SANDIA REPORT

SAND99-2801
Unlimited Release
Printed November 1999

Salinas - User's Notes

Garth Reese, Manoj K. Bhardwaj, Dan Segalman, Kenneth Alvin and Brian Driessen

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

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

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

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

Prices available from (703) 605-6000
Web site: http://www.ntis.gov/ordering.htm

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

NTIS price codes
Printed copy: A19
Microfiche copy: A01

Revision: 1.26

Date: 1999/10/30 22:46:02

Abstract

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

Salinas User Notes

Contents

Salinas

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

This document describes the input for the Salinas program. Examples of input specifications are scattered throughout the document. Appendix A provides several full input files. Appendix B provides instructions on invoking Salinas on a serial UNIX platform. Appendix C details how to execute Salinas on the ASCI-red machine, `janus`.

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

1 Introduction – Input File

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

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

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

Except for data within quotes, the input file is case insensitive. The software converts everything to lower case unless it is enclosed in quotes. Either the single quote ' or the double quote " may be used. The quotes may be nested, e.g. `'a string with "embedded" quotes'`, but only with the other style mark.

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

```
#include english_materials
```

The **#include** may occur anywhere on the line (though for readability and consistency we recommend that it be the start of the line). The file name must immediately follow and should NOT be enclosed in quotes. Case sensitivity will be preserved. Summarizing, a minimum of two files are needed to run **Salinas** , namely, a text input file, e.g. *example.inp*, and an **Exodus** input file,[1] e.g. *example.exo*, which contains the finite element model. The finite element model is specified in *example.inp* as the geometry file.

Each of the **Salinas** input sections is described in the following section.

2 The Salinas Input File

2.1 SOLUTION

The **solution** section determines which solution method, and options are to be applied to the model. The available solution types are shown in Table 1. Relevant options are shown in Table 2, and are described in section 2.2.

Table 1: Salinas Solution Types

Solution Type	Description	Parameters
dump	form matrices only	
eigen	real eigensolution	nmodes, shift
ceigen	unimplemented	
statics	static stress	
checkout	skip large matrix and solves	
transient	implicit transient analysis	time_step, nsteps, nskip
transhock	shock response spectra using direct implicit transient analysis	time_step, nsteps, nskip, freq_step, freq_min, freq_max, srs_damp
modalfrf	frequency response using modal superposition	nmodes, shift, freq_step, freq_min, freq_max
modaltransient	transient analysis using modal superposition	nmodes, shift, time_step, nsteps, nskip
modalshock	shock response spectra using modal approximate implicit transient analysis (unimplemented)	nmodes, shift, time_step, nsteps, nskip, freq_step, freq_min, freq_max, srs_damp

2.1.1 Eigen

The **eigen** keyword is needed to obtain the eigenvalues and mode shapes of a system. The parameters which can be specified for an eigensolution are shown in the table below. By default, if **nmodes** is not specified, a value of 10 is used.

Parameter	Argument
nmodes	*Integer*
shift	*Real*

The **shift** parameter indicates the shift desired in an eigenanalysis. The shift

value represents a shift in the eigenvalue space (i.e. ω^2 space). The value to select is problem dependent. A good starting value is $-\omega_i^2$, where ω_i is the expected first nonzero eigenvalue. For example, a **SOLUTION** section for an eigenanalysis with a **shift** of -1.0, will look like the following, if 12 modes are needed.

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

2.1.2 Ceigen

Currently this is not implemented.

2.1.3 Statics

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

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

2.1.4 Transient

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

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

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

2.1.5 Transhock

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

Parameter	Argument
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
freq_step	*Real*
freq_min	*Real*
freq_max	*Real*
srs_damp	*Real*
node_set	*Integer*

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

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra, and are required. **freq_min** and **freq_max** control the frequency range of the computed spectra, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**; this parameter only controls the quantity of output. The keyword **srs_damp** is a damping constant used for the shock response spectra calculation and is optional. It represents the damping for each single degree of freedom oscillator in the shock spectra computation. Its default value is 0.03. Finally, **node_set** specifies the node set in the Exodus file which defines the nodes for which shock spectra are to be calculated. Its is optional and the default is to compute spectra for all nodes in the model.

Note: Currently, the shock spectrum procedure will only compute acceleration results. The options specified in the OUTPUT and ECHO blocks are used in the transient portion of the analysis, but are ignored for the post-processing of the transient results into shock spectra. Thus, if displacement, velocity, and/or acceleration is selected in the OUTPUT and/or ECHO sections for a shock spectra analysis, the results echoed to the output listing or the Exodus output file will be time history results as requested, but the only shock spectra results will be for acceleration response for the nodes in the specified node set. Furthermore, *the calculated shock*

spectra will only be echoed to the output listing; they are not output to the Exodus results file. The shock spectra output options will be revised and improved in future releases.

2.1.6 Modalfrf

Option **modalfrf** is used to perform a modal superposition-based frequency response analysis. The following table gives the parameters needed for **modalfrf**.

Parameter	Argument
nmodes	*Integer*
shift	*Real*
freq_step	*Real*
freq_min	*Real*
freq_max	*Real*

The **shift** and **nmodes** parameters control the eigenanalysis (see section 2.1.1). The **freq_step**, **freq_min**, and **freq_max** parameters are used to define the frequencies for computing the frequency response, and are required. Keywords **freq_min** and **freq_max** control the frequency range of the computed spectra, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**; this parameter only controls the quantity of output.

2.1.7 Modaltransient

Option **modaltransient** is used to perform a modal superposition-based implicit transient analysis. The following table gives the parameters needed for **modaltransient**. Damping for the model is defined in section 2.15.

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

The **shift** and **nmodes** parameters control the eigenanalysis (see section 2.1.1). The parameters **time_step**, which defines the time integration step size, and **nsteps**,

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

2.1.8 Modalshock

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

Parameter	Argument
nmodes	*Integer*
shift	*Real*
time_step	*Real*
nsteps	*Integer*
nskip	*Integer*
freq_step	*Real*
freq_min	*Real*
freq_max	*Real*
srs_damp	*Real*
node_set	*Integer*

The **shift** and **nmodes** parameters control the modal solution described in section 2.1.1. The time stepping parameters **time_step**, **nsteps** and **nskip** are described in the transient section (2.1.4). Use **freq_min** and **freq_max** to control the frequency range of the computed spectra. The **freq_step** parameter controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**; this parameter only controls the quantity of output. The optional parameter **srs_damp** is a damping constant used for the shock response spectra calculation. Its default value is 0.03. Finally, **node_set** specifies the node set in the **Exodus** file which defines the nodes for which shock spectra are to be calculated. Its is optional and the default is to compute spectra for all nodes in the model. Damping for the model is defined in section 2.15.

2.1.9 Checkout

The **checkout** solution method tests out various parts of the code without forming the system matrices or solving the system of equations. This solution method may

be used to check input files for consistency and completeness on a serial platform before allocating expensive resources for a full solution.

2.1.10 Dump

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

2.2 Solution Options

The options described in Table 2 and in the following paragraphs are part of the **Solution** section in the input file. None of the keywords are required.

Table 2: Salinas Solution Options

Option	Description	Parameters
lumped	Use lumped mass matrices	none
mechanism	check for mechanisms	check_only,continue, abort
constraintmethod	method of applying MPCs	Lagrange or Transform

2.2.1 Lumped – option

Option **lumped** in the **SOLUTION** section causes **Salinas** to use a lumped mass matrix, and not consistent mass matrix, in the analysis.

2.2.2 Mechanism – option

Mechanisms as defined here are zero energy modes introduced into the substructures as a result of decomposing the global finite elment model by a domain decomposition tool. This option will be used rarely since the decomposition tool, `nem_slice`, has options to prevent mechanisms from occurring in the decomposed subdomains. These decompositions are needed for the parallel execution of **Salinas** . Currently, this option works only if the finite element model is composed of either hexes and/or wedges.

