because the sliding contact constrains the normal behavior, the Joint2G parameters for that direction are irrelevant. Because the surfaces are flexible, properly constraining the transverse motion of the connection nodes is challenging. The constrain method is specified using the `side`. The `Rrod` and `average` methods are available. The example uses the `Rrod` approach.

```
Rigidrod
        sideset 1
end
Rigidrod
        sideset 2
end

Block 3
        Joint2G
                Kx = Iwan 1
                Ky = Iwan 1
                Krz = Elastic 1e9
        // not needed
                Krx = Elastic 1e9
                Kry = Elastic 1e9
                Kz = Elastic 1e6
END
Tied Data
        name = 'block_3_tj'
        surface 1,2
        transverse slip
end
```

Figure 24-55. – Conventional Input for Whole Lap Model with Sliding Contact

```
Tied Joint
        Normal Definition = slip
                surface 1,2
        Shear Definition
                side = Rrod
                connect to Block 3
end

Block 3
        Joint2G
                Kx = Iwan 1
                Ky = Iwan 1
                Krz = Elastic 1e9
        // not needed
                Kz = Elastic 1e6
                Krx = Elastic 1e9
                Kry = Elastic 1e9
END
```

Figure 24-56. – Tied Joint Input for Whole Lap Model with Sliding Contact

1. INPUT FILES FOR HOWTO PROBLEMS

1. Input. static.inp

Refer to Section 1 for details of the test.

```
SOLUTION
  title 'static run of a test fixture model'
  statics
END

PARAMETERS
   wtmass=0.00259
END

FILE
   geometry_file 'FILEPATH'
END

LOADS
  nodeset 2
    force 1.0 0.0 0.0
    scale 200.0
END

BOUNDARY
  nodeset 1
    fixed
END

OUTPUTS
  displacement
  stress
END

ECHO
  mass block
END

BLOCK 1
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
END

MATERIAL 1
```

```
   // fixture - Ti
   density=0.16
   E=1.6e+07
   nu=0.3
END

GDSW
  solver_tol=1e-8
END
```

2. Input. eigen.inp

Refer to Section 1 for details of the test.

```
SOLUTION
  title 'eigen run of a test fixture model'
  eigen
    nmodes 12
END

PARAMETERS
   wtmass=0.00259
   eigen_norm=visualization
END

FILE
   geometry_file 'FILEPATH'
END

BOUNDARY
  nodeset 1
    fixed
END

OUTPUTS
  displacement
END

ECHO
  mass block
END

BLOCK 1
   // fixture
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
END

MATERIAL 1
   // fixture - Ti
   density=0.16
   E=1.6e+07
   nu=0.3
END

GDSW
  solver_tol 1.0e-10
END
```

