SALINAS–Program Notes

Garth Reese[*] Dan Segalman[†] Manoj K. Bhardwaj[‡]
Kenneth Alvin[§] Brian Driessen Kendall Pierson[¶]
Timothy Walsh[‖]

Sandia National Laboratories
Albuquerque, NM 87185-0847

January 8, 2004

Revision: 1.30

Date: 2003/10/27 23:32:21

[*]Phone: 845-8640
[†]Phone: 844-0972
[‡]Phone: 844-3041
[§]Phone: 844-9329
[¶]Phone: 284-5894
[‖]Phone: 284-5374

Contents

Salinas - Program Notes

The problem of calculating the terms of the stiffness matrix.

1 Notes on calculating stiffness matrixes for Hex elements

The following applies to any solid isoparametric element, but is implemented in code on hex elments. This discussion addresses calculation of relevant operators on the shape functions and eventual integration into the stiffness matrices.

1.1 Derivation

We begin with the separation of the strain into deviatoric and dillitational parts so that their contributions to the stiffness matrix can be computed separately. This is part of the strategy for avoid ing overstiffness with respect to bending.

The strain energy density in the case of an isotropic, linearly elastic material is:

$$p = \frac{1}{2}(2G\epsilon + \lambda tr(\epsilon)I) \bullet \epsilon \tag{1}$$

with some re-arrangement, this can be shown to be:

$$p = G\hat{\epsilon} \bullet \hat{\epsilon} + \frac{1}{2}\beta(tr(\epsilon))^2 \tag{2}$$

where $\hat{\epsilon} = \epsilon - \frac{1}{3}tr(\epsilon)I$.

Having separated the part of the strain energy density due to deviatoric part of the strain from the part of the strain energy density due to the dillitational part of the strain, we shall integrate them separtely. First, we must determine how to express the strains in terms of nodal degrees of freedom.

We know that the deformation field is linear in the nodal degrees of freedom and that the displacement gradient is also, so we should be able to expand each of those quantities as follows. Let P_j be the node associated with the $j$the degree of freedom and let s_j be the direction associated with that degree of freedom. The displacement field is:

$$\vec{u}(x) = \tilde{N}^{P_j}(x)u_{s_j}^{P_j}\vec{e}_{s_j} \tag{3}$$

where summation takes place over the degree of freedom j.

Similarly, the displacment gradient is:

$$\vec{\nabla}\vec{u}(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x)u_{s_j}^{P_j}\vec{e}_{s_j}\vec{e}_k \tag{4}$$

We now define the shape deformation tensor W^j corresponding to the j th nodal degee of freedom:

$$W^j(x) = (\frac{\partial}{\partial u_{s_j}^{P_j}})\vec{\nabla}\vec{u}(x) \tag{5}$$

which, with Equation 4 yields:

$$W^j(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x)\vec{e}_{s_j}\vec{e}_k \tag{6}$$

The symmetric part of this tensor is:

$$S^j(x) = \frac{1}{2}(W^j(x) + W^j(x)^T) \tag{7}$$

and the strain tensor is

$$\epsilon(x) = S^j(x)u_{s_j}^{P_j} \tag{8}$$

From the above, we construct the dillatational and deviatoric portions of the strain in terms of the nodal displacement components:

$$tr(\epsilon(x)) = b^j(x)u_{s_j}^{P_j} \tag{9}$$

where

$$b^j(x) = tr(S^j(x)) \tag{10}$$

Similarly,

$$\hat{\epsilon}(x) = \hat{B}^j(x)u_{s_j}^{P_j} \tag{11}$$

where

$$\hat{B}^j(x) = S^j(x) - \frac{1}{3}b^j(x)I \tag{12}$$

The stiffness matrix is evaluated using the consitutive equation (Equation 2) and the following definition:

$$K_{m,n} = \frac{\partial^2}{\partial u_{s_m}^{P_m}\partial u_{s_n}^{P_n}}\int_{volume} p(x)dV(x) \tag{13}$$

This plus our expressions for strain in terms of the nodal degrees of freedom yield us the following expression for element stiffness:

$$K_{m,n} = G \int_{volume} (\hat{B}^m(x))^T \bullet \hat{B}^n(x) dV(x) \tag{14}$$

$$+\beta \int_{volume} b^m(x) b^n(x) dV(x) \tag{15}$$

1.2 Implementation

From the above it is seen that once the shape deformation tensor W^j is found, the rest of the calculation follows naturally. The calculation of the components of that tensor is presented here.

The components of W^j are

$$W^j_{mn} = \vec{e}_m \cdot W^j \cdot \vec{e}_n \tag{16}$$

$$= \delta_{m,s_j} (\frac{\partial}{\partial x_n}) \tilde{N}^{P_j}(x) \tag{17}$$

The partial derivative $(\frac{\partial}{\partial x_n})\tilde{N}^{P_j}(x)$ is calculated from

$$(\frac{\partial}{\partial x_n}) \tilde{N}^{P_j}(x(\xi)) = (\frac{\partial}{\partial \xi_\alpha}) N^{P_j}(\xi) J^{-1}_{\alpha,n} \tag{18}$$

where

$$J_{m,\gamma} = \frac{\partial}{\partial \xi_\gamma} x_m(\xi) \tag{19}$$

and

$$N(\xi) = \tilde{N}(x(\xi)) \tag{20}$$

The issue of selective integration in the elements is discussed in a Framemaker file /home/djsegal/MPP/notes/IsoInt.frm. The formulation discussed there applies to all the isoparametric solid elements.

2 Notes on volumetric and deviatoric strain coefficient settings for higher order elements (Hex20, Tet10, ...)

Quadratic elements (elements with bilinear or higher order shape functions) such as the Hex20 and Tet10 are naturally soft and do not need to be softened by positive values of G and β (see the section "Notes on calculating stiffness matrices for Hex elements" and the associated Framemaker file IsoInt.frm for definitions of G and β.) Therefore, G=0 and β=0 are good values for such elements.

3 Notes on calculating stiffness matrixes for Wedge elements

3.1 Shape Functions

The shape functions are given explictly in *Hughes*. These are provided as bi-linear polynomials in r, s, t, and ξ, where r and s are independent coordinates of the triangular cross-sections, $t = 1 - r - s$, and ξ is the coordinate in the third direction. For our purposes, it is necessary to expand the shape functions as polynomials in r, s, and ξ:

$$N_k = A_0^k + A_1^k r + A_2^k s + A_3^k \xi + A_4^k r\xi + A_5^k s\xi \tag{21}$$

The shape functions and the coefficients are given in the following table:

Shape Function	A_0	A_1	A_2	A_3	A_4	A_5
$N_1 = \frac{1}{2}(1-\xi)r$		$\frac{1}{2}$			-$\frac{1}{2}$	
$N_2 = \frac{1}{2}(1-\xi)s$			$\frac{1}{2}$			$-\frac{1}{2}$
$N_3 = \frac{1}{2}(1-\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$N_4 = \frac{1}{2}(1+\xi)r$		$\frac{1}{2}$			$\frac{1}{2}$	
$N_5 = \frac{1}{2}(1+\xi)s$			$\frac{1}{2}$			$\frac{1}{2}$
$N_6 = \frac{1}{2}(1+\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$

3.2 Quadrature

Three reasonable quadratures for wedges that come to mind are indicated in the following table:

No. Points	r	s	ξ
1	1/3	1/3	0
2	1/3	1/3	-1/$\sqrt{3}$
	1/3	1/3	1/$\sqrt{3}$
6	1/6	1/6	-1/$\sqrt{3}$
	1/3	1/6	-1/$\sqrt{3}$
	1/6	1/3	-1/$\sqrt{3}$
	1/6	1/6	1/$\sqrt{3}$
	1/3	1/6	1/$\sqrt{3}$
	1/6	1/3	1/$\sqrt{3}$

4 Notes on calculating stiffness and mass matrices for Tet10 elements

The 4-point integration is given in *Hughes*, and the 16-point integration is given in *Jinyun*. It is believed that a higher order integration is needed for the mass matrix than the stiffness matrix and that the reason is that the mass matrix involves higher degree polynomials. (Using 4-point integration to try to estimate the mass matrix of a natural element resulted in a 30 by 30 mass matrix with several zero eigenvalues.)

5 Notes on Tetrahedral shape functions and related functions

gmreese. Dec 19, 2000.

The tet elements (both tet4 and tet10) were developed from formulations in Cook. In that formulation, the element shape functions are defined in terms of four volumetric coordinates ξ_i. However, we have implemented the element using the standard isoparametric formulation. This uses three coordinates (which I will label w_i). Most of the internal formulation of the element uses the volumetric coordinates. However, when computing jacobians of transformation, the isoparametric formulation uses a 3x3 matrix. The code simply used the first three coordinates, and computed the fourth coordinate internally. This results in an incorrect jacobian.

To do it correctly, we should compute the full jacobian of transformation from the x, y, z coordinates of the real space, to the w_1, w_2, w_3 coordinates of the isoparametric space. Thus,

$$\frac{dx_i}{dw_i} = \frac{dx_i}{d\xi_i}\frac{d\xi_i}{dw_i}$$

Since, $d\xi_i/dw_i$ is computed in element space, it is a constant.

$$\frac{d\xi_i}{dw_i} = \begin{pmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The determinant of transformation of this matrix is -1. Since we don't actually use the jacobian matrix in external calculations, it is only necessary to transform the determinant of the jacobian. Internally we compute $det(dx_i/d\xi_i)$; to get $det(dx_i/dw_i)$ we simply multiply by -1.

The current code does not make all these changes. However, the current code has been performing well, and has had verified results. Rather than changing all the gradients, etc. I am changing only the computation of the determinant of the jacobian which is found in Jacobian(). Later there will be a need to reformulate many of our elements for inclusion in Sierra. This may be a good time to consider getting all the gradients correct.

6 Notes on calculating shape functions and their gradients for the Hex20 element

See file Hex20.frm, which is a Framemaker file with a detailed description of how the shape functions and their gradients are calculated for the Hex20 element.

7 Anisotropic Elasticity

Anisotripic elasticity requires special care in the rotation of the matrix of matrerial parameters when those parameters are given in some coordinate system other that in which the element matrices are calculated. A derivation of the formulae for rotating those matrices is given in a framemaker file
/home/djsegal/MPP/notes/anisoConst.frm.

8 Two Node Beam

This is the definition for a Beam element based on Cook's development. See Cook, Robert D, Malkus, David S., and Plesha, Michael E. "Concepts and Applications of Finite Element Analysis", Wiley 1989, pp 113-115.

The beam uses underintegrated cubic shape functions. Only isotropic material models are supported. Torsional affects are accounted for in the axis of the beam. The beam is uniform in area and bending moments, i.e. they are not a function of position in the beam.

The following parameters are read from the exodus file.

1. The cross sectional area of the beam (Attribute 1)

2. The orientation of the beam (Attributes 2, 3 and 4)

   The orientation should not be aligned with the beam axis. In the event of an inproperly specified orientation, a warning will be written, and a new orientation selected. The orientation is an x,y,z triplet specifying a direction. It does not need to be completely perpendicular to the beam axis, nor is it required to be normalized. The orientation vector, and the beam axis define the plane for the first bending direction.