The three selections for the mechanism option are *check_only*, *abort*, and *continue*. The *check_only* option will check for mechanisms in the geometry file and stop. It will NOT obtain any solution. The *abort* option will check for mechanisms and solve the problem if there are no mechanisms. If a mechanism does exist, the code will

abort at that point. The *continue* option will check for mechanisms and attempt to solve the problem regardless of any mechanisms that may exist. However, if mechanisms do exist, a warning message will be printed.

2.2.3 Constraintmethod – option

The **constraintmethod** option is defined in the **SOLUTION** section to indicate how multipoint constraints (MPC) will be applied. The selections for applying MPCs are are **Lagrange** and **Transform**. These methods are explained in detail on pp. 272-278 in Ref. 2.

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

2.3 PARAMETERS

This *optional* section provides a way to input parameters that are independent of the solution method or solver. Currently the only parameter supported is **WtMass**, but others may be added in the future. The parameters and their meanings are listed below.

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

Example,

```
Parameters
    wtmass = 0.00259
End
```

2.4 FETI

This *optional* section provides a way to input parameters specific to the Finite Element Tearing and Interconnecting[3] (FETI) solver, if used. If the FETI solver is not used, this section is ignored. It includes the following parameters, shown in Table 3, and options. For those options which are strings, only enough of the string to identify the value is required. The defaults are shown in the following example.

Table 3: FETI Section Options

Variable	Values	Description
rbm	Algebraic/Geometric	rigid body mode method
scaling	Yes/No	scaling method
preconditioner	LUMped/DIRichlet	(both may be used)
max_iter	*Integer*	maximum number iterations
solver_tol	*Real*	
orthog	*Integer*	max number of orthog. vectors
rbm_tol_svd	*Real*	SVD tolerance in rigid body modes
rbm_tol_mech	*Real*	mechanical tolerance in rbm
projector	Standard/Q	projector
level	1 or 2	feti1 or feti2
local_solver	AUto, SKyline, SParse	solver for local lu decomp
precondition_solver	AUto, SKyline, SParse	solver for preconditioner
coarse_solver	AUto, SKyline, SParse	solver for coarse G^TG problem
	PSparse, DIrect, NOne	(psparse is parallel sparse)
grbm_tol	*Real*	tolerance for rigid body detection in G^TG

```
FETI
      rbm  algebraic
      scaling no
//      preconditioner (no preconditioner)
      max_iter 400
      solver_tol 1.0e-8
      orthog 200
      rbm_tol_svd 1.0e-10
      rbm_tol_mech 1.0e-8
      projector standard
      level 1
      local_solver skyline
      precondition_solver auto
      coarse_solver skyline
      grbm_tol 1e-6
END
```

2.5 ECHO

Results, in ASCII format, from the various intermediate calculations may be output to a results file, e.g. *example.rslt*, where the filename is generated by taking the basename of the text input file (without the extension) and adding *.rslt* as an extension. Output to the results file is selected in the **Salinas** input file using the **ECHO** section. An example is given below, and the interpretation of these keywords is shown in Table 4.

```
echo
    materials
    elements
    jacobian
    all_jacobians
    timing
    mesh
    input
    nodes
end
```

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

2.6 OUTPUTS

The **outputs** section determines which data will be written to selected output files. All geometry based finite element results are written to an output exodus file. The name of this file is generated by taking the base name of the input exodus geometry file, and inserting *-out* before the file extension. For example, if the input exodus file specification is *example.exo*, output will be written to *example-out.exo*. More details are available in the **FILE** section (2.7).

Various non-geometry based finite element data, such as system matrices and tables may be available in Matlab compatible format, or in Harwell-Boeing format. These ASCII files have the *.m* or *.hb* file extensions respectively. The base file names are derived from they type of data being output. These files are generated in the current working directory.

In the following example, the mass and stiffness matrices will be output in Matlab format, but the displacement variables, stresses and strains will not be output. All

Table 4: ECHO Section Options

Option	Description
materials	material property info, e.g. E, G
elements	element block info, i.e. what material, element type, etc
jacobian	block summary of jacobians
all_jacobians	jacobians for every element
mesh	summary of data from the input **Exodus** file
input	input deck echo, and summaries of many sections
nodes	nodal summary
timing	timing information
subdomains "0:3:6,10"	Controls which processor will output results file
mass	mass properties
feti_input	
displacement	nodal displacements (better in output section)
velocity	nodal velocities (better in output section)
acceleration	nodal accelerations (better in output section)
force	applied forces (better in output section)
strain	element strains at centroids
stress	element stresses at centroids
all	everything
none	nothing

the various options of the **OUTPUT** section are shown in Table 5. The next sections describe each of the options and their results assuming an input file named *example.inp* and a geometry file named *exampleg.exo*.

```
OUTPUTS
        maa
        kaa
        faa
//      deform
//      stress
//      strain
END
```

2.6.1 Maa

Option **maa** in the **OUTPUTS** section will output the analysis-set mass matrix to a file named *example_Maa.m*. If the harwellboeing option is selected, output will also go a file named *example_Maa.hb*.

2.6.2 Kaa

Option **kaa** in the **OUTPUTS** section will output the analysis-set stiffness matrix to a file named *example_Kaa.m*. If the harwellboeing option is selected, output will also go a file named *example_Kaa.hb*.

2.6.3 Faa

Option **faa** in the **OUTPUTS** section will output the analysis-set force vector to a file named *example_Faa.m*. If the harwellboeing option is selected, output will also go a file named *example_Faa.hb*.

2.6.4 Elemqualchecks

Option **Elemqualchecks** takes either one of two choices, **on** or **off**. The default is **on**. If this option is **on**, then all of the elements in the input file are checked for quality using methods developed by Knupp (Ref. 4). If the element is suspected to be bad, a warning message is printed. The default is ON.

2.6.5 Deform

Option **deform** in the **OUTPUTS** section will output the displacements calculated at the nodes to the output exodus file.

2.6.6 Velocity

Option **velocity** in the **OUTPUTS** section will output the velocities at the nodes to the output exodus file.

2.6.7 Acceleration

Option **acceleration** in the **OUTPUTS** section will output the accelerations at the nodes to the output exodus file.

2.6.8 Strain

Option **strain** in the **OUTPUTS** section will output the strains for all the elements to the output exodus file. For more information on stress/strain recovery, see section 4.

2.6.9 Stress

Option **stress** in the **OUTPUTS** section will output the stresses for all the elements to the output exodus file. For more information on stress/strain recovery, see section 4.

2.6.10 Nodalstrain

Currently, **nodalstrain** option is not implemented.

2.6.11 Nodalstress

Currently, **nodalstress** option is not implemented.

2.6.12 Harwellboeing

Option **harwellboeing** in the **OUTPUTS** section will output the mass and stiffness matrices in Harwell-Boeing format to files with *.hb* extension.

2.6.13 Mfile

Option **mfile** will cause **Salinas** to output various Mfiles like *Ksrr.m*, *Mssr.m*, etc. These files are mainly used by the **Salinas** developers for code maintenance and verification. Since many of these files can be quite large, caution should be exercised when using this option on large models.

2.6.14 Force

Option **force** in the **OUTPUTS** section will output the applied force vector to the output exodus file.

Table 5: OUTPUT Section Options

Option	Description
maa	mass matrix in the a-set
kaa	stiffness matrix in the a-set
faa	force vector in the a-set
elemqualchecks on || off	default is on
deform	displacements at nodes
velocity	velocity at nodes
acceleration	acceleration at nodes
strain	strain of element
stress	stress of element
*nodalstrain	strains at the nodes of the f.e.m.
*nodalstress	stresses at the nodes of the f.e.m.
harwellboeing	mass and stiffness matrices in Harwell-Boeing format
mfile	Outputs various Mfiles (mainly for developers)
locations	Outputs nodal coordinates and DOF to node map
force	Outputs RHS of system of equations to be solved

*Currently, nodalstrain and nodalstress are not implemented.