3. Input. transient.inp

Refer to Section 1 for details of the test.

```
// direct transient run example

SOLUTION
  title 'direct transient run of a test fixture model'
  transient
    time_step 1.0e-4
    nsteps    100
END

PARAMETERS
 wtmass=0.00259
END

FILE
   geometry_file 'FILEPATH'
END

LOADS
  nodeset 1
    force 0.0 1.0 0.0
    scale 1.0e7
    function 1
END

HISTORY
  nodeset 33
  nodeset 148
  nodeset 270
  displacement
  acceleration
END

OUTPUTS
END

ECHO
  mass block
END

//  Block and material input

BLOCK 1
   // fixture
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
END

MATERIAL 1
   // fixture - Ti
   density=0.16
```

```
   E=1.6e+07
   nu=0.3
END

//  function defining a haversine pulse

FUNCTION 1
  name "1500g_03ms.fun"
  type LINEAR
  data  0.000000e+000   0.0000
  data  3.636364e-006   1.5100
  data  7.272727e-006   6.0339
  data  1.090909e-005   13.5535
  data  1.454545e-005   24.0385
  data  1.818182e-005   37.4467
  data  2.181818e-005   53.7241
  data  2.545455e-005   72.8051
  data  2.909091e-005   94.6130
  data  3.272727e-005   119.0599
  data  3.636364e-005   146.0473
  data  4.000000e-005   175.4667
  data  4.363636e-005   207.1995
  data  4.727273e-005   241.1179
  data  5.090909e-005   277.0855
  data  5.454545e-005   314.9573
  data  5.818182e-005   354.5809
  data  6.181818e-005   395.7967
  data  6.545455e-005   438.4387
  data  6.909091e-005   482.3353
  data  7.272727e-005   527.3097
  data  7.636364e-005   573.1808
  data  8.000000e-005   619.7639
  data  8.363636e-005   666.8714
  data  8.727273e-005   714.3136
  data  9.090909e-005   761.8995
  data  9.454545e-005   809.4375
  data  9.818182e-005   856.7361
  data  1.018182e-004   903.6050
  data  1.054545e-004   949.8554
  data  1.090909e-004   995.3010
  data  1.127273e-004   1039.7588
  data  1.163636e-004   1083.0500
  data  1.200000e-004   1125.0000
  data  1.236364e-004   1165.4400
  data  1.272727e-004   1204.2073
  data  1.309091e-004   1241.1456
  data  1.345455e-004   1276.1062
  data  1.381818e-004   1308.9483
  data  1.418182e-004   1339.5398
  data  1.454545e-004   1367.7574
  data  1.490909e-004   1393.4876
  data  1.527273e-004   1416.6266
  data  1.563636e-004   1437.0813
  data  1.600000e-004   1454.7695
  data  1.636364e-004   1469.6197
  data  1.672727e-004   1481.5723
  data  1.709091e-004   1490.5792
  data  1.745455e-004   1496.6039
  data  1.781818e-004   1499.6224
  data  1.818182e-004   1499.6224
  data  1.854545e-004   1496.6039
  data  1.890909e-004   1490.5792
  data  1.927273e-004   1481.5723
  data  1.963636e-004   1469.6197
  data  2.000000e-004   1454.7695
  data  2.036364e-004   1437.0813
  data  2.072727e-004   1416.6266
  data  2.109091e-004   1393.4876
```

```
  data  2.145455e-004   1367.7574
  data  2.181818e-004   1339.5398
  data  2.218182e-004   1308.9483
  data  2.254545e-004   1276.1062
  data  2.290909e-004   1241.1456
  data  2.327273e-004   1204.2073
  data  2.363636e-004   1165.4400
  data  2.400000e-004   1125.0000
  data  2.436364e-004   1083.0500
  data  2.472727e-004   1039.7588
  data  2.509091e-004   995.3010
  data  2.545455e-004   949.8554
  data  2.581818e-004   903.6050
  data  2.618182e-004   856.7361
  data  2.654545e-004   809.4375
  data  2.690909e-004   761.8995
  data  2.727273e-004   714.3136
  data  2.763636e-004   666.8714
  data  2.800000e-004   619.7639
  data  2.836364e-004   573.1808
  data  2.872727e-004   527.3097
  data  2.909091e-004   482.3353
  data  2.945455e-004   438.4387
  data  2.981818e-004   395.7967
  data  3.018182e-004   354.5809
  data  3.054545e-004   314.9573
  data  3.090909e-004   277.0855
  data  3.127273e-004   241.1179
  data  3.163636e-004   207.1995
  data  3.200000e-004   175.4667
  data  3.236364e-004   146.0473
  data  3.272727e-004   119.0599
  data  3.309091e-004   94.6130
  data  3.345455e-004   72.8051
  data  3.381818e-004   53.7241
  data  3.418182e-004   37.4467
  data  3.454545e-004   24.0385
  data  3.490909e-004   13.5535
  data  3.527273e-004   6.0339
  data  3.563636e-004   1.5100
  data  3.600000e-004   0.0000
END

GDSW
  solver_tol 1.0e-8
END
```