3. The first bending moment, I1. (Attribute 5).

4. The second bending moment. I2. (Attribute 6).

5. The torsional moment, J. (Attribute 7).

9 Truss

This is the definition for a Truss element based on Cook. See Cook, Robert D, Malkus, David S., and Plesha, Michael E. "Concepts and Applications of Finite Element Analysis", Wiley 1989, pp 214-216.

The truss uses linear shape functions. Unlike the truss elements used by Nastran, there is no torsional stiffness. The truss is uniform in area, i.e. the area is not a function of position in the truss.

The following parameters are read from the exodus file.

1. The cross sectional area of the truss (Attribute 1)

10 Springs

The *Spring* element is the simplest one dimensional element. It has no mass. Entries in the stiffness matrix are added by hand. Note the following.

- The force generated in a *Spring* element should be colinear with the the nodes. Typically spring elements connect coincident nodes so that no torques are generated.

- *Springs* attach 3 degrees of freedom. In the event that some of the spring constants are zero, there is no effective stiffness for that associated degree of freedom. However, the degree of freedom will remain in the A-set matrices. This will be a problem if the other degrees of freedom are not attached to other elements which provide stiffness entries connecting them to the remainder of the model.

The data for spring elements is entered in the input file. Three values are given, Kx, Ky, and Kz. This results in a 6x6 element stiffness matrix,

$$K' = \begin{pmatrix} K_x & 0 & 0 & -K_x & 0 & 0 \\ 0 & K_y & 0 & 0 & -K_y & 0 \\ 0 & 0 & K_y & 0 & 0 & -K_z \\ -K_x & 0 & 0 & K_x & 0 & 0 \\ 0 & -K_y & 0 & 0 & K_y & 0 \\ 0 & 0 & -K_z & 0 & 0 & K_z \end{pmatrix}$$

Notice that K' is blocked. It could be written more simply,

$$K' = \begin{pmatrix} K'_{11} & K'_{12} \\ K'_{12} & K'_{11} \end{pmatrix}$$

The rotation matrix for the two endpoints is block diagonal. As a result, the stiffness matrix in the basic coordinate system can be written,

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{12} & K_{11} \end{pmatrix}$$

where,

$$K_{ij} = R^T K'_{ij} R$$

and R is the 3x3 rotation matrix of section 18.

11 Multi-Point Constraints, MPCs

A description of *MPC*s is contained in the users manual. This section discusses the coordinate system dependencies.

*MPC*s may be defined in any coordinate system. However, all nodes in the *MPC*s are defined in the same system. This is done for convenience in parsing, and not for any fundamental reason. Consider a constraint equation where each entry in the equation could be specified in a different coordinate system.

$$\sum_i C_i u_i^{(k_i)} = 0$$

where C_i is a real coeffient, and $u_i^{(k_i)}$ represents the displacement of degree of freedom i in degree of coordinate system k_i. We can transform to the basic coordinate system using $u_i^{(k_i)} = \sum_j R_{ji}^{(k_i)} u_j^{(0)}$, where $R^{(k_i)}$ is the rotation matrix for coordinate system k_i. Then we may write,

$$\sum_{i,j} C_i R_{ji}^{(k_i)} u_j^{(0)} = 0$$

or,

$$\sum_i C_i^{(k_i)} u_i^{(0)} = 0$$

where $C_i^{(k_i)} = \sum_j R_{ij}^{(k_i)} C_j$. Note however, that in this analysis, we have assumed that the dimension of C is 3. Thus, rotation into the basic frame will likely increase the number of coefficients.

Salinas is designed to support constraints through at least two methods. This include a constraint transform method and Lagrange multipliers. Lagrange multipliers have not been implemented at this time.

11.1 Constraint Transforms

Constraints may be eliminated using the constraint transform method. This is described in detail in Cook, chapter 9. In this method, the analysis set is partitioned into constrained degrees of freedom and retained degrees of freedom. The constrained dofs are eliminated.

Unlike many Finite Element programs, Salinas does not support user specification of constraint and residual degrees of freedom. The partition of constrained and retained degrees of freedom is performed simultaneously in the "gauss()" routine. This routine performs 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 dofs in Nastran has led to serious difficulty to insure that the constrained dofs are independent and never specified more than once.

For constraint elimination we have a constraint matrix $C = C_c C_r$, where C_c is a square, nonsingular matrix and C_r is the solution. We wish to solve for,

$$C_{rc} = -[C_c]^{-1} C_r$$

This is equivalent to the Gauss-Jordan elimination probrlem for $Kx = b$ if we let $C_r = b$, $C_c = K$ and $x = -C_{rc}$. There is one additonal wrinkle: we have mixed the rows of C so C_c is intermingled with C_r. However, we only require that C_C be non-singular. Therefore if we do a gauss elimination with full pivoting we should simultaneously obtain an acceptable reordering of C, and botain C_{rc}.

In practice, it is not even necessary that C_c be non-singular. It is not uncommon for two identical constraints to be specified. The program issues a warning and continues.

Constraint transform methods do not currently support recovery of MPC forces.

The Gaussian elimination is presently being performed with a sparse package called "SuperLU," instead of a dense gaussian elimination, to speed up the time to create C_{rc}. On some platforms, e.g., sgi and dec, the blas routine dmyblas2.c in the CBLAS directory of of the SuperLU directory (need superlu-underscore-salinas.tar to create this) should be the one and only routine whose object file is placed into the SuperLU-blas library (presently called libblas-underscore-super.a) to be linked in to create the salinas executable. Failure to include this routine will cause failures of the type "Illegal value in call to DSTRV" on the above platforms, and including more than just dmyblas2.c can cause slow performance on many platforms as the SuperLU-CBLAS could override the built-in blas routines. (The built-in routines are almost always faster.)

12 Rigid Elements

Salinas will support standard *pseudo*elements for rigid bodies. These include,

- RRODs - a rigid truss like element, infinitely stiff in extension, but with no coupling to bending degrees of freedom.
- RBARS - a rigid beam, 6 degrees of freedom deleted
- RBE2 - a rigid solid. $6(n-1)$ degrees of freedom deleted, where n is the number of nodes
- RBE3 - an averaging type solid. This connects to many nodes, but removes only 6 dofs.

All of the rigid elements are stored and applied internally as MPC equations. The RBE2 is a special case of RBAR (actually just multiple instances). Note, that unlike MPC equations, these rigid elements do activate (or touch) degrees of freedom. In general, an MPC equation will not activate a degree of freedom. In the case of a rigid element however, it is necessary to activate the degrees of freedom before constraining them. Otherwise the rigid elements do not act like real elements.

Rigid elements are input into Salinas using exodus beam elements. A block entry is then provided in the input file indicating what type of rigid element is required. There is no stiffness or mass matrix entry for any type of rigid elements (only the MPC entries described above).

12.1 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.

12.2 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.

It is apparent that the last two of these constraints may be specified mathematically by requiring that the translation be the cross product of the rotation vector and the bar direction.

$$\vec{T} = \vec{R} \times \vec{L}$$

where $\vec{T}$ is the translation difference of the bar (defined as $\vec{U}_2 - \vec{U}_1$),
$\vec{R}$ is the rotation vector, and
$\vec{L}$ is the vector from the first grid to the second.

The three constraints in the cross product, together with the three constraints requiring identical rotations at both ends of the bar form the six required constraint equations.

12.3 RBE3

The RBE3 elements 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 an 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.

While the first of these constraints is easy enough to apply using multipoint constraints, the second is a little more difficult. We seek a least squares type solution.

Let $\vec{D}_i = \vec{U}_i - \vec{U}_{slave}$,
$\vec{L}_i = \vec{X}_i - \vec{X}_{slave}$

The L represent a vector from the "origin" to the point i, while the D_i represent the differential displacement of the same points. Note that the origin is at the location of the slave node, and will not in general be at the centroid of the structure.

We will use least squares to compute the rotational vector of the slave node. This is equivalent to computing a rotational inertial term and requiring a similar net rotation for the centroid.

The displacement at the centroid should be given by,

$$\vec{D}_i = \vec{R} \times \vec{L}_i$$

or, in the least squares sense we seek to minimize E.

$$E = \sum_i (\vec{D}_i - \vec{R} \times \vec{L}_i) \cdot (\vec{D}_i - \vec{R} \times \vec{L}_i)$$

Take the derivative of E with respect to a component of R, r_k.

$$\frac{dE}{dr_k} = 0 = 2\sum_i (\hat{e}_k \times \vec{L}_i) \cdot (\vec{R} \times \vec{L}_i) - \vec{D}_i \cdot (\hat{e}_k \times \vec{L}_i)$$

Now, let $R = \sum_m r_m \hat{e}_m$. We substitute for R in the previous equation to obtain,

$$\sum_m \sum_i r_m (\hat{e}_k \times \vec{L}_i) \cdot (\hat{e}_m \times \vec{L}_i) - \vec{D}_i \cdot (\hat{e}_k \times \vec{L}_i) = 0$$

Now, if we write L_i as a column vector then the expression $(\hat{e}_k \times \vec{L}_i) \cdot (\hat{e}_m \times \vec{L}_i)$ can be written as $L_i^T L_i \cdot I - L_i L_i^T$. If the sum on i is performed for the first term, we may write,

$$\sum_m r_m A_{mk} - \sum_i \hat{e}_k \cdot (\vec{L}_i \times \vec{D}_i) = 0$$

This provides three equations (one for each k) in the 3 unknowns, r_m.

The solution is found by looping once through all i to fill in the A matrix, and simultaneously to fill out the coefficients for the three equations involving D_i. Once the loop has been completed, the coefficients of R are known, and the three components of r_m can be added for each of the three equations. Each equation has 3 components of R, $2n$ components of U_i and 2 components of U_{slave} for a total of $2n + 5$ equations.

12.4 Parallel Implementation of Rigid Elements

The constraints listed above can be divided into three main groups.

global Equations that are by construction, global in nature. This includes all equations entered directly in the .inp file.

local Constraints that are specifically local in nature. This includes RBAR elements within a subdomain. Indeed, since an RBAR is entered as a pseudo element, it will exist in only one subdomain (though both nodes could be boundary nodes).

other Constraints that could be either. This includes portions of an RBE3 element for example.

This division is important for two reasons. First, the division helps us keep track of order. Second, some local processing of local constraints may be possible, which could lead to a reduction in communication during the solve stage.

To build the global constraint equations, every processor must arrive at the same sorted list of constraints. The **global** equations are easy. Every processor reads these first from the input token files. The **local** equations are also relatively straightforward. We choose an *rb_root*. Every processor sends their local constraints to this root, which concatenates them, and sends them back out. The order is unimportant. (Alternatively we could do an all-to-all and sort by processor). The **other** equations are more of a challenge, which may need to be handled on a case by case basis. Currently they are only represented by RBE3 elements.

For RBE3 elements, every processor can determine the number and rank order of the RBE3. We determine a *root* for each equation. Each processor may then send its contribution to the RBE3 to that root. Note that the data format for an RBE3 is not the same as an MPC. Finally, the root can assemble all the components of the RBE3 so the equations can be assembled. Assembled RBE3 equations can be sent to the *rb_root* for concatenation with the local equations.

The selection of the root node for this operation is not very important. Note that after the message passing, all the processors have the same data, so there is no real memory issue. Also, the computational burden is light, and the algorithm has all processors waiting for this processor anyway.

13 Notes on shell offset

These are preliminary notes... A sort of design document. They are made before any implementation.