2.7 FILE

Disk files names are specified in the **FILE** section. The two possible parameters for the **FILE** section are,

Option	Description
geometry_file	Indicates which **Exodus** file to use
numraid	Indicates how many raids are available (for parallel execution)

In an MP environment, the file name is determined by the number of raid controllers and the processor number. The actual file name is computed by this command:

```
sprintf(filename,fmt, (my_proc_id%numraid)+1, my_proc_id );
```

where fmt is the string specified for the geometry file. The number of raid devices is defined using the keyword **numraid**. For example, on a single processor, a **FILE** section may look like this.

```
FILE
    geometry_file 'exampleg.exo'
END
```

On multiple processors this might look like:

```
FILE
  geometry_file '/pfs_grande/tmp_%.1d/junkg/datafile.par.16.%.2d'
  numraid 2
END
```

This will result in opening these files:

```
/pfs_grande/tmp_1/junkg/datafile.par.16.00
/pfs_grande/tmp_2/junkg/datafile.par.16.01
/pfs_grande/tmp_1/junkg/datafile.par.16.02
/pfs_grande/tmp_2/junkg/datafile.par.16.03
/pfs_grande/tmp_1/junkg/datafile.par.16.04
...
/pfs_grande/tmp_2/junkg/datafile.par.16.15
```

Note that if the file name is not included in quotes, it will be converted to lower case. Appendix C shows the steps involved in the parallel execution of **Salinas**.

2.7.1 Additional Comments About Output

A text log or *results* file can be written for the run. Details of the contents of the results file are controlled in the **ECHO** section (see section 2.5). The results file name is determined by the name of the input file, and will be in the same directory as the input text file, regardless of whether **Salinas** is being executed in serial or parallel. However, if executing in parallel, using the subdomain option in the **ECHO** section allows control of the number of *results* files. For example, if running on 100 processors, up to 100 result files may be output. Using **subdomain** "0:2" will only output three files, from subdomains 0, 1, and 2. The default is to output a *results* file only for processor zero. The results file name uses the base name of the input, with an extension of ".rslt". In a parallel computation, the results file names use the base name of the input file, followed by an underscore and the processor number, then followed by the ".rslt" extension.

For calculations in which geometry based output requests are included (see section 2.6), an output **Exodus** file will be created. The **Exodus** file is a binary file containing the original geometry plus any any requested output variables. The output **Exodus** file name is determined from the geometry file name. The base name of the output is taken from the geometry file by inserting the text "-out" just before the file name extension. The output **Exodus** file will be written to the same directory where the geometry file is stored. If executing **Salinas** on a parallel machine, the **Exodus** output files should be written to the raid disks for reasonable performance.

2.8 BOUNDARY

Boundary conditions are specified within the **Boundary** section. Currently, only node sets may be used to specify boundary conditions and in the global coordinate system only. The following example illustrates the method.

```
BOUNDARY
  nodeset 1
    coordinate 7  // read but not currently implemented
       x = 0.1
       y = 0
       RotZ = 0
  nodeset 2
    fixed
END
```

The descriptors for the boundary conditions are, **x**, **y**, **z**, **RotX**, **RotY**, **RotZ**, and **fixed**. A trailing equals sign is optional. Note, that only constant prescribed displacements are allowed on nodesets. In the future boundary conditions may be defined in terms of any coordinate system (see section 2.12).

2.9 LOADS

Loading conditions are specified within the **loads** section. The following example illustrates the method.

```
LOADS
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      force = 0. -1 0
   body
      gravity
      0.0 1.0 0
      scale -32.2
   sideset 7
      pressure 15.0
END
```

Loads may be applied to node sets, side sets or the entire body (in the case of inertial loads). Loads are applied in the global coordinate system using nodesets. Pressure loads may be applied using side sets. The pressure is always normal to the surface. All loads applications are additive.

The syntax followed is to first define the region over which the load is to be applied (either **nodeset**, **sideset** or **body**). Each such region defines a *load set.* For each such definition, one (and only one) load type may be specified. However, any region definition may be repeated so that forces and moments may be applied using the same node set. The load types are,

Option	Parameters
force	*val1 val2 val3*
moment	*val1 val2 val3*
gravity	*val1 val2 val3*
pressure	*val1*

Following the definition of the load type, a vector (or scalar in the case of pressure loads) must be specified. This defines the nominal loading of the load set. The **pressure** loading may only be applied to side sets.

2.9.1 Time Varying Loads

Additional options provide the capability of varying the load over time. The **LOADS** options include,

- **scale** with one parameter provides a scale factor to be applied to the entire load set. Only one scale may be provided per load set.

- **function**. A time varying function may be applied by specifying a function ID. Only one function may be applied per load set. The function is defined in the **FUNCTION** section (see section 2.13 on page 24). The loads applied at time t for a particular load set will be the sum of the force or moment vectors summed over the nodes of the region and multiplied by the scale value and the value of the time function at time t.

Variation of the load over space is accomplished using node set or side set distribution factors. If these are provided in the **Exodus** file, the load set is spatially multiplied by these factors. The total loading is the sum of the loads for each load set summed over all the load set regions.

2.10 BLOCK

Each element block in the **Exodus** file, must have a corresponding **BLOCK** entry in the input file. This section contains information about the properties of the elements within the block. These properties depend on the element type. Clearly shells will require a thickness, while it is meaningless for solids. An example is provided below.

```
// the following element block is Tria3
BLOCK 32
    material 2
    tria3
    thickness 0.01
END

// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default integration mode is "UNDER"
// The only required argument is the material card
BLOCK 34
        material 3
END
```

A list of the applicable attributes for different element types is shown in Table 6. Each element type is outlined in section 3.

2.11 MATERIAL

Most element blocks must specify a material. Details of that material are included in the material section. The material section contains a material identifier (which is usually an integer, but may be any string), an optional **name** keyword followed by a material name, a material type keyword and the necessary parameters. The different material types and their parameters are summarized in Table 7.

For example,

```
MATERIAL 3
    isotropic
    name "steel"
    E 30e6
    nu .3
END
```

Deterministic materials may be input as **isotropic**, **orthotropic** or **anisotropic**. In addition, stochastic isotropic materials may be specified as **S_isotropic**.

Table 6: Element Attributes

Element Type	attr	keyword	Description
ConMass	1	Mass	concentrated mass
	2	Ixx	xx moment of inertia
	3	Iyy	yy moment of inertia
	4	Izz	zz moment of inertia
	5	Ixy	xy moment of inertia
	6	Ixz	xz moment of inertia
	7	Iyz	yz moment of inertia
	8,9,10	offset	offset from node to CG
Beam	1	Area	Area of beam
	2,3,4	Orientation	orientation vector. For the orthogonal direction
	5	I1	First bending moment
	6	I2	Second bending moment
	7	J	Torsion moment
Spring	1	Kx	spring constant in X
	2	Ky	spring constant in Y
	3	Kz	spring constant in Z
Triangle	1	thickness	thickness
Quad	1	thickness	thickness

2.11.1 Isotropic Material

Isotropic materials require specification of two of the following parameters.

Parameter	Description
E	Young's Modulus
nu	Poisson's Ratio
G	Shear Modulus
K	Bulk Modulus

Isotropic materials are the default, and the keyword **isotropic** is not required.