4. Input. modaltransient.inp

Refer to Section 1 for details of the test.

```
SOLUTION
  title 'modal transient run of a test fixture model'
  case eigen
    eigen
      nmodes 20
      shift -1e6
  case trans
    modaltransient
      time_step 1.0e-4
      nsteps    100
      load      1
END

PARAMETERS
 wtmass=0.00259
END

FILE
   geometry_file 'FILEPATH'
END

LOAD 1
  nodeset 1
    force 0.0 1.0 0.0
    scale 1.0e7
    function 1
END

HISTORY
  nodeset '33'
  nodeset '148'
  nodeset '270'
  disp
  velocity
  acceleration
END

OUTPUTS
END

ECHO
  mass block
END

//  Block and material input

BLOCK 1
   // fixture
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
```

```
END

MATERIAL 1
   // fixture - Ti
   density=0.16
   E=1.6e+07
   nu=0.3
END

//  function defining a haversine pulse

FUNCTION 1
  name "1500g_03ms.fun"
  type LINEAR
  data  0.000000e+000   0.0000
  data  3.636364e-006   1.5100
  data  7.272727e-006   6.0339
  data  1.090909e-005   13.5535
  data  1.454545e-005   24.0385
  data  1.818182e-005   37.4467
  data  2.181818e-005   53.7241
  data  2.545455e-005   72.8051
  data  2.909091e-005   94.6130
  data  3.272727e-005   119.0599
  data  3.636364e-005   146.0473
  data  4.000000e-005   175.4667
  data  4.363636e-005   207.1995
  data  4.727273e-005   241.1179
  data  5.090909e-005   277.0855
  data  5.454545e-005   314.9573
  data  5.818182e-005   354.5809
  data  6.181818e-005   395.7967
  data  6.545455e-005   438.4387
  data  6.909091e-005   482.3353
  data  7.272727e-005   527.3097
  data  7.636364e-005   573.1808
  data  8.000000e-005   619.7639
  data  8.363636e-005   666.8714
  data  8.727273e-005   714.3136
  data  9.090909e-005   761.8995
  data  9.454545e-005   809.4375
  data  9.818182e-005   856.7361
  data  1.018182e-004   903.6050
  data  1.054545e-004   949.8554
  data  1.090909e-004   995.3010
  data  1.127273e-004   1039.7588
  data  1.163636e-004   1083.0500
  data  1.200000e-004   1125.0000
  data  1.236364e-004   1165.4400
  data  1.272727e-004   1204.2073
  data  1.309091e-004   1241.1456
  data  1.345455e-004   1276.1062
  data  1.381818e-004   1308.9483
  data  1.418182e-004   1339.5398
  data  1.454545e-004   1367.7574
  data  1.490909e-004   1393.4876
  data  1.527273e-004   1416.6266
  data  1.563636e-004   1437.0813
  data  1.600000e-004   1454.7695
  data  1.636364e-004   1469.6197
  data  1.672727e-004   1481.5723
  data  1.709091e-004   1490.5792
  data  1.745455e-004   1496.6039
  data  1.781818e-004   1499.6224
  data  1.818182e-004   1499.6224
  data  1.854545e-004   1496.6039
  data  1.890909e-004   1490.5792
  data  1.927273e-004   1481.5723
```

```
  data  1.963636e-004   1469.6197
  data  2.000000e-004   1454.7695
  data  2.036364e-004   1437.0813
  data  2.072727e-004   1416.6266
  data  2.109091e-004   1393.4876
  data  2.145455e-004   1367.7574
  data  2.181818e-004   1339.5398
  data  2.218182e-004   1308.9483
  data  2.254545e-004   1276.1062
  data  2.290909e-004   1241.1456
  data  2.327273e-004   1204.2073
  data  2.363636e-004   1165.4400
  data  2.400000e-004   1125.0000
  data  2.436364e-004   1083.0500
  data  2.472727e-004   1039.7588
  data  2.509091e-004   995.3010
  data  2.545455e-004   949.8554
  data  2.581818e-004   903.6050
  data  2.618182e-004   856.7361
  data  2.654545e-004   809.4375
  data  2.690909e-004   761.8995
  data  2.727273e-004   714.3136
  data  2.763636e-004   666.8714
  data  2.800000e-004   619.7639
  data  2.836364e-004   573.1808
  data  2.872727e-004   527.3097
  data  2.909091e-004   482.3353
  data  2.945455e-004   438.4387
  data  2.981818e-004   395.7967
  data  3.018182e-004   354.5809
  data  3.054545e-004   314.9573
  data  3.090909e-004   277.0855
  data  3.127273e-004   241.1179
  data  3.163636e-004   207.1995
  data  3.200000e-004   175.4667
  data  3.236364e-004   146.0473
  data  3.272727e-004   119.0599
  data  3.309091e-004   94.6130
  data  3.345455e-004   72.8051
  data  3.381818e-004   53.7241
  data  3.418182e-004   37.4467
  data  3.454545e-004   24.0385
  data  3.490909e-004   13.5535
  data  3.527273e-004   6.0339
  data  3.563636e-004   1.5100
  data  3.600000e-004   0.0000
END

GDSW
  solver_tol=1e-12
END
```