Consider a shell offset, with an offset vector, $\vec{v}$. Notice that $\vec{v}$ could be defined at each nodal location in what follows, but for this development, we assume a single offset $\vec{v}$ which applies to all nodes. Define a coordinate system at the node, with variables u. On the offset beam the coordinate system is $\tilde{u}$.

Now, u is related simply to $\tilde{u}$. The constraint of a constant offset may be stated that the displacement difference of the two systems must be orthogonal to $\vec{v}$, i.e. $(u - \tilde{u}) = \vec{v} \times \vec{\kappa}$, where $\vec{\kappa}$ is the rotation at the nodes. Notice that the rotation is the same at both nodes.

Thus we can write,

$$\begin{pmatrix} \tilde{u} \\ \kappa \end{pmatrix} = [L] \begin{pmatrix} u \\ \kappa \end{pmatrix} \tag{22}$$

where L is a constant matrix which depends only on the geometry. We can use this transformation matrix to eliminate the degrees of freedom associated with $\tilde{u}$. The energy of the shell can be written,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} \tilde{u} \\ \kappa \end{array} \right\}^T \left[\tilde{K} \right] \left\{ \begin{array}{c} \tilde{u} \\ \kappa \end{array} \right\} \tag{23}$$

But with this substitution,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\}^T \left[L^T \tilde{K} L \right] \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\} \tag{24}$$

If we let $K = L^T \tilde{K} L$, then,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\}^T [K] \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\} \tag{25}$$

Thus, $\tilde{u}$ has been eliminated, and the equations may be rather simply put in terms of the output variables.

One of the critical issues with offsets, is how to specify them in an exodus file. We should use the attributes, but since there is no standard for what an attribute means (except that the first attribute for a shell means thickness), the tools are not well established. Dan suggested that we let the user specify which attribute defines the thickness. The end user could also specify a scaling factor, so the thickness itself might be used as the offset attribute. For example,

```
Block
   Tria3
   thickness=0.01
   off_attr=1   // use thickness as the attribute
   off_scale=-0.5
End
```

14 Notes on Consistent Loads Calculations

Starting with equation 4.1-6 from *Concepts and Applications of Finite Element Analysis* by Cook *et al.*,

$$\{r_e\} = \int_{V_e} [B]^T [E] \{\epsilon_0\} dV - \int_{V_e} [B]^T \{\sigma_0\} dV + \int_{V_e} [N]^T \{F\} dV + \int_{S_e} [N]^T \{\Phi\} dS \tag{26}$$

where each of these terms are defined in Section 4.1 of the above mentioned reference. The load vector, $\{r_e\}$, is composed of four parts in Eqn. 26. In this document, only the last part, which is the contribution of the surface tractions to the load vector, will be considered. Rewritting,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS \tag{27}$$

Here, the integral is calculated over the surface of the element on which the surface traction, $\{\Phi\}$, is applied. Therefore,

$$\{\Phi\} = [\Phi_x \Phi_y \Phi_z]^T \tag{28}$$

and $[N]$ is the shape function matrix of the element on which the surface tractions, $\{\Phi\}$, are applied. In Salinas, $\{\Phi\}$ can be applied within PATRAN by applying a spatial field to a specified side set. As a result, when calculating the load vector, this field must be accounted for. In Salinas however, this spatial field values will be available only at the nodes of the element. Using the nodal values of this surface traction, the value inside must be defined using an interpolation function over the surface or side of the element. Since only one value per node may be specified on the side set in Salinas, a surface traction may be applied only in one direction at a time. Therefore, $\{\Phi\}$ will be defined as

$$\{\Phi\} = \left\{ \begin{array}{c} n_x \\ n_y \\ n_z \end{array} \right\} \Phi(x, y, z) \tag{29}$$

14.1 Salinas Element Types

The following 3-D and 2-D elements have consistent loads implemented:

1. Hex8
2. Hex20
3. Wedge6
4. Tet4
5. Tet10
6. Tria3
7. TriaShell
8. Tria6 (four Tria3s)
9. QuadT (twor Tria3s)
10. Quad8T (1 QuadT and 4 Tria3s)

14.2 Pressure Loading

Here, we will consider only pressure loads on 3-D elements, such that

$$\{\Phi\} = \left\{ \begin{array}{c} N_x \\ N_y \\ N_z \end{array} \right\} \Phi(x, y, z) \tag{30}$$

where $[N_x, N_y, N_z]^T$ is the normal to the element face. Hence, the consistent loads can be calculated as,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS = \int_{S_e} [N]^T \Phi(x, y, z)(\vec{a} \times \vec{b}) dS_e \tag{31}$$

Here,

$$\vec{a} = [\frac{\partial x}{\partial r}, \frac{\partial y}{\partial r}, \frac{\partial z}{\partial r}]^T \tag{32}$$

$$\vec{b} = [\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s}, \frac{\partial z}{\partial s}]^T \tag{33}$$

where Φ is the pressure load, and (x, y, z) are the physical coordinate directions, and (r, s) are the local element directions for the face of the element. Notice, taking the cross-product of $\vec{a}$ and $\vec{b}$, the normal is obtained.

14.3 Shape Functions for Calculating Consistent Loads

For 3-D elements, all the faces are either quadrilateral or triangular shaped. Hence, shape functions for quads and triangles could be used to evaluate the consistent loads. If the shape functions for the 3-D elements are used, it will reduce code and "fit" better into the current finite element class structure. This is what is currently implemented. This requires a "mapping" of the 3-D elements' faces to a 2-D plane. The additional overhead for using the 3-D elements is that each face of the element must have this "mapping" which states how the elements' 3-D shape functions will map to a 2-D element. For example, for a Hex20, the element coordiantes (η_1, η_2, η_3) are defined in a particular way. For each face of the Hex20, defined by a side id, the face will have a local coordinate system (r, s). The "mapping" will define how (r, s) are related to (eta_1, eta_2, eta_3). This will also help defined what how 2-D Gauss points are mapped to the 3-D face. These mappings are done for all the 3-D elements.

15 Shell Elements

All the 2-D elements (shell elements) are based on the Tria3. The consistent loads calculations for the Tria3 can be "copied" to the TriaShell. This way all the shell elements will use the same consistent loads implementation. Since Carlos Felippa designed the Tria3, his consistent loads implementation is used. The portion for linearly varying pressure loads is shown here. If the loads are aligned along an edge, $\{q\}$, they need to be decomposed into (qs, qn, qt). Where (s, n, t) are coordinate directions along the element edge. Coordinate s varies along the element edge tangentially, n is normal to the element edge, and t is tangent to the element edge in the transverse direction, i.e., in the direction of the thickness. Once, the edge load is decomposed, the equations for consistent loads are

$$f^1{}_s = \frac{1}{20}(7q_{s1} + 3q_{s2})L_{21} \qquad f^2{}_s = \frac{1}{20}(3q_{s1} + 7q_{s2})L_{21} \tag{34}$$

$$f^1{}_n = \frac{1}{20}(7q_{n1} + 3q_{n2})L_{21} \qquad f^2{}_n = \frac{1}{20}(3q_{n1} + 7q_{n2})L_{21} \tag{35}$$

$$f^1{}_t = \frac{1}{20}(7q_{t1} + 3q_{t2})L_{21} \qquad f^2{}_t = \frac{1}{20}(3q_{t1} + 7q_{t2})L_{21} \tag{36}$$

$$m^1{}_s = m^2{}_s = 0 \tag{37}$$

$$m^1{}_n = -\frac{1}{60}(3q_{t1} + 2q_{t2})L^2{}_{21} \qquad m^2{}_n = \frac{1}{60}(2q_{t1} + 3q_{t2})L^2{}_{21} \tag{38}$$

$$m^1{}_t = -\frac{1}{40}(3q_{n1} + 2q_{n2})L^2{}_{21} \qquad m^2{}_t = \frac{1}{40}(2q_{n1} + 3q_{n2})L^2{}_{21} \tag{39}$$

where q_{s1} is the value of q in the s direction at node 1 of the edge, L_{12} is the length of the edge. The superscipts 1,2 are the node numbers of the edge. Note, it is assumed here that the load q is per unit length, but this is not assumed when creating the sideset in PATRAN for example. Therefore, this distributed load is multiplied, in Salinas, by the thickness of the triangle. Now if the pressure load is on the face of the Tria3, the equations become

$$f^1{}_x = f^1{}_y = m^1{}_z = f^2{}_x = f^2{}_y = m^2{}_z = f^3{}_x = f^3{}_y = m^3{}_z = 0 \tag{40}$$

$$f^1{}_z = (\frac{8}{45}p_1 + \frac{7}{90}p_2 + \frac{7}{90}p_3)A \tag{41}$$

$$f^2{}_z = (\frac{7}{90}p_1 + \frac{8}{45}p_2 + \frac{7}{90}p_3)A \tag{42}$$

$$f^3{}_z = (\frac{7}{90}p_1 + \frac{7}{90}p_2 + \frac{8}{45}p_3)A \tag{43}$$

$$m^1{}_x = \frac{A}{360}[7(y_{31} + y_{21})p_1 + (3y_{31} + 5y_{21})p_2 + (5y_{31} + 3y_{21})p_3] \quad (44)$$

$$m^1{}_y = \frac{A}{360}[7(x_{13} + x_{12})p_1 + (3x_{13} + 5x_{12})p_2 + (5x_{13} + 3x_{12})p_3] \quad (45)$$

$$m^2{}_x = \frac{A}{360}[(5y_{12} + 3y_{32})p_1 + 7(y_{12} + y_{32})p_2 + (3y_{12} + 5y_{32})p_3] \quad (46)$$

$$m^2{}_y = \frac{A}{360}[(5x_{21} + 3x_{23})p_1 + 7(x_{21} + x_{23})p_2 + (3x_{21} + 5x_{23})p_3] \quad (47)$$

$$m^3{}_x = \frac{A}{360}[(3y_{23} + 5y_{13})p_1 + (5y_{23} + 3y_{13})p_2 + 7(y_{23} + y_{13})p_3] \quad (48)$$

$$m^3{}_x = \frac{A}{360}[(3x_{32} + 5x_{31})p_1 + (5x_{32} + 3x_{31})p_2 + 7(x_{32} + x_{31})p_3] \quad (49)$$

where $y_{ij} = y_i - y_j$ and $x_{ij} = x_i - x_j$, A is the area of the triangle, p_i is the value of the pressure load at node i, and (x_i, y_i) are coordinates of the triangle in 2-D space.
Finally, the "pseudo" elements (QuadT, Quad8T, Tria6) created by using Tria3s require a little extra overhead. For example, the Quad8T is composed of 1 QuadT and 4 Tria3s. However, since it is defined as a Quad8T, it will have distribution factors at its 8 nodes, and these distribution factors have to be mapped to the 1 QuadT and the 4 Tria3s. The number of distribution factors will be 3 however, if the load is applied to its edge. Therefore, this extra coding can be seen in the ElemLoad method of the shells' classes.

16 Thermal Expansion loads (or initial strains)

We have user requests for a thermal expansion load. These can be used for preloads for bolts for example. I here outline the procedure. Equations reference Cook, 3rd edition.