2.11.2 Anisotropic Material

Anisotropic materials require specification of a 21 element C_{ij} matrix corresponding to the upper triangle of the 6x6 stiffness matrix. Data is input in the order C_{11},C_{12}, C_{13}, C_{14}, C_{15}, C_{16}, C_{22}, etc. The C_{ij} must be preceded by the keyword **Cij**. The keyword **Anisotropic** is also required. Materials are specified in the order xx, yy, zz, zy, zx, xy. Note that this ordering varies in the literature. It differs from the ordering in Nastran and Abaqus, but is consistent with much of the published materials science data. An example input file with an anisotropic material is found in section A.2.

2.11.3 Orthotropic Material

Orthotropic material entry is identical to the anisotropic case with the exception that the keyword **orthotropic** replaces **anisotropic**, and only 9 C_{ij} entries are specified. These entries correspond to C_{11}, C_{12}, C_{13}, C_{22}, C_{23}, C_{33}, C_{44}, C_{55} and C_{66}. Like the anisotropic material definition, the order is xx, yy, zz, zy, zx, xy.

2.11.4 Stochastic Material

For stochastic materials, all material properties are determined by a table look-up, based on the element ID. The file name for the table lookup is taken from the **name** identifier. The file is a standard text file with the first column corresponding to the element ID. The second column is the bulk modulus, K, and the third (and final) column is the shear modulus, G. The element IDs in the file need not be continuous, but they must be sorted in increasing order. Thus the **S_isotropic** data lookup file contains the element ID, the bulk modulus and the shear modulus, with one line for

Table 7: Material Stiffness Parameters

material type	parameters
isotropic	any two of K, G, E or ν
orthotropic	nine C_{ij} entries
anisotropic	21 C_{ij} entries
S_isotropic	file containing K and G

each element. The stochastic material model is very preliminary and is expected to change significantly in the next few years.

2.11.5 Density

For solutions requiring a mass matrix, all material specifications require a keyword **density** followed by a scalar *value*.

2.12 COORDINATE

Coordinate systems may be defined for reference to the materials and boundary conditions (currently unimplemented). Note that all nodal locations, outputs, etc are still defined in the basic coordinate system. These new coordinate systems are always defined based on three vectors. These vectors are illustrated in Figure 1.

1. A vector from the origin of the basic coordinate system to the origin of the new coordinate system, v_1.

2. A vector (defined in the basic system) from the origin of the new system to a point on the new system Z axis, v_2.

3. A vector, v_3, from the origin of the new system to a point in the $\tilde{X}\tilde{Z}$ plane of the new system. This vectors need not be orthogonal to v_2, but it may not be parallel to it.

Coordinate systems for cartesian, cylindrical and spherical coordinates may be defined. In the case of noncartesian systems, the XZ plane is used for defining the origin of the θ direction only.

For example, to create a cylindrical system located at a point (1,1,1) with the axis in the (0,0,1) direction and the θ origin in the global Y direction.

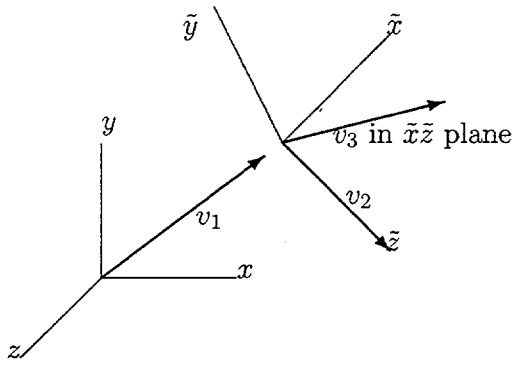

Figure 1: Coordinate System Definition Vectors

```
Coordinate 7
    cylindrical
    1 1 1
    1 1 2
    1 2 1
END
```

The keywords for the coordinate system definitions are:

1. RECTANGULAR or CARTESIAN to define a cartesian system,
2. CYLINDRICAL for a cylindrical, i.e. polar system, and
3. SPHERICAL for a spherical system.

2.13 FUNCTION

Time or frequency dependent functions for transient and frequency response analysis can be defined using the **function** section. The following examples illustrate the use of this section.

```
FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 2
// This is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945.
// The equation is y(t)=-800*t^2 + 8.9943*sqrt(t)

  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END
```

The keywords for the function definitions are:

1. TYPE to define the functional form,
2. NAME for reference in echo and output, and
3. DATA for the functional parameters.

Currently there are two types of functional forms, **linear** and **polynomial**. The data elements are defined in the context of this form.

2.13.1 Linear Functions

For linear functions, the data elements are points of the function where the user defines the value of the independent variable (e.g. time) and the corresponding value of the function. Linear interpolation is used to find all other values of the function. In order to make the linear interpolation unique, the order of the input data is important. Input checks will ensure that time on subsequent data points is always greater than or equal to time on the previous data point so that curves cannot double back on themselves. For example,

```
FUNCTION 3
    name "illegal_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.05 1.
    data 0.04 0.   //illegal. the first column must never decrease
END
```

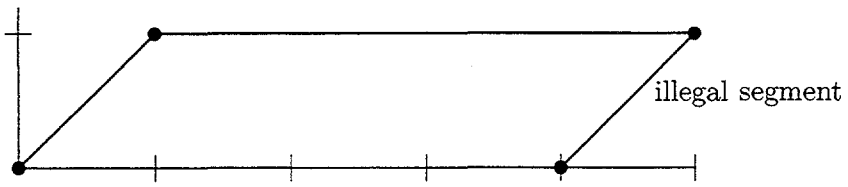

Figure 2: Linear function #3. "illegal_fun"

Linear functions will extrapolate by using the value of the nearest data point. For example, in the following function, f(t=0.3) = 0.5.

```
FUNCTION 5
    name "extrap_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.02 0.5
END
```

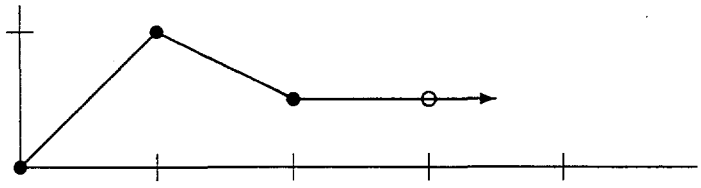

Figure 3: Linear function #5. "extrap_fun"

2.13.2 Polynomial Functions

For polynomials, the data points given are the exponent of the independent variable and a scale factor for that term. The independent variable taken to any real power will always be evaluated as positive. If powers are repeated, their coefficients will sum. For example,

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.
    data 2.0 0.1
    data 1.0 0.5
END
```

is equivalent to

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.5
    data 2.0 0.1
END
```

The function value as a function of the independent variable t is,

$$f(t) = 1.5t + 0.1t^2.$$

2.14 SENSITIVITY

This section controls global parameters related to sensitivity analysis. Sensitivity analysis is not performed in **Salinas** unless this section is present in the input file. The following example illustrates the legal keywords.

```
SENSITIVITY
  values all
  vectors 1 thru 3  5  7 thru 9
  iterations 8
  tolerance 1e-7
END
```

The keywords **values** and **vectors** are used to control what types of sensitivities are computed for which cases in the analysis. In modal analysis, these refer to the eigenvalues and eigenvectors, respectively, and the case numbers represent the mode numbers. In static and transient analysis, **vectors** refers to the displacement vector results, and **values** has no meaning. Also, in modal analysis, eigenvalue sensitivities are always computed when eigenvector sensitivities are requested for a mode. Allowable values are:

```
vectors all              // compute for all cases/modes
vectors none             // compute for no cases/modes
vectors                  // default, same as all
vectors 1 2 3 5          // cases/modes 1,2,3,5
vectors 1 thru 3 5       // using thru to define range
```

Omitting the keyword **vectors** (or **values**) is equivalent to not requesting those sensitivities; in other words, it is equivalent to **vectors none**. The keywords **iterations** and **tolerance** are used in computing eigenvector derivatives. The default values are `10` and `1.0e-06`, respectively.