5. Input. modalfrf.inp

Refer to Section 1 for details of the test.

```
SOLUTION
  title 'modal frf run of a test fixture model'
  case eigen
    eigen
      nmodes 20
      shift -1e6
    restart auto
  case frf
    restart auto
    modalfrf
      load      1
END

PARAMETERS
   wtmass=0.00259
END

FILE
   geometry_file 'FILEPATH'
END

LOAD 1
  nodeset 1
    force 0.0 1.0 0.0
    scale 1.0e7
    function 1
END

FUNCTION 1
  type linear
  data 0.0    1.0
  data 1.0e8  1.0
END

FREQUENCY
  nodeset 270
  acceleration
  freq_min 100
  freq_max 8000
  freq_step 200
END

DAMPING
  gamma 0.02
END

HISTORY
  nodeset '33'
  nodeset '148'
  nodeset '270'
  acceleration
END

OUTPUTS
END

ECHO
  mass block
END

//  Block and material input
```

```
BLOCK 1
   // fixture
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
END

MATERIAL 1
   // fixture - Ti
   density=0.16
   E=1.6e+07
   nu=0.3
END


GDSW
  solver_tol=1e-10
END
```

6. Input. randomvibration.inp

Refer to Section 1 for details of the test.

```
SOLUTION
  title 'modal random vibration run of a test fixture model'
  case eigen
    eigen
      nmodes 20
      shift -1e6
  case rvib
    modalranvib
    lfcutoff -10
END

PARAMETERS
 wtmass=0.00259008
END

FILE
   geometry_file 'FILEPATH'
END

LOADS
END

FREQUENCY
  nodeset 1,270
  acceleration
  freq_min 100
  freq_max 8000
  freq_step 200
END

DAMPING
  gamma 0.02
END

// scale = concentrated mass * wtmass
RANLOADS
  matrix 1
  load 1
    nodeset 1
    force 1.0 0.0 0.0
    scale 2.59e+4
  load 2
    nodeset 1
    force 0.0 1.0 0.0
    scale 2.59e+4
  load 3
    nodeset 1
    force 0.0 0.0 1.0
    scale 2.59e+4
END

MATRIX-FUNCTION 1
  name 'psd input'
  symmetry hermitian
  dimension 3x3
  data 1,1
    real function 1
  data 2,2
    real function 1
  data 3,3
    real function 1
END
```

```
FUNCTION 1
  Name = "psd"
  type = linear
  data 100.0   0.
  data 300.0   0.001
  data 500.0   0.01
  data 700.0   0.1
  data 7500.0  0.1
  data 7700.0  0.01
  data 7900.0   0.001
  data 8100.0   0.
END

OUTPUTS
  displacement
  acceleration
  vrms
END

ECHO
  mass block
END

//  Block and material input

BLOCK 1
   // fixture
   material 1
END

BLOCK 2
  rbar
END

BLOCK 3
  ConMass
    Mass 1.0e7
    Ixx  1.0e8
    Iyy  1.0e8
    Izz  1.0e8
    Offset= 0.0 0.0 0.0
END

MATERIAL 1
   // fixture - Ti
   density=0.16
   E=1.6e+07
   nu=0.3
END


GDSW
  solver_tol=1e-8
END
```

7. Random Vibration Input. Vran1.inp

Refer to Section 15 for details of the test.

```
//salinas input created using pat2exo from patran file 'vtube.out'
SOLUTION
  case eig
   eigen nmodes=9
   shift=-1e5
  case rms
   modalranvib
   truncationMethod = displacement
   keepmodes=3   // force modal truncation
  title 'vtube test of random vibration'
END

RANLOADS
   matrix=1
   load=1
     nodeset 12
force=0 1 0
scale 1.00e3  // needed to convert to g
// loads input in lbs. The PSD is in g^2/Hz.
// F = accel * mass
//   = accel * (scale_factor)
//   = accel * ((1000*.00259)*384.6)
END

Frequency
   freq_step=100
   freq_min=300
   freq_max=1e4
   BLOCK=all
END

MATRIX-FUNCTION 1
   Name=input_psd
   symmetry=symmetric
   dimension=1x1
   data 1,1
     real function 1
END

FUNCTION 1
   Name='psd'
   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

DAMPING
  gamma=0.01
END

PARAMETERS
 wtmass=0.00259
END

FILE
   geometry_file 'FILEPATH'
//   geometry_file 'vtube.exo'
```

```
END

BOUNDARY
  nodeset 124
  rotx=0 roty=0 rotz=0 x=0 z=0
//    fixed
//  nodeset 25
//    fixed
//  nodeset 26
//    fixed
END

LOADS
//   nodeset 3
//      force = 1.0 0. 0.
//      scale = 1000.
//      function = 2
// note... no sidesets in file.
//   sideset 7
//      pressure 15.0
//   body
//      gravity
//      0.0 1.0 0
//      scale -32.2
END

OUTPUTS
vrms
END

ECHO
   vrms
END


GDSW
  solver_tol 1e-9
END

BLOCK 101
   material 101
  quadt
   thickness= 0.200000003E+00
   // patran/exo type 'QUAD'/QUAD. Number nodes 4
END

BLOCK 102
//   material 0
  ConMass
    Mass=1000
    Ixx =0
    Ixy =0
    Iyy =0
    Ixz =0
    Iyz =0
    Izz =0
    Offset= 0 0 0
   // patran/exo type 'BEAM'/BEAM. Number nodes 2
END

Block 10001
    RBAR  // RBE type elements
    // # links 16
END
Block 1000
   material=1000
   beam2
   area=1
```

```
   i1=.1
   i2=.1
   j=.2
   orientation=1 0 .10
end

MATERIAL 101
   // material type 'Iso'
   density=0.1
   Isotropic
   E=1e+07
   nu=0.35
END

MATERIAL 1000
   // material type 'Iso'
   density=0.1e-5
   Isotropic
   E=1e+09
   nu=0.35
END
```