We are really doing a statics calculation, with the loads defined from equation 4.1-6.

$$\{r_e\} = \int_{V_e} [B]^T [E]\{\epsilon_0\} dV - \int_{V_e} [B]^T \{\sigma_0\} dV + \int_{V_e} [N]^T \{F\} dV + \int_{S_e} [N]^T \{\Phi\} dS \quad (50)$$

where each of these terms are defined in Section 4.1 of the above mentioned reference. The load vector, $\{r_e\}$, is composed of four parts in Eqn. 50. In this section, only the first term is considered, as thermal loads can be thought of as initial strain, ϵ_0. This capability is somewhat complementary to the TSR (initial stress) calculation, but I'd like it generalized to solids, shells and beams.

The initial strain from thermal expansion can be written (1.7-9),

$$\{\epsilon_0\} = [\alpha_x T \ \alpha_y T \ \alpha_z T \ 0 \ 0 \ 0] \tag{51}$$

Cook states (p. 134) that it's not clear if we should average the temperature field (to be of the same order as the stress field in the element), or not. For simplicity, we average the field, and we consider only isotropic thermal expansion.

$$\{r_e\} = \alpha T \int_{V_e} [B]^T [E] [1 \ 1 \ 1 \ 0 \ 0 \ 0] \ dV \tag{52}$$

Note that this is very similar to the expression for the element stiffness matrix. Thus very few modifications are needed for the isoparametric solids.

Shells are a little trickier, as we want consistent loads. Beams are easy as the force is just αEAT in the beam direction on each of the nodes (in the element coordinate system).

The user interface is straightforward and is shown in this example.

```
Solution
  statics
End

Loads
   nodeset 1 force 1 0 0
   nodeset 2 temperature 5.0
end

Block 1
  material 1
end

Material 1
  E=10e6
  density=.1
  Talpha=1e-6
end
```

The new keywords are *temperature* and *Talpha*. The first acts much like a pressure. Of course the temperature value of "5" in the example is multiplied by the nodeset distribution factors as well, so a spatially varying load is possible. These are body loads so they should not be divided on boundaries (just like pressure again). The material parameter *Talpha* is just the temperature dependent α described above.

17 Stress/Strain Recovery

Stresses and strains are recovered at the centroids of the finite elements using standard finite element procedures. Currently, stress/strain recovery is not implemented for 1-D elements. Nodal stresses/strains are not recovered at this point. In addition, the stresses/strains calculated for shell elements are calculated in element space and not global space.

18 Coordinate Systems

Coordinate systems are provided for a number of applications including:

1. specification of boundary constraints (SPCs)
2. specification of multi-point constraints (MPCs) - not implemented
3. specification of material property rotations for anisotropic materials.
4. specification of spring directions (see section 10).

There are some applications for coordinate systems which we do NOT intend to support. These include,

1. specification of nodal locations,
2. specification of output coordinate systems.
3. specification of new coordinate systems in any but the basic system.

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.

Each coordinate system carries with it a rotation matrix. It is important to clarify the meaning of that matrix. Specifically,

$$X' = RX$$

Where X' is the new system of coordinates, R is the rotation matrix and X is the basic coordinate system. For cartesian systems, the rotation matrix

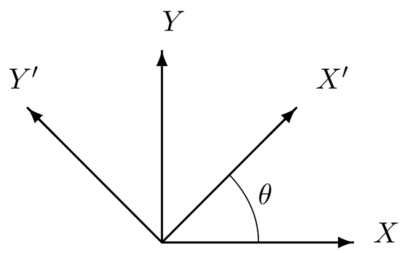

Figure 1: Original, and rotated coordinate frames

is static. Curvilinear systems will require computation of a new rotation matrix at each location in space.

The usual identity on rotation matrices applies, namely:

$$X = R^T X' \tag{53}$$

and

$$R^T R = R R^T = I$$

As an example, consider a cartesian system as shown in Figure 1.

The new system (marked by primes) is rotated θ from the old system with the new X' axis in the first quadrant of the old system. The rotation matrix is,

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

19 Constraint Transformations in General Coordinate Systems

In general, constraint equations can be applied in any coordinate system. We here describe the transformation equations and implications for general constraints in any coordinate system. The implications of this use in Salinas are also outlined.

Consider a constraint equation,

$$C'u' = Q \tag{54}$$

where the primes indicate a generalized coordinate frame. The frame may be transformed to the basic coordinate system using equation 53, and

$$u' = Ru \tag{55}$$

We can now rewrite equation 54,

$$\begin{aligned} C'Ru &= Q \\ Cu &= Q \end{aligned} \tag{56}$$

where $C = C'R$.

Thus a general system of constraint equations may be easily transformed to the basic system. Further, the rotational matrix is a 3x3 matrix which may be applied to each node's degrees of freedom separately.

19.1 Decoupling Constraint Equations

We still have a coupled system of equations. We partition the space into constrained and retained degrees of freedom, and describe the constrained dofs in terms of its Schurr complement.

$$u = \begin{bmatrix} u_r \\ u_c \end{bmatrix} \tag{57}$$

The whole constraint equation may be similarly partitioned.

$$\begin{bmatrix} C_r & C_c \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = [Q] \tag{58}$$

Note that C_r is an cxr matrix, C_c is cxc, and Q is a vector of length c. Under most conditions Q is null.

This may be solved for u_c,

$$u_c = C_c^{-1}Q - C_c^{-1}C_r u_r \tag{59}$$

We must be concerned with cases where C_c may be either singular or over constrained. The former case occurs if we try to eliminate c equations, but the rank of C is less than c. This could occur if the equations are redundant. We can over constrain the system only if Q is nonzero. Both these situations need attention, but both can be dealt with.

We may also write the solution using a transformation matrix, T.

$$\begin{bmatrix} u_r \\ u_c \end{bmatrix} = [T]\,[u_r] + \tilde{Q} \tag{60}$$

where

$$T = \begin{bmatrix} 1 \\ C_{rc} \end{bmatrix} \tag{61}$$

$$C_{rc} = -C_c^{-1} C_r \tag{62}$$

and

$$\tilde{Q} = \begin{bmatrix} 0 \\ C_c^{-1} Q \end{bmatrix} = \begin{bmatrix} 0 \\ \breve{Q} \end{bmatrix} \tag{63}$$

19.2 Transformation of Stiffness Matrix

We assume a similar partition of the stiffness matrix. The equations for statics are then,

$$\begin{bmatrix} K_{rr} & K_{rc} \\ K_{cr} & K_{cc} \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = \begin{bmatrix} R_r \\ R_c \end{bmatrix} \tag{64}$$

or,

$$[K]\,[T]\,u_r + [K]\left[\tilde{Q}\right] = R \tag{65}$$

and

$$T^T K T u_r = T^T \left\{ R - K\tilde{Q} \right\} = \tilde{R} \tag{66}$$

We can define the reduced equations,

$$\tilde{K} = T^T K T = K_{rr} + K_{rc} C_{rc} + C_{rc}^T K_{cr} + C_{rc}^T K_{cc} C_{rc} \tag{67}$$

and,

$$\begin{aligned} \tilde{R} &= T^T R - T^T \begin{bmatrix} K_{rc}\breve{Q} \\ K_{cc}\breve{Q} \end{bmatrix} \\ &= R_r + C_{rc}^T R_c - K_{rc}\breve{Q} - C_{rc}^T K_{cc}\breve{Q} \end{aligned} \tag{68}$$

The solution in the retained system is

$$\tilde{K} u_r = \tilde{R} \tag{69}$$

The system may now be solved using the reduced equations, and the constrained degrees of freedom may be solved using equation 59. Much of this is detailed in Cook, but without the constrained right hand side.

For eigen analysis the mass matrix may be transformed exactly as the stiffness matrix in equation 67. There is no force vector.

For transient dynamics the mass and stiffness matrix transform the same. The force vector and force vector corrections may be time dependent. There is currently no structure to store these time dependent terms in Salinas.

19.3 Application to single point constraints

Our initial efforts at applying single point constraints (SPC) has been limited to the basic coordinate system. In that system the equations decouple, C_c is unity and C_{rc} is zero. Then equations 67 and 68 reduce to elimination of rows and columns.

To properly account for the coupling that occurs when the constraints are not applied in the basic coordinate system, we must generate all the constraint equation on the node. This may be up to a 6x6 system. I believe that there is no real conflict in first applying constraints in the basic system, then adding additional constraints in other systems.

The process for applying constraints can be summarized as follows.

1. Generate the constraint equation in the generalized coordinate system (equation 54).

2. Transform the constraint equation to the basic coordinate system (equation 55).

3. Determine the constraint degrees of freedom. It may need to be done in concert with the next step to keep from degrading the matrix condition.

4. Compute the two transformation matrices C_c^{-1} and C_{rc} from equations 58 and 62.

5. Compute the corrections to the force vector from equation 68. We currently do not have a structure to store these corrections, except for the case of statics.

6. Compute the reduced mass and stiffness matrices from equation 67.

7. Eliminate the constraint degrees of freedom from the mass and stiffness matrix.

   In addition, for post processing,

8. store the terms of the equations necessary to recover the constraint degrees of freedom (equation 59).

A few words about post processing could also prove useful. In the first implementation of Salinas, constraints were applied only in the basic coordinate system. The degree of freedom to eliminate was obvious from the exodus file, and it's value was a constant (usually zero). Note that we re-opened the exodus file to get the required information. In this later version, a more general approach must be used. We use the following strategy.

1. degrees of freedom directly constrained to zero are handled implicitly. This is done by setting the G-set vector to zero before merging in the A-set results. There is no storage cost for this.

2. Other degrees of freedom are managed using an spc_info object. An array of these objects will be stored globally. Each object contains the degree of freedom to fill, an integer indicating the number of other terms, a list of dofs/coefficients, and a constant. This facilitates solutions of the form,

$$u_{\text{spc}} = \text{constant} + \sum_{i}^{\text{retained dofs}} u_i C_i \tag{70}$$

19.4 Multi Point Constraints

The application to multipoint constraints is very straight forward. The only difference is that the whole system of equations must be considered together. This changes the linear algebra significantly because the matrices must now be stored in sparse format. However, the steps that are applicable for single point constraints apply here as well. Section 11 deals more explicitly with MPCs.

19.5 Transformation of Power Spectral Densities

Note: The following is taken almost verbatim from Paez's book. We identify how to transform output PDS.

Let $\mathbf{H}(f)$ denote a frequency response function vector for a given input (in the global system) expressed as,

$$\mathbf{H}(f) = H_1(f)\mathbf{e}_1 + H_2(f)\mathbf{e}_2 + H_3(f)\mathbf{e}_3$$

where $\mathbf{e}_i$ represents the unit vectors of this space. Note that $\mathbf{H}(f)$ is an output vector at a single location in the model. $\mathbf{H}(f)$ can also be expressed using an alternate set of unit vectors, $\tilde{\mathbf{e}}_i$.

$$\mathbf{H}(f) = \tilde{H}_1(f)\tilde{\mathbf{e}}_1 + \tilde{H}_2(f)\tilde{\mathbf{e}}_2 + \tilde{H}_3(f)\tilde{\mathbf{e}}_3$$

Taking the dot product of these two equations and equating the results, we have,

$$\tilde{H}_1(f) = \sum_{k=1}^{3} c_{ki} H_k(f) \tag{71}$$

where

$$c_{ki} = \mathbf{e}_k \cdot \tilde{\mathbf{e}}_i$$

The spectral density function $G_{ij}(f)$ (for a given input and at a single output location) can be expressed as,

$$G_{ij}(f) = \alpha H_i^*(f) H_j(f) \tag{72}$$

where α is a constant and superscript * denotes complex conjugate. Similarly for the alternative coordinate frame,

$$\tilde{G}_{ij}(f) = \alpha \tilde{H}_i^*(f) \tilde{H}_j(f)$$

We may use equation 71 to express $\tilde{G}$ in terms of the H_i. We may then use the spectral definition in equation 72 to provide the transformation of spectral densities.

$$\begin{aligned} \tilde{G}_{ij}(f) &= \alpha \left(\sum_{k=1}^{3} c_{ki} H_k^*(f) \right) \left(\sum_{m=1}^{3} c_{mj} H_m(f) \right) \\ &= \sum_{k=1}^{3} \sum_{m=1}^{3} c_{ki} c_{mj} G_{km} \end{aligned} \tag{73}$$

This can be expressed in matrix notation as $\tilde{G} = C^T G C$.