The selection of parameters is controlled by the inclusion of a +/- symbol following a parameter in the input deck. Examples of valid sensitivity parameter definitions are:

```
MATERIAL 1
  E 10e6 +/- 1e6              // absolute tolerance specified
  density 2.59e-4 +/-         // no tolerance, use default
END

BLOCK 1
  area 0.10 +/- 5 %           // relative tolerance specified
END

BLOCK 2
```

```
  thickness  +/- 1 %           // relative to exodus attr
END

LOADS
  nodeset 1
  force 0. 0. 1000 +/- 0 0 10  // tolerance for vector param
END
```

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

2.15 DAMPING

This section allows input of simple global viscous damping models, using either modal damping rates or stiffness and mass proportional damping. The various options for the DAMPING section are shown in Table 8.

Table 8: DAMPING Section Options

Parameter	Description
alpha	mass proportional damping parameter (real)
beta	stiffness proportional damping parameter (real)
gamma	uniform modal damping rate (% of critical) (real)
mode	individual modal damping rates (% of critical) (integer, real)

The damping matrix or modal damping coefficient is determined by summing contributions from all damping parameters given in Table 8. For modal superposition-based analysis, including **modalfrf** and **modaltransient**, all the given parameters are defined. For direct implicit transient analysis, only the mass and stiffness proportional parameters have meaning and yield the physical damping matrix C as

$$C = \alpha \cdot M + \beta \cdot K$$

where M and K are the assembled mass and stiffness matrices, respectively.

The effect of the mass and stiffness proportional parameters on modal damping depends on the frequencies of the modes. For modal-based analysis, the damping rate for mode i with radial frequency ω_i is given as

$$\zeta_i = \alpha/(2\omega_i) + \beta \cdot \omega_i/2 + \Gamma + \texttt{mode[i]}$$

where the viscous damping term in the modal equilibrium equation is $2\zeta_i\omega_i$. For example the following damping input section could be used in a modal transient analysis.

```
DAMPING
  alpha  0.001
  beta 0.00005
  gamma 0.005
  mode  1  0.01
  mode  2  0.005
  mode  3  0.015
END
```

It produces the following damping ratios.

Mode	modal damping ratio	modal viscous damping term
1	$0.015 + 0.001/(2\omega_1) + 0.00005\omega_1/2$	$0.030\omega_1 + 0.001 + 0.00005\omega_1^2$
2	$0.010 + 0.001/(2\omega_2) + 0.00005\omega_2/2$	$0.020\omega_2 + 0.001 + 0.00005\omega_2^2$
3	$0.020 + 0.001/(2\omega_3) + 0.00005\omega_3/2$	$0.040\omega_3 + 0.001 + 0.00005\omega_3^2$

In direct (i.e. non-modal-based) transient analysis, the same damping input section produces a physical damping matrix $C = 0.001M + 0.00005K$.

3 Element Library

Short descriptions of each of the types of elements follow. Most of the parameters for the element are supplied either in the database file (i.e. **Exodus** file) or in the text input file (*.inp). If parameters exist in both locations, the values specified in the text input will over ride the exodus database specification.

3.1 Hex8

The **Hex8** is a standard 8 node hexahedral element with three degrees of freedom per node. The **Hex8** element has 8 integration points. The shape functions are trilinear. It supports isotropic and anisotropic materials.

There are two variations of **Hex8**. The default element is an under integrated Hex with properties similar to those of most commercial finite element codes. The underintegration produces an element that is soft relative to a fully integrated element. It may be specified by **Hex8** or by **Hex8u**.

The fully integrated Hex is specified by **Hex8F**. While it performs adequately when the element shape is nearly cubic, it performs quite poorly for larger aspect ratios. For most problems involving bending the Hex8u is recommended.

3.2 Wedge6

The Wedge6 is a compatibility element for the **Hex8**, it is not recommended that the entire mesh be built of **Wedge6** elements. They are primarily intended for applications where triangles are naturally generated in mesh generation.

3.3 Tet4

This is a standard 4 node tetrahedral element with three degrees of freedom per node. The **Tet4** element has one integration point. The shape functions are linear. It is not recommended to use only Tet4 elements for the entire mesh because standard, linear tetrahedra are typically much too stiff for structural applications. The **Tet4** is provided primarily for those applications where a mesh may be partially filled with these elements. If a model is constructed of all tetrahedral elements (as by an automatic mesh generator), the **Tet10** is strongly recommended over the **Tet4**.

3.4 Tet10

This is a standard 10 node tetrahedral element with three degrees of freedom per node. The **Tet10** uses 4-point integration for the stiffness matrix and 16-point integration for the mass matrix. The shape functions are quadratic. This is a very good element for use in most structural analyses.

3.5 QuadT

The **QuadT** is a quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements. It is not an optimal element, but is adequate for most applications. A more optimimal element is currently under development. See the description of the **Tria3** for details on the element.

3.6 TriaShell

The **TriaShell** element has 3 nodes with 6 degrees of freedom (DOF) per node. The **TriaShell** is generated by decoupling the membrane DOF and the bending DOF. Allman's Triangular (AT) element[5] models the membrane DOF., while the Discrete Kirchoff Triangle[6] (DKT) models the bending DOF. These two elements are combined into the **TriaShell** element. It currently supports only isotropic materials. The **TriaShell**, like the **Tria3**, has a single parameter, `thickness`.

3.7 Tria3

The **Tria3** is a three dimensional triangular shell with membrane and bending stiffness. There are 6 degrees of freedom per node. In most respects it is very similar to the **TriaShell**. It is the default element for triangular meshes. The **Tria3** was provided by Carlos Felippa of UC Boulder. It currently supports only isotropic materials. It has a single parameter, `thickness`.

#	Keyword	Description
1	thickness	Thickness of the shell

Attributes may either be entered in the **Exodus** file, or in the input file. If an attribute is entered in both locations, the value in the input file will be honored. An example element block is shown below.

```
Block 3
        Tria3
        Thickness 0.01
        material 71
End
```

3.8 Beam2

This is the definition for a Beam element based on Cook's (Ref. 2) development. This beam is similar to the standard Nastran CBAR element. It has no shear contribution. The **Beam2** has 7 parameters.

#	Keyword	Description
1	Area	Area of beam
2,3,4	Orientation	orientation vector
5	I1	First bending moment
6	I2	Second bending moment
7	J	Torsion moment

No stress or strain output is available for beams. Beams are restricted to isotropic materials.

Attributes may either be entered in the **Exodus** file, or in the input file. If an attribute is entered in both locations, the value in the input file will be honored. The following section illustrates the definition of a **Beam2** block.

```
Block 3
   Beam2
   Area 0.71
   I1 .05
   I2 5e-2
   J .0.994
   orientation 1.0 -1.0 0.9
   material 7
 End
```

No beam offsets are supported.

3.9 OBeam

These beams are provided by Carlos Felippa of UC Boulder. They are similar to the simple beams of **Beam2**. They use identical parameters. Because of this

duplication, these beams will probably be eliminated in the future.

3.10 Truss

This is the definition for a **Truss** element based on Cook (Ref. 2). Trusses have stiffness in extension only. The **Truss** has 1 parameter.

#	Keyword	Description
1	Area	Area of truss

No stress or strain output is available for trusses.

3.11 ConMass

Concentrated masses are used to apply a known amount of mass at a point location. Because many meshing tools build beams as a building block for **ConMass**, the geometry definition may be either a line or a point, i.e. the **Exodus** file element types are **BEAM**, **BAR**, **TRUSS** or **SPHERE**. If a beam is used, all the mass is associated with the *first* node of the beam.

Parameters for the **ConMass** are listed below. If not provided in the **Exodus** file or the block definition, all parameters (with the exception of **mass**) default to zero. The value for the **Mass** must be explicitly defined.