8. Infinite Element Input

Refer to Section 20 for details of the test.

```
SOLUTION
  transient
  time_step 1.0e-2
  nsteps 500
END

FILE
  geometry_file 'FILEPATH'
//  geometry_file 'infinite_100elem.exo'
END

LINESAMPLE
  samples per line 2
  endpoint 0 0 500 0 0 500.001
  format exodus
END

OUTPUTS
apressure
END

ECHO
END

BOUNDARY
  sideset 1
    infinite_element
    use block 111
END

BLOCK 1
        material "air"
END

Block 111
    infinite_element
       radial_poly legendre
       order 3
       neglect_mass yes
       ellipsoid_dimensions 200 200 200
END

MATERIAL "air"
        density 1.293
        acoustic
        c0 332.0
END


FUNCTION 3
  type analytic
  evaluate expression = "sin(2 * pi * t)"
END

LOADS
  sideset 2
  acoustic_accel = -1.0
  function = 3
END

GDSW
   solver_tol    1.0e-9
```

```
END
```

9. Random Pressure Input

Refer to Section 22 for details of the test.

```
SOLUTION
   transient
   time_step 1.0e-4
   nsteps 20
END

FILE
   geometry_file 'cylinder_random.exo'
END

LOADS
           sideset 1
                randompressure
                Delta_T=1e-3
                cutoff_freq = 4.999999994286667e+02
                correlation_length_z 0.5
                correlation_length_r = 0.2
                ntimes  = 5
correlation_function = 1
coordinate 1
END

Begin rectangular coordinate system 1
    origin   = 0 0 0
    z point  = 1 0 0
    xz point = 1 0 1
end


BOUNDARY
END

function 1
    type linear
    data -0.001909859319285 0
    data  0 1
    data  0.001909859319285 0
end

OUTPUTS
  statistics
  force
  pressure // DON'T DELETE
END

PARAMETERS
    RandomNumberGenerator = test
END

ECHO
END



GDSW
        LO_option 0
        krylov_method=1
        max_iter=2000
        solver_tol=1e-4
        orthog=4000
        prt_summary=1
        prt_debug=1
```

```
        overlap = 20
        prt_timing yes
        coarse_option 0
END


BLOCK 1
        material 1
END

MATERIAL 1
E 72e9 //(N/m^2)
        nu .33
        density 2700 //(kg/m^3)
END
```