20 History and Frequency files - in parallel

History and frequency files differ from standard exodus output in several important ways. All of these stem from the difference that there is no direct mapping from an original genesis file.

1. Node and element mappings may be quite different.
2. Some element blocks in the original may be NULL. In the extreme cases, there may be NO elements at all in the output.

3. nemesis type information is fragmented. The originals have communicators, and other information on how to join these files. The reduced files will have to extract this information for nem_join type programs to run properly.

4. Typical history files are much smaller. This can mean that some subdomains have no data for history files.

5. outputs may be used for different purposes. For example, the frf files will not typically be used for visualization, but only for frf plots.

Because of inherent difficulties in joining these files, and particularly when there are empty files on a subdomain, I propose that we do our own "join" operation on the fly. The process is outlined here.

1. A "root" node is selected. Because many of our decompositions are not well balanced, I propose selecting a processor with the least number of geometric nodes (and the greatest processor number).

2. The output file should be unique, and not depend on that root. I propose using a file name as if the root were always zero.

3. Nodes should be gathered from the different processors. However, I propose duplicating nodes on the boundaries of subdomains. This will lead to a model that cannot be run as a genesis file, (since nodes aren't equivalenced). But that should not be a problem since the .h or .frq files are not intended as genesis input. This also leads to a solution that depends (slightly) on decomposition. I don't see that as a problem. A more sticky problem is that the node number map will no longer be unique. I'm not sure if this is a problem or not. Also, the node number map will point to the implicit numbering of the genesis file, rather than the explicit numbering.

4. Elements would need to be gathered by blocks and their results concatenated. There will be complications here since some blocks will record the element type as NULL. However, these can all be overcome. The end file will again have the implicit genesis element numbers.

5. Nodal results can be concatenated easily, in an almost subdomain by subdomain fashion. There will be duplication for nodes on the boundary, and the ordering will be determined by the decomposition.

Element results will be more complicated, because the concatenation would first occur on the block level. Again, the element ordering would be partly determined by the decomposition.

6. Parallel and serial files will differ in the number of nodes and in the node and element numbering. These issues could be resolved by a fairly simple program to equivalence and add the new numbering. That would be a serial program requiring the .h file and the original genesis file as an input.

21 Random Vibration

Details of random vibration analysis are included in a number of papers[1]. These few paragraphs document what was implemented.

21.1 algorithm

The first step in the calculation is computation of Γ_{qq}, which is performed in `ComputeGammaQQ`. This is accomplished as follows.
Let the modal frequency response be defined as,

$$q_i(f) = \frac{1}{\omega_i^2 - \omega^2 + 2j\omega\omega_i\gamma_i}$$

The modal force contribution from load a is,

$$\begin{aligned} F_{i,a}(f) &= \sum_k \phi_{ik} f_k^a s_a(f) \\ &= Z_a^i s_a(f) \end{aligned}$$

where f_k^a is the k component of the force vector associated with load a, and $s_a(f)$ contains all of the frequency content of the force, but none of the spatial dependence. We have defined Z_a^i for each load that represents the sum of all the spatial contributions for mode i. It represents the frequency independent component of the force for load a.

$$Z_a^i = \sum_k f_k^a \phi_{ik}$$

[1]Reese, Field and Segalman, *A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments* Shock and Vibr. Digest, Vol. 32, No. 6, Nov 2000.

A transfer function to an output degree of freedom, k, from the input load a, may be written as a modal sum.

$$H_{ka}(f) = \sum_i F_{ia}(f) q_i(f) \phi_{ik}$$

where ϕ_{ik} is the eigenvector of mode i.

21.2 Power Spectral Density

The displacement power spectral output (at a single location) is a 3×3 matrix.

$$\begin{aligned} G_{mn}(f) &= \sum_{a,a'} H^*_{ma}(f) H_{na'}(f) \\ &= \sum_{i,j} \sum_{a,a'} F^*_{ia}(f) q^*_i(f) \phi_{im} F^*_{ja'}(f) q_j(f) \phi_{jn} \\ &= \sum_{i,j} \sum_{a,a'} q^*_i(f) q_j(f) \phi_{im} \phi_{jn} Z^i_a S^{a,a'}(f) Z^j_{a'} \end{aligned}$$

Here $S^{a,a'}(f)$ is the complex cross-correlation matrix between loads a and a'. The subscripts m and n are applicable to the 3 degrees of freedom at a single location.

By summing over the loads we may reduce the power spectral expression to a sum on modal contributions.

$$G_{mn}(f) = \sum_{i,j} \phi_{im} \phi_{jn} \mathcal{G}_{ij}(f) \tag{74}$$

where

$$\mathcal{G}_{ij}(f) = q^*_i(f) q_j(f) \sum_{a,a'} Z^i_a Z^j_{a'} S^{a,a'}(f) \tag{75}$$

Note that with the exception of the Z^i_a (which may be computed only once and are a fairly small matrix), all the terms in equation 75 are completely known on each subdomain.

21.3 RMS Output

The RMS output for degree of freedom m is given by,

$$X_{rms} = \sqrt{\int G_{mm}(f) df}$$

$$= \sqrt{\int \sum_{i,j} \phi_{im}\phi_{jm}\mathcal{G}_{ij}(f)df}$$
$$= \sqrt{\sum_{i,j} \phi_{im}\phi_{jm}\Gamma_{ij}}$$

where

$$\Gamma_{ij} = \int \mathcal{G}_{ij}(f)df$$

.

The parallel result can be arrived at by computing Z_a^i on each subdomain, and then summing the contributions of each subdomain. Note that Z_a^i contains the spatial contribution of the input force. At boundaries that interface force must be properly normalized just as an applied force is normalized for statics or transient dynamics by dividing by the cardinality of the node. Once Z has been summed, Γ_{ij} may be computed redundantly on each subdomain. The only communication required is the sum on Z (a matrix dimensioned at the number of loads by the number of modes).

The acceleration power spectral density is just $G_{mm}(\omega)\omega^4$. Section 19.5 provides details about transforming power spectra to an output coordinate system.

21.4 RMS Stress

A description of the algorithm for computation of the von Mises RMS stress is included in the reference at the beginning of this chapter. Two methods are available, but both use the integrated modal contribution Γ_{ij} as the basis for their computation. The more complete method relies on a singular value decomposition. Portions of that method are touched on below

21.5 matrix properties for RMS stress

Since $S(f)$ is Hermitian, it follows that Γ_{qq} is also necessarily hermitian. It will not in general be real. Therefore, the `svd()` must be computed using complex arithmetic. We use the `zgesvd` routine from `arpack`. The results from the `svd` of an hermitian matrix are real eigenvalues (stored in X), and complex vectors, stored in Q.

At the element level another `svd` must be performed. In this case we are computing the singular values of the matrix C.

$$C = XQ^\dagger BQX$$

where,

$$B = \Psi^T A \Psi$$

Obviously, B is symmetric. It can be shown that $Q^\dagger B Q$ is hermitian. If we examine a single element of C we can see that it contains the sum over all the terms in an hermitian matrix. That sum is necessarily real, since it can be computed by adding the lower half with it's transpose and then summing the diagonal. Let,

$$A_{ij} = \sum_{m,n} Q^*_{mi} B_{mn} Q_{nj} = \sum_{m,n} a_{ij}$$

But,

$$A^*_{ji} = \sum_{m,n} Qm, j * B_{mn} Q^*_{ni} = \sum_{m,n} Q_{nj} B_{mn} Q^*_{mi} = \sum_{m,n} a^*_{ij}$$

We therefore only need use the real `svd` routines to compute the results at each output location.

21.6 model truncation

The `svd` calculations provide the information needed for model truncation. In general, if the size of the model grows, the number of modes required for an analysis also grows. The relationship is very model dependent. However, the computational time for calculating the `svd` varies as the cube of the dimension of the matrix. Since the `svd(Γ)` is only computed once, it is not terribly important. However, the computation of each decomposition of C occurs at each output location and can significantly affect performance. In the model problem where the dimension of C was allowed to remain the same as the number of modes, increasing the number of modes from 20 to 100 changed the time for the analysis by factor of more than 100 (close to the 5^3 one might expect). Clearly, this is unacceptable especially as the desired models may have many hundreds of modes.

The `svd(Γ)` provides important information about the number of independent processes. Note that C includes the `svd` values from this calculation. We truncate by computing all the `nmodes x nmodes` terms in B, but only retaining `Cdim` columns of Q, where `Cdim` is chosen so the values of X are not too small. Thus, X[(`Cdim`)]/`X[0]` > `10`$^{-14}$. This restricts the dimension of C to a fairly small number, while retaining all components that contribute significantly to its value. As a result, the entire calculation appears to scale approximately linearly with the number of modes.

22 High Precision solutions

We've built serial salinas on DEC workstations with Real=long double, or 128 bits of precision. It was a significant effort to rebuild all the libraries to be compatible, and there are still issues and limitations.

1. We are limited to a serial solution on DEC and SGI workstations only. On DEC, libraries were built using "-real_size 128" for fortran and "-long_double_size 128" for C programs. On SGI we used the "-d16" flag for fortran. C programs need 'long double' explicitly (there is no flag to turn doubles into 128 bits).

2. complex solutions are not at all to be trusted. Thus, the direct frf solution fails. This is certainly due to blas and superlu libraries.

3. MPC elimination is not working on the SGI.

4. I suspect superlu wasn't translated properly. Unlike most of the fortran code, source code changes were required in the superlu package. Many of these changes could be accomplished with a few changes to the include files, but based on the success (or lack thereof), I still haven't got it all right.

5. Some of the regression tests fail. For example, the modaltrans test fails, but the differences are fairly small. I suspect that this is an issue of increased accuracy, but that has not been confirmed.

6. complex variables are different on the different platforms. The DEC provides for a compiler flag to generate 256 bit complex. The SGI does not.

7. Long doubles use a different print format than standard doubles. You can of course cast the long doubles to double (for output), but that does not always give the desired result. I've tried to make the required changes. On the SGI, the compiler checks for errors in print statements for stdio functions. I think I've cleaned them all up. Unfortunately, this doesn't catch error() functions. This points out the advantage in using ostream instead of stdio type functions. If I've missed some of these on output, the values will not print properly, but the code should not fail. If any have been missed on input, there is a more significant problem.

23 HexShells

Hexshells are provided to give the analyst an element with performance similar to a standard shell, but with the mesh topography of a brick. Thus, thin regions of the model can be meshed with hexshells, without concern for the bad aspect ratio of the elements, and with topography consistent with a solid mesh.