#	keyword	Description
1	Mass	concentrated mass
2	Ixx	xx moment of inertia
3	Iyy	yy moment of inertia
4	Izz	zz moment of inertia
5	Ixy	xy moment of inertia
6	Ixz	xz moment of inertia
7	Iyz	yz moment of inertia
8,9,10	offset	offset from node to CG

As an example element block,

```
Block 5
        ConMass
        Mass 1000.0
        Ixx 1.0
        Iyy 2.0
```

```
        IZZ 1.5
        offset 30.0 40.0 50.0
End
```

3.12 Spring

The **Spring** element provides a simple spring connection between nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the exodus database using `BEAM` or `BAR` elements.

The **Spring** element has three parameters. Note also, that if not all parameters are non-zero, the three degrees of freedom of both nodes must be attached to the model in some other way. Currently there is no spring for rotation.

#	Keyword	Description
1	Kx	spring constant in X
2	Ky	spring constant in Y
3	Kz	spring constant in Z

As an example element block,

```
Block 51
        Spring
        Kx 1e6
        Ky 1.11E7
End
```

3.13 Dead

A **dead** element has no mass and no stiffness. It may be of any dimensionality, solid, planar, line or point. Interior nodes to a block of **Dead** elements will not be included in the computation of the model. There are no parameters for **Dead** elements.

3.14 MPC

Multi-Point Constraints (or **MPC**s) are constraint equations applied directly to the stiffness matrix. They are not elements, and are not available from an **Exodus**

database. However, in many respects they look like elements, and can be thought of as elements. Some analysis codes treat them as pseudo elements.

All **MPC**s describe constraint equations of the form,

$$\sum_i C_i u_i = 0$$

where C_i is a real coefficient, and u_i represents the displacement of degree of freedom i.

Unlike many Finite Element programs, **Salinas** does not support user specification of constraint and residual degrees of freedom (DOF). The partition of constrained and retained degrees of freedom is performed simultaneously by gauss elimination with full pivoting so the constrained degrees of freedom are guaranteed to be independent. Redundant specification of constraint equations is handled by elimination of the redundant equations and issue of a warning. User selection of constrained DOF in Nastran has led to significant headaches for analysts who must insure that the constrained DOF are independent and never specified more than once.

Each **MPC** is specified in the input file with a section descriptor. Note that a separate section is required for each equation (or degree of freedom eliminated). An optional coordinate system may be specified on the input, but must be the first entry in the section. The **MPC** will be stored internally in the basic coordinate system (coordinate 0). The input consists of a triplet listing the global ID of the node, a degree of freedom string, and the coefficient of that degree of freedom. The degree of free strings are x, y, z, Rx, Ry, Rz. They are case insensitive.

In the following example, the x and y degrees of freedom in coordinate system 1 are constrained to be equal for node 4.

```
MPC
    coordinate 1
    4 x 1.0
    4 y -1.0
END
```

3.15 RROD

An **RROD** is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for an **RROD** may be conveniently stated that the dot product of the translation and the beam axial direction for a **RROD** is zero. There is one constraint equation per **RROD**.

The **RROD** is specified using beams or trusses in the **Exodus** database, with a corresponding **Block** section in the salinas text input file. No material is required and any number of connected or disconnected **RROD**s may be placed in a block. The following is an example of the input file specification for **RROD**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RROD
END
```

3.16 RBar

An **RBAR** is a *pseudo*element which is infinitely stiff in all the directions. The constraints for an **RBAR** may be summarized as follows.

1. the rotations at either end of the **RBAR** are identical,
2. there is no extension of the bar, and
3. translations at one end of the bar are consistent with rotations.

The **RBAR** is specified using beams or trusses in the **Exodus** database, with a corresponding Block section in the input file. No material is required and any number of connected or disconnected **RBAR**s may be placed in a block. The following is an example of the input file specification for **RBAR**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBAR
END
```

3.17 RBE2

Salinas has no support for the Nastran **RBE2** element. However, in most cases there is little difference between the **RBE2** element and a collection of **RBAR**s.

3.18 RBE3

The **RBE3** pseudo-element's behavior is taken from Nastran's element of the same name. Note however, that the precise mathematical framework of the Nastran **RBE3** element is not specified in the open literature. This element should act like

a Nastran **RBE3** for most applications. The element is used to apply distributed forces to many nodes while not stiffening the structure as an **RBE2** or **RBAR** would. The **RBE3** uses the concept of a slave node. Constraints are specified as follows.

1. The translation of the slave node is the sum of translations of all the other nodes in the element.
2. The rotation of the slave node is the weighted average rotation of all the other nodes about it.

Because all the nodes in an **RBE3** are not identical, each **RBE3** requires its own block ID. In the **Exodus** file, all links connecting to a single **RBE3** are defined in a single element block. The input file then specifies that this is an **RBE3** element block, as shown in the example below. If the model requires many **RBE3**s, a separate block will need to be specified for each.

Note: care must be taken to insure that only one node of the **RBE3** has multiple connections to its links. Further, all links in the **RBE3** must be connected to the slave node.

The following is an example of the input file specification for an **RBE3** if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBE3
END
```

4 Stress/Strain Recovery

Stresses and strains are recovered at the centroids of the finite elements using standard finite element procedures. Stress and strain recovery is not implemented for 1-D elements. The stresses/strains calculated for shell elements are calculated in element space and not global space.

References

[1] Schoof, L. A. and Yarberry, V. R., "EXODUS II: A Finite Element Data Model," Tech. Rep. SAND92-2137, Sandia National Laboratories, 1994.

[2] Cook, R. D. and D. S. Malkaus, M. E. P., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, third edn., 1989.

[3] Farhat, C. and Roux, F.-X., "A Method of Finite Element Tearing and Interconnecting and Its Parallel Solution Algorithm," *International Journal for Numerical Methods in Engineering*, **vol. 32**, 1991, pp. 1205–1227.

[4] Knupp, P. M., "Achieving Finite Element Mesh Quality Via Optimization of the Jacobian Matrix Norm and Associated Quantities : Part II - A Framework for Volume Mesh Optimization and the Condition Number of the Jacobian Matrix," Tech. Rep. SAND99-0709J, Sandia National Laboratories, 1998.

[5] Allman, D. J., "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems," *Computers and Structures*, **vol. 19**, no. 1-2, 1996, pp. 1–8.

[6] Batoz, J.-L., Bathe, K.-J., and Ho, L.-W., "A Study of Three-Node Triangular Plate Bending Elements," *International Journal for Numerical Methods in Engineering*, **vol. 15**, 1980, pp. 1771–1812.

(this page intentionally blank)

A Salinas Example Input Files

The following sections give examples of **Salinas** input files. Note, case sensitivity of the keywords is ignored unless in quotes. The exception is the **#include** command, where the filename following the command must not be in quotes, but case sensitivity is preserved.

A.1 An Eigenanalysis Input File

The following input file will output the first four mode shapes to an **Exodus** output file name *hexplate-out.exo*. A results file, *hexplate.rslt*, will not be created since no results have been selected for output in the **ECHO** section.

```
SOLUTION
        eigen
        nmodes 4
        title 'Obtain First Four Mode Shapes For Hexplate'
END

// The f.e.m. is in hexplate.exo
FILE
        geometry_file         'hexplate.exo'
END

BOUNDARY
        nodeset 77
                fixed
END

LOADS  // loads are unnecessary for eigenanalysis
END

// Only deformations will be output
OUTPUTS
//        maa
//        kaa
//        faa
        deform
//        stress
//        strain
```

```
END

// No results are output to the text log file, *.rslt
ECHO
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
 none
END


// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default hex is an underintegraged hex.
BLOCK 44
         material 3
         hex8
END

MATERIAL 3
         name "steel"
         E 30e6
         nu .3
         density 0.288
END
```

A.2 An Anisotropic Material Input File

The following input file is an example of a hexahedral mesh with anisotropic properties.