10. Geometric Rigid Body Mode Input

Refer to Section 13 for details of the test.

```
SOLUTION
  case out
     geometric_rigid_body_modes
  case flexibleModes
     eigen
     nmodes 10
END

FILE
//   geometry_file 'simpleTied.exo'
   geometry_file FILEPATH
END

BOUNDARY
END

PARAMETERS
   num_rigid_mode 6
   RbmTolerance 2.e-6
// Interestingly this is not the tolerance that gdsw uses.
 wtmass=0.00259
END

OUTPUTS
   disp
END

ECHO
  mass block
END

LOADS
   sideset 3
     traction 1 1 1
     scale = 1.0
     function 1
END

DAMPING
  beta 2.0e-6
END

TIED JOINT
  normal definition = slip
  surface 1,2
    search tolerance 1.0e-3
  side = free
  connect to block 3
END

BLOCK 1
   material 1
   nonlinear=no
END

BLOCK 2
   material 1
   nonlinear=no
END

BLOCK 3
 coordinate 2
```

```
 joint2g
 kx = iwan 1
 ky = iwan 1
 krz = elastic 1.0e9
END

MATERIAL 1
   density 0.3
   E = 3.0e7
   nu = 0.3
END

PROPERTY 1
    chi -.82
    phi_max = 1.75e-4
    R = 5.5050e+6
    S = 2.1097e+6
END

Begin rectangular coordinate system 2
    origin   = 0 -3.83232e-2 -5.96407
    z point  = 1.0 -3.83232e-2 -5.96407
    xz point = 1.0 0.4616768 -6.46407
end



GDSW
  max_numterm_C1 500
  overlap 2
  krylov_method 1
  prt_constraint 1
END
```

11. Wet Modes Input

Refer to Section 11 for details of the test.

```
// Sierra/SD input for Wet Modes Calculation of a Submerged Cylinder
// by Nicholas Reynolds, Code 6640, NSWCCD
// 22 FEB 2016

// ------------------------------------------------------------------------

SOLUTION
  title=' Acoustic analysis'
  case rigid
      geometric_rigid_body_modes
  case flex
  eigen
    nmodes 20
    fluidloading=yes
END

GDSW
  solver_tol 1.0e-6
  krylov_method 1
  overlap 2
END

// ------------------------------------------------------------------------

FILE
  geometry_file FILEPATH
END

// ------------------------------------------------------------------------
LOADS
END

PARAMETERS
num_rigid_mode 6
END

BOUNDARY
  sideset 102 // outer acoustic surface
p=0
//    infinite_element
//    order = 8
//    source_origin = 96 0 0
//    ellipsoid_dimensions 136.0 54.0 54.0
//    neglect_mass = no
  sideset 103 // free surface
    slosh = 2.59e-3 /// 1/(32.2*12 in/s/s)
END

TIED DATA
  surface 101, 1
  search tolerance = 2
END

// ------------------------------------------------------------------------

OUTPUTS
  disp
END

ECHO
  mass
  input
```

```
END

// ------------------------------------------------------------------------

MATERIAL steel
  e = 3.0e7
  density = 7.324e-4
  nu = 0.3
END

MATERIAL fluid
  acoustic
  density 3.46822e-006 // pci
  c0 22878 // sound speed
END

// ------------------------------------------------------------------------

BLOCK 1
  material = steel
  thickness = 1.3644
  nquad
END

BLOCK 2
  material = steel
  thickness = 1.3644
  nquad
END

BLOCK 101
  material = fluid
//tet4
END

// ------------------------------------------------------------------------
```

12. CBR Input

Refer to Section 8 for details of the test.

```
SOLUTION
      case eig1  // compute the full system. floating.
        eigen nmodes=10 shift=-1e6
      case cbr   // reduce the model
        cbr
        shift=0.
        nmodes 4
        title 'CBR example for "How To" document'
END

cbmodel
   nodeset=nodelist_3
   format=mfile
   file=cbr.m
   globalsolution
end

history
  nodeset 1:2
  disp
end

FILE
   geometry_file 'cbr.exo'
END

BOUNDARY
// free/free system
END

OUTPUTS
  disp
END

ECHO
END

BLOCK 1
        material 2
END

MATERIAL 2
        E 30e6
        nu .3
        density 0.288
END
```

13. Acoustic Scattering Input

Refer to Section 21 for details of the test.

```
SOLUTION
    case out
    transient
          time_step 1.66666666667e-06
          nsteps 1000
          nskip = 1
          load 10
          scattering
    title 'scattering'
END

FILE
     geometry_file 'FILEPATH'
END

Parameters
End



History
  velocity
  nodeset 1
  nodeset 2
End

BOUNDARY
  sideset 1
    infinite_element
    use block 111
  sideset 4
    y=0
    rotz=0
    rotx=0
  sideset 5
    x=0
    rotz=0
    roty=0
END

LOAD 10
    sideset 2
       acoustic_vel = 100
       function = 1
    sideset 3
       pressure = 1
       function = 1
       scale 100
END

TIED DATA
   Surface 2,3
   search tolerance = 5
END

FUNCTION 1
       type planar_step_wave
       origin = 0 0 -10
       direction 0 0 1
       k0 = 1.0
       material = "water"
END
```

```
OUTPUTS
END

ECHO
END

BLOCK 1
material "water"
END

BLOCK 2
        material "steel"
nquad
        thickness = 0.1
END

Block 111
    infinite_element
      order = 10
      ellipsoid_dimensions 30 30 30
END

MATERIAL "water"
        # from paper 0.96e-4 lb-sec2/in4
density 0.96e-4
        acoustic
        c0 60000
END

MATERIAL "steel"
E 0.29e8
nu .3
density 0.732e-3
END


GDSW
  solver_tol 1e-12
  krylov_solver = gmres
  prt_summary 3
END
```