The element is documented extensively in the description by Carlos Felippa. The paragraphs in this document summarize the limitations of the shells and the possible usage.

Because hexshells have an inherent thickness direction, it is important to be able to identify that direction. There are (at least) four methods to accomplish this.

natural The *natural* ordering of the nodes in the element can determine the thickness direction. This is the method used by Carlos in developing the element. I believe that the connectivity for the element will indeed have to be modified to properly interface to his software.

sideset The placement of a sideset on one (or both) thickness faces of the elements uniquely identifies the thickness direction.

topology Usually the topology can be used to identify the thickness direction. The hexshell should be used in a sheet. If the hexshells are considered alone, only the free surfaces of the sheet are candidates for the thickness direction. Further, once the thickness direction is established for one element, it must propagate to the neighbors. (Note that this implies that we can't have a self intersecting sheet).

projection The thickness direction could be determined by the closest projection to a coordinate direction.

We will try to support all of the above methods. The *topology* method puts the least burden on the analyst. It is the least explicit however, and the most work to implement (especially in parallel). The next simplest (for the analyst) is the *projection* method. Sideset methods are burdensome for both the analyst and the code development team. The *natural* method is the easiest to implement, but can be next to impossible for the analyst to use.

Input will be structured as follows. Keywords are associated with each method. Only one of the four keywords above can be entered. If no keyword is entered, then *topology* is assumed.

```
Block 9
   HexShell
   orientation sideset='1,2'
   material=9
end
```

or,

```
Block 10
   HexShell
   orientation topology
   material=9
end
```

24 Adding a New element

At the request of Dan Segalman, I'm adding this description of what must be done when adding a new element. Much of this documentation is duplicated in the source code.

1. First, you must determine where the element ought to fit in the element inheritance heirarchy. I prefer to copy both a .H and a .C file from an element that is somewhat similar.

   Give the element a meaningful name, and have it inherit from appropriate elements. Usually this means from IsoSolid, TwoDim, OneDim, ZeroDim or Rigid, but it could be from another element.

2. Select the appropriate attributes for the element. Attributes may be input in the exodus file or in the text control file (the .inp file). Since it is harder to add these new elements to preprocessors than it is to salinas, you may want to consider carefully how to structure things so the attributes come out in the right order. For example, if your element is beam-like, and has one attribute that you may want to put in by a field in patran, you would have that attribute be first. Within patran your analysts may specify that this is a beam, and enter the attribute with a field [2]. Once the salinas control has been written, the attribute takes on the meaning of the new element.

[2] In the beam element, this would be the "area" attribute.

This sort of "trickery" on attributes is unavoidable since exodus has no way of assigning any meaning to an attribute. All we get is the attribute order.

The number of attributes and their text values are typically set in the .H file. At the element level they are then accessed by `element->Attribute(k)`.

3. Add the .H file to the included files in `Elements.H`.

4. Edit `Elements.C`. Make the following changes.
   - add the element to the identity list
   - add it to the allocation block. Make sure you keep the same order.
   - make sure MAXELEM is large enough.
   - if the element is a higher order isoparametric solid element, add its Typename() to the if-test in the ElemMass function of IsoSolid.C

5. Generate a control file, make a new copy of salinas, and test it.

25 Adding an Output Variable

It is still a bit of a chore to add an element output variable. Following the addition of element orientations, or *eorient*, I am sketching the process. The sketch will necessarily be incomplete, but may make the next addition easier.

1. The first step is to add the documentation to the user's manual, and to the theory manual if needed. I find it very important to get the input format right first. I find it deplorable to modify our user input format just so it is easy to parse.

2. Next, we must insure that the proper output can be computed. This usually involves modifications of the element routines. For example, to add `eorient` output, I needed to add a routine to `Element.[HC]`, `IsoSolid.[HC]` and to `TwoDim.[HC]`.

3. Because it is simpler, I usually test the routines using the *ECHO* options first. To do this, I did the following.

(a) add a class derived from `ElemVar`. I just copied the `ElemForce` class and added appropriate data for my element orientation. These are relatively compact classes with a lot of polymorphism, so cut and paste is not such a bad idea.

(b) insert the class at the end of the `WriteElemVars` routine. This is only about 3 or 4 lines.

(c) Modify `options1.C` and `echo.h` to include the additional variables and the associated parsing.

(d) Try to build. I think in this example, there were a few small other changes, but I can't recall the details.

(e) Modify the input, run it, and look for the results in the output. I know that in this case, I had to run the code in the debugger to see why it wasn't writing my variable. Clearly the design or process could stand improvement.

4. Next we need to get the exodus output going. Unfortunately, that is a bit more of a mess.

   (a) Edit genstuff.C. It contains sections on adding the element labels, the truth table, and returning the element variable id. All need to be updated. As part of this process, I found I had to also update the dimension of the flag passed to some of these routines. That required modification of a few more routines.

   Add element labels to the `AddExoElemLabels`, and insure that they are properly activated there. This means adding a "flag" for the new element variable. The flag must be turned on in `SetExoParams`. There is also a counter later in that routine that will need to be corrected for the new number of element variables. If you don't fix that, you'll hit an assert when you execute (so that's easy at least). Also in this routine, you have to add the logic for modifying the element variable truth table.

   (b) I had to edit output.C as well, to insure that nothing had a conflict there. This even though I wasn't yet adding history file output (see below). These are in `OutFileBlock::ExoElemVar`.

   (c) Modify the input, run it, and debug it. I'm probably forgetting a fair amount here.

5. Finally the history files need attention. This is currently a huge mess. I ran into problems with the spell checker, but the biggest problems

included incompatibilities in the truth table and the element variable output. This code currently fails in parallel runs. It is really in disarray. I'd like to see the code combined in some way with what we do in genstuff (which works OK by the way).

(a) You'll have to make sure the element variable is included in the switch statement in `OutFileBlock::ElemVar`.

(b) You must make sure the index gets right in `OutFileBlock::ExoElemVar`.

While the process is a bit too complex, it did take less than a day to add all of the code and documentation for the element orientation output.

26 GasDmp

Comments on implementation by Troy Skousen. July 1, 2003.

The version that is in the code that I added is equation 10 in the memo.

$$\Gamma = -\frac{1}{U}\int_{-W/2}^{W/2}(p[x]-p_\infty)dx = \mu\left(\frac{W}{G}\right)^3\left(1+\frac{6\Lambda}{G}\right)^{-1}\left\{1+6\eta\left(\frac{G}{W}\right)+12\zeta\left(\frac{G}{W}\right)^2\right\} \tag{76}$$

It integrates the pressure along the width of the beam resulting in force per unit length, Γ. Multiply that result by the length of the portion of the beam corresponding to the GasDmp element to obtain force. The inputs required to evaluate equation 10 are as follows:

Memo Name	Code Name	Description
W	W	Beam Width
dL	dL	Considered Length of Beam
m	mm	Molecular Mass of Gas
p_{ref}	P0	Reference pressure
T	T	Gas Temperature
μ_{ref}	muRef	Reference Viscosity
T_{ref}	TRef	Reference Temperature
ω	ww	Viscous Temperature Exponent

The constant values are:

Memo Name	Code Name	Description
k_B	kb	Boltzmann Constant
σ	sigma	Coefficient of Tangential Momentum
π	pi	Pi

I was told that the value for the coefficient of tangential momentum should be 1 for now.

With some of these values the following are evaluated.

Memo Name	Equation	Code Variable	Description
ρ	mp/k_BT	rho	Gas Density
μ	$\mu_{ref}(T/T_{ref})^{\omega}$	mu	Gas Viscosity
$\bar{c}$	$(8k_BT/\pi m)^{1/2}$	cbar	Mean Molecular Speed
λ	$2\mu/\rho\bar{c}$	lambda	Mean free path
Λ	$\left(\frac{2-\sigma}{\sigma}\right)\lambda$	LL	Slip Length

The following values are evaluated as well.

$$\eta = \frac{0.63393 + 3.23135(\Lambda/G) + 1.78154(\Lambda/G)^2}{1 + 1.17621(\Lambda/G)}(\text{correlation}), \quad (77)$$

$$\zeta = \frac{0.44525 + 1.84421(\Lambda/G) + 0.90995(\Lambda/G)^2}{1 + 0.86502(\Lambda/G)}(\text{correlation}). \quad (78)$$

In the code these values are eta and xsi respectively. See the memo for a description of what they are.

Values obtained from Salinas:

Memo Name	Code Name	Description
	du0	Relative Displacement
U	v	Relative Velocity

The relative displacement is used to find the total gap,

$$G = G0 + du0 \quad (79)$$

where $G0$ is the initial gap.

With this information equation 10 from the memo can be calculated.

The result from this is multiplied by dL to get the force value.

Notes:

At this point the entire length of the beam is considered with one GasDmp element, but after a proper mesh of the beam is made this should change to be the portion of the length of the beam that is considered in association with a GasDmp element. This equation is only good for meshes that have one element across the width of the beam.

For meshes with more than one element across the width, equation 9 from the memo should be used, multiplied by the length and width associated with a GasDmp element. The position along the width of the beam becomes a part of the equation at that point.

27 Interpolation of Direct FRF results

Charbel Farhat put out a paper that demonstrates for certain acoustics problems, a Taylor series expansion of the solutions may be made. Thus, by computing a few linear solves, you can get the derivatives of the function, and nearby solutions can be found by expansion. The trade off is that you must compute the multiple right hand sides for the derivatives, but a reformulation of the left hand side is not required.[3]

I thought that it may also be possible to do the same thing with structural response. It is true that the derivatives in a direct frf are not hard to obtain. However, as the graphs below demonstrate, the resonance is too sharp to be recovered by a power expansion.

Several things were attempted. I tried computing the power series by itself. This was then expanded in a Taylor series. With 8 terms in the expansion, we were clearly not converging. I next tried looking at interpolating using the known values of the derivatives.

I examined retaining only the first derivative, and retaining both the first and second derivatives. The results were interesting, and retain some of the features of the full solution. As shown in figure 2, the real values of the function have more fidelity than the coarse solution. They also have the proper shape. Likewise, the imaginary parts of the function have the right general features (as shown in figure 3). However, it is also clear from both of these figures, that the interpolated function cannot truly follow the fully sampled function. This becomes more obvious in figure 4, where errors in the relative phase term results in a noticable dip in the function where a peak is expected.

Charbel's work studied the acoustic response due to a plane wave reflecting from a perfectly rigid reflector. The acoustic field must have a nonreflecting boundary condition. These are the conditions required to have no resonance in the field. His Taylor series approximation was adequate for the sinusoidal variations required in this special case problem.

[3] Djellouli, R., Farhat, C. and Tezaur, R. "A Fast Method for Solving Acoustic Scattering Problems in Frequency Bands", J. Comp. Physics, **168**, 412, 2001.