```
SOLUTION
        eigen
        title 'Example of anisotropic format'
END

FILE
        geometry_file          'anisogump.exo'
END

boundary
       nodeset 4 y = 0
       nodeset 5 x = 0
       nodeset 6 z = 0
end

loads
        // sum of forces on surface should be equal to area
        // imposed forces are additive
        nodeset 1 force = 0.0  0.083333  0.0
        nodeset 2 force = 0.0 -0.041666  0.0
        nodeset 3 force = 0.0 -0.020833  0.0
end


OUTPUTS
//        maa
//        kaa
//        faa
          deform
//        stress
//        strain
END

ECHO
```

```
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
none
END


// the following element block  is all hex
BLOCK 1
        hex8
        material 1
END

MATERIAL 1
        name "anisotropic gump"
        anisotropic
        Cij
        1.346     0.5769    0.5769  0       0       0
                  1.346     0.5769  0       0       0
                            1.346   0       0       0
                                    0.3846  0       0
                                            0.3846  0
                                                    0.3846
        density 1
END
```

A.3 A Multi-material Input File

The next example shows the input for an **Exodus** model with many element blocks and materials. Keyword **lumped** in the **SOLUTION** section causes **Salinas** to use a lumped mass matrix instead of a consistent mass matrix.

```
SOLUTION
        eigen
        nmodes 1
        titile 'Multiple block, multiple material example'
        lumped
END

FILE
        geometry_file          'multi.exo'
END

BOUNDARY
    nodeset 1
    fixed
    nodeset 3
    x = 0
    y = 0
    z = 0
    RotY = 0
    RotZ = 0
END

OUTPUTS    // output only displacements to exodus file
     deform
END

ECHO
    none
END

// element block specifications. One such definition per element
// block in the exodus (genesis) database.
BLOCK 1
```

```
        material 2
        Beam2
END

BLOCK 101
        kind solid
        integration full
        wedge6
        MATERIAL 1
END

BLOCK 2
        material 2
END

BLOCK 102
        kind solid
        integration full
        wedge6
        MATERIAL 2
END

BLOCK 3
        material 3
END

BLOCK 103
        kind solid
        integration full
        wedge6
        MATERIAL 3
END

BLOCK 4
        material 4
END

BLOCK 104
        kind solid
        integration full
```

```
        wedge6
        MATERIAL 4
END

BLOCK 5
        material 5
END

BLOCK 105
        kind solid
        wedge6
        integration full
        MATERIAL 5
END

BLOCK 6
        material 6
END

BLOCK 106
        kind solid
        wedge6
        integration full
        MATERIAL 6
END

// material specifications. Extra materials are acceptable, but
// every material referenced in a necessary "Block" definition,
// must be included here.
MATERIAL 1
        name "Phenolic"
        E 10.5E5
        nu .3
        density 129.5e-6
END

Material 2
        name 'Aluminum'
        E 10.0E6
        nu 0.33
```

```
        density 253.82e-6
END


Material 3
        name 'foam'
        E 100.
        nu 0.3
        density 18.13e-6
END

Material 4
        name 'HE'
        E 5E5
        nu 0.45
        density 129.5e-6
END

material 5
        name 'Uranium'
        E 30e6
        nu 0.3
        density 1768.97e-6
end

material 6
        name 'wood'
        E 200.e3
        nu .3
        density 77.7e-6
end
```

A.4 A Modaltransient Input File

The next example shows the input for a **modaltransient** analysis. Accelerations are output to an **Exodus** file *bar-out.exo*. This example has damping, polynomial and linear functions. Also, sensitivities are calculated.

```
SOLUTION
  modaltransient
    nmodes 10 shift -100
    time_step .000005
    nsteps 100
    nskip 1
    title 'Test modal transient on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
  kind solid
END
```

```
MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 0.0001 1.0
  data 0.0001 0.0
  data 1.0 0.0
END

SENSITIVITY
  vectors all
END
```

A.5 A Modalfrf Input File

The next example shows the input for a **modalfrf** analysis. Accelerations are output to an **Exodus** file *bar-out.exo*.

```
SOLUTION
  modalfrf
    nmodes 10 shift -100
    freq_step 1500
    freq_min 1500
    freq_max 3000
    title 'Test modalfrf on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
  kind solid
END
```

```
MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 2
// this is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945
  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 10000. 1.0
END

SENSITIVITY
  vectors all
END
```

A.6 A Statics Input File

The following example is a **statics** analysis which will output stresses to the **Exodus** output file *quadt-out.exo*.

```
SOLUTION
        statics
        title '10x1 beam of quadt'
END

FILE
         geometry_file          'quadt.exo'
END

BOUNDARY
           nodeset 1
            fixed
END

LOADS
           nodeset 2
            force = 1000.0 1000.0 0.0
END

OUTPUTS
      stress
END

ECHO
  none
END

// the following element block is quadt
BLOCK 1
         material 1
         QuadT
END

MATERIAL 1
         name "steel"
```

```
            E 30.0e6
            nu 0.25e0
            density 0.7324e-3
END
```

B Running Salinas on serial UNIX platforms

On serial unix platforms, **Salinas** is run with a single argument, the ASCII input file.

```
salinas example.inp
```

The log file will be written to *example.rslt* if outputs have been specified in the ECHO section. If outputs have been specified in the OUTPUTS section, a new exodus file will be generated. The file name is derived from the `geometry_file` specified in the ASCII input (see section 2.7).

Visualization of the exodus output results can be accomplished using a variety of `seacas` codes. This include `blot` (for models with supported element types), and `mustafa`. Commercial software with exodus preferences is also available. These include *MSC/Patran* and *EnSight*. For more information, contact the authors.

(this page intentionally blank)

C Running Salinas in Parallel

This appendix gives an example of how to perform an analysis on the Intel Teraflop (**janus**) using **Salinas**. This implies that the execution of **Salinas** will be in parallel. There is some overhead to running in parallel versus serial. Assuming a **Salinas** text input file exists and an **Exodus** file exists which contains the finite element model, the following steps are needed.

1. Decide on how many processors, *nproc*, are needed.

2. Create an input file for **nem_slice**. The partition software can be executed on a workstation to create a load balance file. The name of this file is specified in the input file for *nem_slice*, and usually has a *.nem* extension.

3. Create your workspace on **janus** on /scratch/tmp_?? - where ?? is (currently) your choice of 1 thru 10.

4. Move the **Salinas** input file, **Exodus** file, and load balance file to your work space on **janus**.

5. Create an input file for **nem_spread**. Execution of **nem_spread** (on **janus**) with this input will create *nproc* **Exodus** files from the master **Exodus** file and move them to the locations specified in the **nem_spread** input file.

6. Modify the **FILE** section of the **Salinas** input file to agree with the number of RAID disks available and the location of the subdomain **Exodus** files created by **nem_spread**.

7. Modify the **ECHO** section in the **Salinas** input file using the keyword **subdomain** to indicate which processors should produce text results files. Having all processors output text results files is very slow for large models.

8. Use the **yod** command to run **Salinas** in parallel.

9. Create an input file for **nem_join** to join your results back into one **Exodus** output file.

Each step is detailed in the following paragraphs. Additional information on parallel execution can be found at *http://jal.sandia.gov* under the **SEACAS** documentation link.

C.1 Number of Processors Needed

Running **Salinas** in parallel requires the user to specify how many processors at a minimum are needed in order to "fit" the problem into available memory on **janus**. For most applications running om compute nodes with 128MB per node, a good rule of thumb is to have approximately 2000 elements/processor. If a finite element model has 1,000,000 elements, use 500 processors.

C.2 Using Nem_slice to load balance the model

An example of a **nem_slice** input file is, e.g. *junk_slice.inp*,

```
Graph Type = elemental
Decomposition Method = multikl,cnctd_dom
Input ExodusII File = junk.exo
Output NemesisI File = junk.nem
#Solver Specifications
Machine Description = mesh=500
Misc Options = face_adj
#Weighting Specifications
```

This input file will create a load balance file, *junk.nem*, for running **Salinas** on 500 processors. Note, the *face_adj* option is useful for 3-d models to prevent mechanisms from appearing in the decomposed subdomains and is highly recommended for optimal performance.

To create the load balance file, *junk.nem*, simply type

```
prompt> nem_slice -a junk_slice.inp
```

The load balancing software, **nem_slice**, is typically executed on a serial machine such as a workstation. More detailed information on **nem_slice** is available at *http://jal.sandia.gov* under the link to the **SEACAS** documentation.

C.3 Janus Work Space

To run **Salinas** in parallel, work space on **janus** is needed. On the /scratch space on **janus**, there are 10 temp directories. Simply choose one, and make a directory using your username, as follows.