14. Lighthill Function Loading - Input

Refer to Section 23 for details of the test.

```
SOLUTION
  transient
    time_step 1.0e-4
    nsteps 50
    nskip 1
    rho 0.9
    lumped_consistent
END

FILE
   geometry_file 'temp1/acoustic_ns_halfsphere_distFactors.exo'
END

LOADS
  nodeset 1
  lighthill = 1.0
  function = 1
END

LINESAMPLE
  samples per line 100
  endpoint 0. 0. 0. -1 0. 0.
  format exodus
END


FUNCTION 1
       type readnodalset
       nodeset 1
       name "divT_"
       exo_var vector divT_
       interp = linear
END

BOUNDARY
  sideset 6
     infinite_element
     use block 111
END

OUTPUTS
END

ECHO
END

BLOCK 1
material 1
END

BLOCK 2
material 1
END

Block 111
infinite_element
     ellipsoid_dimensions 1 1 1
     order = 8
     source_origin = 0.05 0 0
     neglect_mass = yes
END
```

```
MATERIAL 1
acoustic
density 1.1
c0 343 // reduced to slow down wave
END

Tied Data
surface 2, 5
End
```

15. Linear Buckling - Input

Refer to Section 12 for details of the test.

16. Sierra SM FRF Comparison

Refer to Section 7 for details of the test.

16.1. Modal FRF

```
SOLUTION
   case eig
      eigen
      nmodes = 20
   case test2
      modalfrf
END

FILE
   geometry_file = 'beam_frf.e'
END

LOADS
   nodeset 500
   force = 0.0 0.0 1.0
   scale = 1
   function = 1
END

FUNCTION 1
   type LINEAR
   name "white noise"
   data 0.0 1.0
   data 200. 1.0
END

DAMPING
 alpha = 5
END

BLOCK 1
   material = 1 // rubber linear
END

BLOCK 90
   rbar
END

BLOCK 91
   conmass
   mass = 1e-3
   Ixx = 1e-3
   Iyy = 1e-3
   Izz = 1e-3
END

MATERIAL 1 // linear
   isotropic
   density 0.0343
   E 218
   nu = .499
END

PARAMETERS
   wtmass = 0.002588
END

OUTPUTS
```

```
   disp
   stress
END

FREQUENCY
   freq_min = .1
   freq_step = .1
   freq_max = 50
   acceleration
   disp
   nodeset 2
END

ECHO
   mass=block
END
```

16.2. Direct FRF

```
SOLUTION
   case test2
      directfrf
END

FILE
   geometry_file = 'beam_frf.e'
END

LOADS
   nodeset 500
   force = 0.0 0.0 1.0
   scale = 1
   function = 1
END

FUNCTION 1
   type LINEAR
   name "noise"
   data 0.0 1.0
   data 200. 1.0
END

DAMPING
 alpha = 5
END

BLOCK 1
   material = 1 // rubber linear
END

BLOCK 90
   rbar
END

BLOCK 91
   conmass
   mass = 1e-3
   Ixx = 1e-3
   Iyy = 1e-3
   Izz = 1e-3
END

MATERIAL 1 // linear
   isotropic
   density 0.0343
```

```
   E 218
   nu = .499
END

PARAMETERS
   wtmass = 0.002588
END

OUTPUTS
   disp
   stress
END

FREQUENCY
   freq_min = .1
   freq_step = .1
   freq_max = 50
   acceleration
   disp
   nodeset 2
END

ECHO
   mass=block
END
```