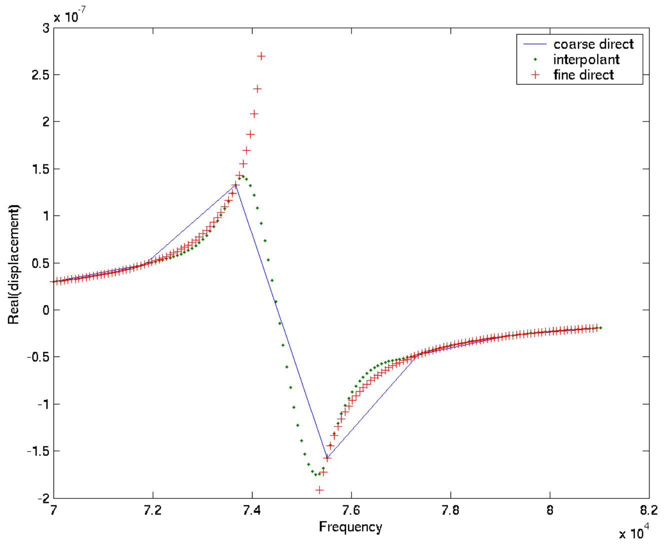

Figure 2: Real part of Solution

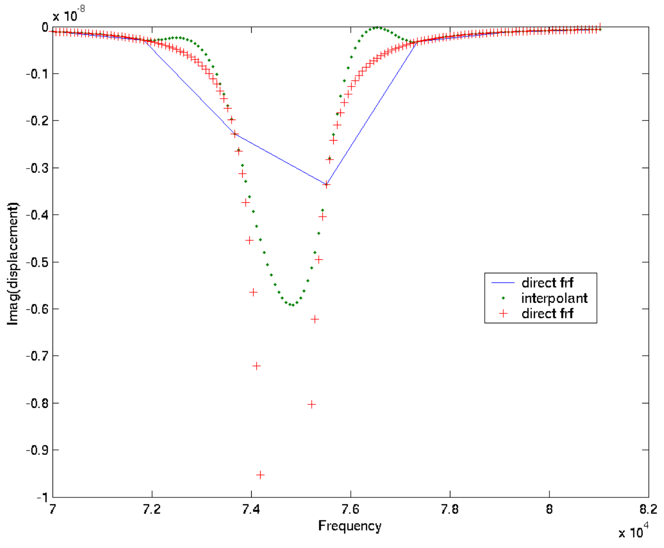

Figure 3: Imaginary part of Solution

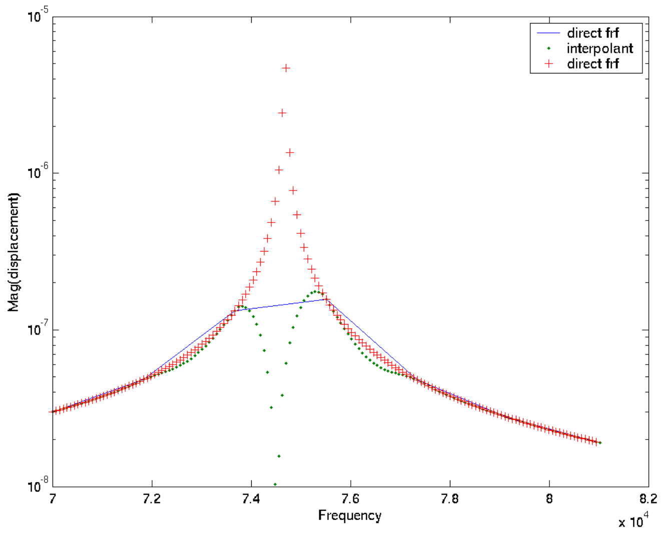

Figure 4: Magnitude of Solution

A better functional for representing these resonant conditions is a rational function such as those used by Padé.

$$U(x) = \frac{\sum_{i=0}^{M} a_i x^i}{\sum_{j=0}^{L} b_j x^j} \tag{80}$$

where $b_0 = 1$ and other terms are selected to meet the boundary conditions.

By maintaining continuity of the function and 2 derivatives at each end, there are six (complex) parameters. Initial studies with $M = 0$ and $L = 5$ were very successful in matching the resonant response, but unsuccessful at the zeros of the function. Experiments with $M = 2$ and $L = 3$ provide excellent agreement with the direct response terms. At the peaks, and at the zeros, the functions are in agreement to more than 7 digits. See Figure 5

This method also has potential to provide an error indicator. In theory, increasing the order of the derivative (and hence the order of the rational function) should increase the accuracy of the solution. Thus, the difference between a 6 term (2nd derivative) and a 4 term (1st derivative) match can be easily and inexpensively computed. Unfortunately, this difference is only an error *indicator*, it is not an estimator. There is no mathematics to prove

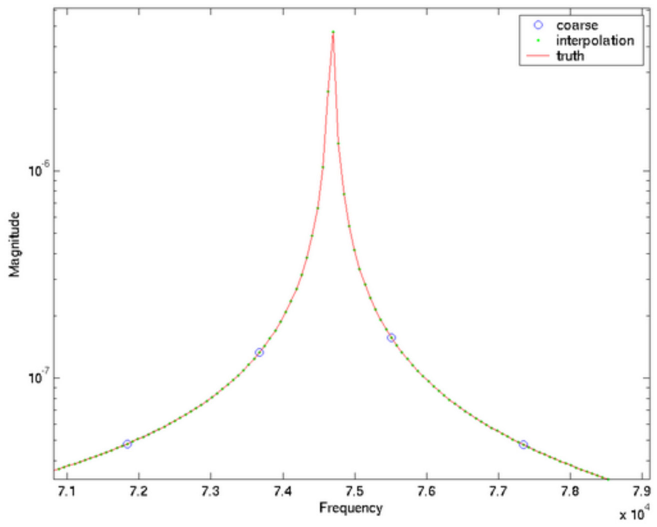

Figure 5: Interpolation Using Rational Functions

that the real error will be bounded by this indicator.

27.1 Rational Function Interpolation - $C1$

We present the mathematics behind a rational function interpolation in $C1$, i.e. continuity of the function and 1st derivatives. We describe the function using a rational function with $L = 1$, and $M = 2$.

$$u(x) = \frac{a_0 + a_1 x}{1 + b_1 x + b_2 x^2} \tag{81}$$

Multiplying by the denominator and differentiating we have,

$$(b_1 + 2b_2 x)\, u + \left(1 + b_1 x + b_2 x^2\right) u' = a_1 \tag{82}$$

Here the primes indicate the derivative with respect to x.

We obtain 4 equations in 4 unknowns by evaluating these two equations at $x = 0$ and $x = h$.

$$\begin{aligned} u_0 &= a_0 & (83)\\ b_1 u_0 - a_1 &= -u_0' & (84)\\ b1 h u_h + b_2 h^2 u_h - a_1 h &= a_0 - u_1 & (85)\\ b1(u_h + h u_h') + 2b_2 h u_h + b_2 h^2 u_h' - a_1 &= -u_1' & (86) \end{aligned}$$

Where $u_0 = u(0)$, $u'_0 = \frac{du}{dx}|_{x=0}$, and similar expressions hold at $x = h$.

The first equation is trivial. Substituting u_0 for a_0 in the remaining equations, we arrive at a simple $3x3$ expression for the coefficients.

$$\begin{bmatrix} u_0 & 0 & -1 \\ hu_h & h^2 u_h & -h \\ u_h + hu'_h & 2hu_h + h^2 u'_h & -1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} -u'_0 \\ u_0 - u_h \\ -u'_h \end{bmatrix} \tag{87}$$

These equations are solved at each degree of freedom in the model, at each coarse frequency step. With the coefficients known, intermediate points may be interpolated using equation 81.

27.2 Rational Function Interpolation - $C2$

We present the mathematics behind a rational function interpolation in $C2$, i.e. continuity of the function and 1st and 2nd derivatives. We describe the function using a rational function with $L = 2$, and $M = 3$.

$$u(x) = \frac{a_0 + a_1 x + a_2 x^2}{1 + b_1 x + b_2 x^2 + b_3 x^3} \tag{88}$$

Multiplying by the denominator and differentiating we have,

$$\left(b_1 + 2b_2 x + 3b_3 x^2\right) u + \left(1 + b_1 x + b_2 x^2 + b_3 x^3\right) u' = a_1 + 2a_2 x \tag{89}$$

$$\begin{aligned} (2b_2 + 6b_3 x)\, u + 2\left(b_1 + 2b_2 x + 3b_3 x^2\right) u' + & \\ \left(1 + b_1 x + b_2 x^2 + b_3 x^3\right) u'' &= 2a_2 \end{aligned} \tag{90}$$

Again the primes indicate the derivative with respect to x.

We obtain 6 equations in 6 unknowns by evaluating these three equations at $x = 0$ and $x = h$.

$$u_0 = a_0 \tag{91}$$

$$b_1 u_0 - a_1 = -u'_0 \tag{92}$$

$$2b_2 u_0 + 2b_1 u'_0 + u''_0 = 2a_2 \tag{93}$$

$$(1 + b_1 h + b_2 h^2 + b_3 h^3) u_h = a_0 + a_1 h + a_2 h^2 \tag{94}$$

$$\begin{aligned} (b_1 + 2b_2 h + 3b_3 h^2) u_h + & \\ (1 + b_1 h + b_2 h^2 + b_3 h^3) u'_h &= a_1 + 2a_2 h \end{aligned} \tag{95}$$

$$\begin{aligned} (2b_2 + 6b_3 h) u_h + 2(b_1 + 2b_2 h + 3b_3 h^2) u'_h + & \\ (1 + b_1 h + b_2 h^2 + b_3 h^3) u''_h &= 2a_2 \end{aligned} \tag{96}$$

Where $u_0 = u(0)$, $u_0' = \frac{du}{dx}|_{x=0}$, and similar expressions hold at $x = h$.

Again, the first equation is trivial. Substituting u_0 for a_0 in the remaining equations, we arrive at a $5x5$ matrix expression for the coefficients.

$$\begin{bmatrix} u_0 & 0 & 0 & -1 & 0 \\ 2u_0' & 2u_0 & 0 & 0 & -2 \\ hu_h & h^2 u_h & h^3 u_h & -h & -h^2 \\ u_h + hu_h' & 2hu_h + h^2 u_h' & 3h^2 u_h + h^3 u_h' & -1 & -2h \\ 2u_h' + hu_h'' & A_{52} & A_{53} & 0 & -2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -u_0' \\ -u_0'' \\ u_0 - u_h \\ -u_h' \\ -u_h'' \end{bmatrix} \tag{97}$$

Where $A_{52} = 2u_h + 4hu_h' + h^2 u_h''$, and $A_{53} = 6hu_h + 6h^2 u_h' + h^3 u_h''$.

These equations are solved at each degree of freedom in the model, at each coarse frequency step. With the coefficients known, intermediate points may be interpolated using equation 88.

27.3 Derivatives of Frequency Dependent Dynamic Matrix

A necessary key to computation of the interpolant is the derivative of the dynamic matrix with respect to frequency, ω. We consider here only the system where the coefficient matrices are constants (i.e. we do not consider viscoelasticity). The dynamic matrix is a combination of the stiffness, damping and mass matrices.

$$\left(K + i\omega C - \omega^2 M\right) u(\omega) = f(\omega) \tag{98}$$

$$\text{or} \tag{99}$$

$$A(\omega)u(\omega) = f(\omega) \tag{100}$$

For convenience, we define variables for the first and second derivatives of A with respect to ω.

$$A' \equiv \frac{dA}{d\omega} = iC - 2\omega M \tag{101}$$

$$A'' \equiv \frac{d^2 A}{d\omega^2} = -2M \tag{102}$$

The derivatives up to 6^{th} order in ω are tabulated in table 1. Note that in each case, determination of the derivative requires solution of the same matrix with a right hand side containing only lower order derivatives. In the table, $u^{(n)} = d^n u / d\omega^n$.
where $c_n = c_{n-1} + (n-1)$.