```
janus> cd /scratch/tmp\_1
janus> mkdir \$USER \\
```

After the work space on janus is set up, move the **Salinas** input file, **Exodus** file, and load balance file (*junk.nem*) to it.

C.4 Using Nem_spread

The load balanced **Exodus** database must be "spread" to *nproc* mini-databases. Each processor reads from its own data file. An example **nem_spread** input file is, e.g. *junk_spread.inp*.

```
Input FEM file                    = junk.exo
LB file                           = junk.nem
Debug                             = 4
------------------------------------------------------------
               Parallel I/O section
------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_, subdir=username
```

Here, username must be replaced by the name of the user.

The **Exodus** file and the load balance file need to be defined in the **nem_spread** input file. There are 18 RAID disks currently available on **janus**. These are the number of disks available to which input/output can be performed in parallel. The **FILE** section in the **Salinas** input file needs to have the number of raids defined using the keyword **numraid**. Therefore, for janus, **numraid 18**, must appear in the **Salinas** input file. This number must match the parallel disk info line in the **nem_spread** input file.

If running for the first time on janus, proper directories must be established on the RAID disks. Currently, the raids are setup at */pfs_grande/tmp_??* where ?? is a number between 1 and 18 (18 raids). A few csh shell commands can make the required directories.

```
janus> foreach i (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18)
foreach? mkdir /pfs_grande/tmp_$i/$USER
foreach? end
janus>
```

To execute **nem_spread**,

```
janus> /cougar/bin/yod -sz 4 nem_spread junk_spread.inp
```

This execution of **nem_spread** will spread *nproc* **Exodus** files onto the RAID disks specified in the input file for **nem_spread**. This location must also be specified in the **FILE** section of the **Salinas** input file as follows, assuming your load balance file is *junk.nem* created for 500 processors,

```
FILE
   geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
   numraid 18
END
```

The "%d" after *tmp_* is used in **Salinas** in conjunction with the number of RAIDs available. The "%.3d" at the end of the line for the **geometry_file** is used in conjunction with how many processors the load balance file was created with. The following table shows what must be used after *junk.par.*nproc for various processors requested.

Condition	Use this
nproc<10	"%.1d"
10≤*nproc*<100	"%.2d"
100≤*nproc*<1000	"%.3d"
1000≤*nproc*<10000	"%.4d"

Since **nem_spread** is a parallel code, **yod** must be used to execute it, using the -sz option to specify how many processors are needed. This number need not agree with the number of processors for execution of the analysis. Typically no more than 20 processors would be used to spread files. The **showmesh** utility can be used to indicate the number of interactive processors available.

C.5 Salinas FILE Section

If a load balance file *junk.nem* is created for execution of **Salinas** for 500 processors, and the number of raids is 18, then the **FILE** section of the **Salinas** input file must look like the following.

```
FILE
  numraid 18
  geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
END
```

C.6 Running Salinas

Once the necessary setup has been done, and a parallel **Salinas** code exists in your work space, then

```
janus> cd /scratch/tmp_1/$USER
janus> yod -sz 500 salinas junk.inp
```

This will run **Salinas** in parallel on 500 processors using the input file *junk.inp*.

In practice, only a small number of processors are available interactively on **janus**. To use a larger number of processors, the NQS queuing system must be used. Help is available under the `man` pages on **janus** under the topics `qsub` and `qstat`. To submit an NQS submission, create a small shell script, such as the following.

```
janus> cat run_it
#!/bin/sh
date
cd /scratch/tmp_1/$USER
yod -sz 500 salinas junk.inp
date
```

The NQS job is submitted using `qsub` with a command such as the following.

```
/usr/bin/qsub -lT 90:00 -lP 500 -q snl.day -me run_it
```

This command submits a 90 minute run using 500 processors to the queue `snl.day`. A message will be mailed to you when the run has completed, and output from standard out and standard error will be found in files in your working directory. Status of your run can be obtained using `qstat`. Status of all NQS submissions is available with `qstat -a` or `qstat -av`. Contact *janus-help@sandia.gov* for information on queueing policies and options.

C.7 Using Nem_join

Once the analysis run has been completed, the output exodus files will need to be recombined into a single file for visualization and processing. **Nem_join** accomplishes this process. A **Nem_join** input file is very similar to the **nem_spread** input file. An example input file is, e.g. *junk_join.inp*.

```
Input FEM file                  = junk.exo
Scalar Results FEM file         = junk-out.exo
Use Scalar Mesh File            = yes
Parallel Results file base name = junk.par
Number of processors            = 500
Debug                           = 4
-------------------------------------------------------------
                Parallel I/O section
-------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_,subdir=username
```

To run **nem_join**, simply do the following:

```
janus> yod -sz 4 nem_join junk_join.inp
```

This will create a file *junk-out.exo* in your current directory by combining all the **Exodus** output files located on the RAID disks. This is a standard exodus file which may be visualized and processed using serial tools.

Distribution List:

MS0841 P. J. Hommert, 9100
MS0828 T. C. Bickel, 9101
MS0828 R. K. Thomas, 9102
MS0836 C. W. Peterson, 9104
MS0835 S. N. Kempka, 9111
MS0834 A. C. Ratzel, 9112
MS0826 W. Hermina, 9113
MS0825 W. H. Rutledge, 9115
MS0827 R. Griffith, 9117
MS0847 J. Peery, 9121
MS0555 M. S. Garett, 9122
MS0443 H. S. Morgan, 9123
MS0439 D. R. Martinez, 9124
MS0557 T. J. Baca, 9125
MS0553 R. A. May, 9126
MS0827 J. D. Zepper, 9130
MS0828 J. Moya, 9132
MS0439 B. J. Driessen, 9121 (3)
MS0439 M. J. Bhardwaj, 9121 (10)
MS0439 G. M. Reese, 9121 (25)
MS0439 K. F. Alvin, 9124 (3)
MS0439 D. J. Segalman, 9124
MS0321 W. J. Camp, 9200
MS0318 P. Heermann, 9215
MS0439 J. R. Red-Horse, 9211
MS1111 S. Dosanjh, 9221
MS1110 D. M. Day, 9222 (3)
MS1110 D. Womble, 9222
MS1110 R. S. Tuminaro, 9222
MS1110 N. Pundit, 9223
MS0321 A. L. Hale, 9224
MS0441 R. W. Leland, 9226
MS0439 C. R. Dohrmann, 9226

MS9018 Central Technical File, 8940-2 (1)
MS0899 Technical Library, 4916 (2)
MS0612 Review & Approval Desk, 4912
for DOE/OSTI (1)

Charbel Farhat
University of Colorado at Boulder
Center for Aerospace Structures
Boulder, CO 80309-0429

Michel Lesoinne
University of Colorado at Boulder
Center for Aerospace Structures
Boulder, CO 80309-0429

Kendall Pierson
University of Colorado at Boulder
Center for Aerospace Structures
Boulder, CO 80309-0429