16.3. Adagio Input

```
begin sierra beam_sm_fft

  begin function prescribed_force
    type is piecewise analytic
    begin expressions
        0.0 "1e-4*sin(2*pi*t)"
    end expressions
  end

  begin material rubber
    density = {0.0343*0.002588}

    begin parameters for model elastic
      poissons ratio = 0.499
      youngs modulus = 218
    end parameters for model elastic
  end material rubber

  begin material rbar
    density = 0
    begin parameters for model elastic
      poissons ratio = 0
      youngs modulus = 1e-7
    end parameters for model elastic
  end material rbar

  begin rigid body rbar
  end rigid body rbar

  begin beam section rbar_sec
  rigid body = rbar
  section = bar
  width = 1e-7
  height = 1e-7
  t axis = 0 0 1
  end
```

```
  begin point mass section conmass
  mass = {1e-3*0.002588}
  end

  begin finite element model fft_run
    database name = beam_frf.e
    database type = exodusII

    # - Block id 1 had name bar
    begin parameters for block block_1
      material = rubber
      model = elastic
    end parameters for block block_1

    # - Block id 90 had name rbar
    begin parameters for block block_90
      material =rbar
      model = elastic
      section = rbar_sec
    end parameters for block block_90

    # - Block id 91 had name conmass
    begin parameters for block block_91
      section = conmass
    end parameters for block block_91

  end finite element model fft_run

  begin presto procedure beam_fft

    #
    # *** Time step control information
    begin time control

      begin time stepping block p1
        start time = 0.0
        begin parameters for presto region presto
        time step scale factor =  1.0
          step interval = 100
        end parameters for presto region
      end time stepping block p1

      termination time = 100

    end time control

    begin presto region presto

      begin viscous damping
         include all blocks
         mass damping coefficient = 5
      end viscous damping

      use finite element model fft_run
      ###  output description ###
      begin results output results
        start time = 0
        database name = beam_frf-out.e
        database type = exodusII
At Time 0.0, Increment = 1.0e-1
#At Time 0.0, Increment = 1.0e-5
        nodal Variables = displacement as displ
        nodal Variables = velocity as vel
        nodal Variables = acceleration as acc
      end results output results

      begin prescribed force
      node set = nodelist_500
```

```
      component = z
      function = prescribed_force
      scale factor = 1
      end prescribed force

    end presto region presto

  end presto procedure beam_fft

end sierra beam_sm_fft
```

17. Piezoelectric Transient Input

Refer to Section 17 for details of the test.

```
SOLUTION
  solver=gdsw
  transient
  time_step = 1.000000e-06
  nsteps =  1001
END

FILE
  geometry_file '1/single_patch.par'
END

LOADS
END

GDSW
END

BOUNDARY
  sideset 5       // symmetry boundary condition
    x = 0
  sideset 4       // symmetry boundary condition
    y = 0
  sideset 6       // voltage input
    transV = 1
    function voltage_input
  sideset 7       // grounded voltage
    V = 0
END

RIGIDSET set1
  voltage
  sideset 8
END


FUNCTION voltage_input // voltage input in scaled units (Vin * 1e-9)
  type linear
  name "voltage_in"
  #include CreateInputDeck/voltage_input.inp
END

ECHO
END

OUTPUTS
 disp
 voltage
END

BLOCK 1
  material Aluminum
  hex8u
END

BLOCK 2
  material Piezoelectric
  hex8u
END

BLOCK 3
  material Piezoelectric
  hex8u
```

```
END

MATERIAL ALUMINUM
  density = 2700
  E = {70 * 10^9}
  nu = 0.33
END

// {C11 = 1.38999e+11}
// {C12 = .778366e+11}
// {C13 = .742836e+11}
// {C33 = 1.15412e+11}
// {C44 = 2.5641e+10}
// {C66 = 3.0581e+10}

// {scale = 1e9}

// {ep = 8.85418782e-12 * scale * scale}
// {D11 = ep * 762.5}
// {D33 = ep * 663.2}

// {E11 = -5.20279 * scale}
// {E33 = 15.0804 * scale}
// {E15 = 12.7179 * scale}

MATERIAL PIEZOELECTRIC
  ORTHOTROPIC_PIEZOELECTRIC
    Cij = {C11} {C12} {C13}
                {C11} {C13}
                      {C33}
                      {C44}
                      {C44}
                      {C66}

    permittivity_ij {D11} 0    0
                    0   {D11}  0
                    0     0  {D33}

     e_ij =  0    0    {E11}
             0    0    {E11}
             0    0    {E33}
             0   {E15}  0
           {E15}   0    0
            0      0    0
    density = {7500}
END
```

This page intentionally left blank.

BIBLIOGRAPHY

[1] 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. 2).

This page intentionally left blank.

INDEX

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	K. H. Pierson	1542	0845

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.