Table 1: Nth Derivatives of Dynamic Solutions

Order	Functional
1	$Au^{(1)} = -A'u + f^{(1)}$
2	$Au^{(2)} = -A''u - 2A'u^{(1)} + f^{(2)}$
3	$Au^{(3)} = -3A''u^{(1)} - 3A'u^{(2)} + f^{(3)}$
4	$Au^{(4)} = -6A''u^{(2)} - 4A'u^{(3)} + f^{(4)}$
5	$Au^{(5)} = -10A''u^{(3)} - 5A'u^{(4)} + f^{(5)}$
6	$Au^{(6)} = -15A''u^{(4)} - 6A'u^{(5)} + f^{(6)}$
n	$Au^{(n)} = -c_n A''u^{(n-2)} - nA'u^{(n-1)} + f^{(n)}$

28 Triangular Shell Element

The triangular shell element (TriaShell) is derived as follows. The bending d.o.f. (w, θ_x, θ_y) and the membrane d.o.f. (u, v, θ_z) are decoupled. The idea is to obtain the membrane response using Allman's triangle and the bending response using the discrete Kirchoff triangular (DKT) element.

28.0.1 Allman's Triangular Element

Using the formulation given in Ref. 1 and replacing $\cos(\gamma_{ij}) = \frac{y_{ji}}{l_{ij}}$ and $\sin(\gamma_{ij}) = \frac{-x_{ji}}{l_{ij}}$, we get

$$u = u_1\psi_1 + u_2\psi_2 + u_3\psi_3 + \frac{1}{2}y_{21}(\omega_2 - \omega_1)\psi_1\psi_2 + \frac{1}{2}y_{32}(\omega_3 - \omega_2)\psi_2\psi_3 + \frac{1}{2}y_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{103}$$

$$v = v_1\psi_1 + v_2\psi_2 + v_3\psi_3 + \frac{1}{2}x_{21}(\omega_2 - \omega_1)\psi_1\psi_2 - \frac{1}{2}x_{32}(\omega_3 - \omega_2)\psi_2\psi_3 - \frac{1}{2}x_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{104}$$

The stiffness and mass matrices $([K]_{AT}, [M]_{AT})$ are found using general finite element procedures. Unfortunately, a mechanism exists for this element if the deformations are all zero and the rotations are all the same value. Cook *et al.*[2] have a "fix" for this which has been implemented to avoid undesirable low energy modes produced by this mechanism.

28.0.2 Discrete Kirchoff Element

As for the DKT[3] element, things are not so simple. The nine d.o.f. element is obtained by transforming a twelve d.o.f. element with mid-side nodes to a triangle with the nodes at the vertices only. This is obtained as follows. Using Kirchoff theory, the transverse shear is set to zero at the nodes. And the rotation about the normal to the edge is imposed to be linear. Using these constraints, a nine d.o.f. bending element is derived (DKT) using the shape functions for the six-node triangle. Unfortunately, the variation of w over the element cannot be explicitly written. Therefore, the w variation over the element needs to be calculated before the mass matrix can be obtained.

As stated, the equation for w is not explicitly stated over the element in the derivation by Batoz *at al.*. Using a nine d.o.f. element, a complete cubic cannot be written, since 10 quantities would be needed to get a unique polynomial. The strategy taken here is that the stiffness matrix produced using for the DKT element provides reasonable results, and the derivation of the mass matrix is not as critical. So, the equation for w is taken from Ref. 4, as

$$w = \alpha_1\psi_1+\alpha_2\psi_2+\alpha_3\psi_3+\alpha_4\psi_1\psi_2+\alpha_5\psi_2\psi_3+\alpha_6\psi_3\psi_1+\alpha_7{\psi_1}^2\psi_2+\alpha_8{\psi_2}^2\psi_3+\alpha_9{\psi_3}^2\psi_1 \tag{105}$$

For the AT and DKT elements, the stiffness and mass matrices are derived with the help of Maple. The consistenet mass matrix is derived using "normal" finite element procedures. If a lumped mass matrix is requested then the mass matrix terms associated with the translation d.o.f. are found in the "normal" sense. However, mass matrix terms for the rotational d.o.f. are set to $\frac{1}{125}$ of the translation terms.

In summary, the code has been written which uses the AT and DKT element use in combination as a shell element. The stiffness matrices are calculated without complication. The mass matrix for the AT element is also derived without complication. The mass matrix for the DKT element is derived using an incomplete polynomial, but the results obtained should not be effected very much.

28.0.3 Verification and Validation

The AT element is verified by comparing calculated results with the results published by Allman in Ref. 1. The square plate in pure bending and a cantilvered beam with a parabolic tip load are used as verification examples.

DOF	AT/DKT	ABAQUS	AT/DKT!
x	0.000	0.000	0.000
y	0.000	0.000	0.000
z	-1.405 $\times 10^{-2}$	-1.398 $\times 10^{-2}$	-1.398 $\times 10^{-2}$
θ_x	3.337 $\times 10^{-2}$	3.337 $\times 10^{-2}$	3.337 $\times 10^{-2}$
θ_y	3.106 $\times 10^{-2}$	3.089 $\times 10^{-2}$	3.089 $\times 10^{-2}$
θ_z	0.000	0.000	0.000

Table 2: Comparison of deflections at Node 2

DOF	AT/DKT	ABAQUS	AT/DKT!
x	0.000	0.000	0.000
y	0.000	0.000	0.000
z	1.949 $\times 10^{-2}$	1.955 $\times 10^{-2}$	1.955 $\times 10^{-2}$
θ_x	3.363 $\times 10^{-2}$	3.363 $\times 10^{-2}$	3.363 $\times 10^{-2}$
θ_y	-2.686 $\times 10^{-2}$	-2.702 $\times 10^{-2}$	-2.702 $\times 10^{-2}$
θ_z	0.000	0.000	0.000

Table 3: Comparison of deflections at Node 3

The mass matrix is not verified except to note that the mass is conserved in the u, v directions.

The DKT element is validated by using the experimental data published by Batoz *et al.* in Ref. 3 for a triangular fin. The first 10 eigenvalues for the triangular fin (cantilever) match very well. In addition, the DKT element is verified by using a cantilevered beam and matching deflection results at the tip. If $\nu = 0$, then results should match very closely with Euler-Beam theory results, and they did.

Finally, the AT/DKT element is verified by comparing with published results from Ref. 5. Tables 2 and 3 show that our elements match exactly with ABAQUS to the number of digits shown. The first column is the result produced by Ertas *et al.*, the second column is the result produced by ABAQUS, and the third column is the result produced by SALINAS using this DKT/AT element.

29 Using the Test Problem Implementation Tool

The Test Problem Implementation Tool (TEPIT) was created to allow code developers to easily add test problems to the test suite for Salinas. These problems will be tested in serial and/or parallel depending on the information given.

29.1 Quick Start

For a quick start, change directories to 'test_tool' within Salinas. Create a link using 'ln -s Makefile make.??'. The file you link with will depend on which machine you are running the test. For most cases, link with make.all. After which, type 'make fresh_start'. Next, change directories to 'test_problems1', type 'TestSuite ??'. Currently, the choice are 'serial', 'parallel', or 'parallel_sgi'. This assumes that an appropriately complied salinas exists, otherise the path to the salinas executable is also needed when running 'TestSuite'.

29.2 Creating and running a test problem

To run a test problem, an input file for the test problem is needed. This file must have a suffix of '_test'. For example, in the test_tool/test_problems1/test_example1 subdirectory, a test input file, beam0_test, exists. The contents of the file are:

```
1: begin Beam0_Test
2:   input_file    beam.inp
3:   exodus_input_file beam.exo
4:   dispx   5  7.1446e-03  1
5:   dispx   12 1.6418e-02  0.001
6:   sstressx1  100 -2.8175e+03 2%
7:   dispx   1   0    0.005%
8: end
```

The line numbers are added for this document only.

```
Line 1:  begin Beam0_Test
```

All input files must have a 'begin' keyword followed by the name of the test.

```
Line 2:  input_file  beam.inp
```

The 'input_file' keyword followed by the filename specifies the Salinas input file to be used in this test. This Salinas input file must have some changes made to it to be part of the test suite. The 'file' block, e.g. in beam.inp, must look as follows:

```
File
        NUMRAID
        geometry_file   'FILEPATH'
end
```

This is done so that the test problem can be run on various systems with little difficulty. Since the utility 'grope' is used to determine passes and failures, and since grope only is used with Exodus type files, there is no need to have any options on which will not result any changes to the Exodus output file. E.g., the entire 'ECHO' block isn't needed. Therefore it is recommended that all the options in 'ECHO' and 'OUTPUTS' be turned off if not needed for the test problem.

```
Line 3: exodus_input_file beam.exo
```

The 'exodus_input_file' is needed to specify the Exodus file which will be used with the input file.

```
Lines 4-7:  keyword node#/element#/eigenvalue# value tolerance
```

This format is defined as follows. The 'grope' utility will search for nodal or element variables defined by 'keyword'. It will then use the value found at the node#/element#/eigenvalue# to see if it matches the value within the given tolerance. If so, the test passes, if not, the test fails. All the possible values for 'keyword' are all the elemental or nodal variables that can be output in Salinas, with one exception. The exception is the 'eigenvalue' keyword. This is a recognized keyword, but is NOT a elemental or nodal variable. Note: the node#/element#/eigenvalue# start at 1 for their indexing and not 0.

```
line 8:  End
```

This is required to specify the end of the test problem input file.

To run the example test, go to the test_problems1 subdirectory. To check that all the test input files are correct, run the script 'FileCheck'. This will

create a file 'Log_Test_Suite' which will contain the output of 'FileCheck'. To check only one test problem input file and execute it using salinas, e.g. beam0_test located in test_example1 subdirectory, use the utility 'TestOnlyOne test_example1
beam0_test'. Finally, to run the entire set of test suite problems within the test_problems1 subdirectory, run the script 'TestSuite serial'. The 'serial' keyword could be 'parallel' or 'parallel_sgi'. This will depend on which machine you are on and which version of salinas needs to be tested. To clean up all the temporary files created, run the script 'clean_up' when done.

29.3 Adding a Test Problem to the CVS Repository

For a serial test, the following files are needed:

```
Salinas Input File
Exodus Input File
```

For a parallel test, the additional information is needed:

```
Nemesis Load Balance File
Number of Processors
```

The parallel information for the example given above, can be added to the test problem input file as follows:

```
  load_balance_file  beam.nem
  num_procs          4
```

With the load balance file and number of processors information given, this test is a serial and a parallel test. In other words, if all of the information for a test problem is given, the test problem will be considered a parallel and a serial test problem. However, there might be a need to label a test as a parallel test only. Therefore, the keyword 'parallel_only' is a recognized keyword which will only allow the test to be run on a parallel machine. For an example of a serial and parallel test, see the test_example2 subdirectory located in test_tool/test_problems1 subdirectory of Salinas.

Now that the required files for a serial and/or parallel test are known, simply 'cvs add' the files to the repository and then 'cvs commit' them. However, when adding binary files such as Exodus and Nemesis files, the '-kb' flag must be used when adding the files. For example, to add the example test problem in test_example1 subdirectory:

```
prompt> cvs add beam.inp beam0_test
prompt> cvs add -kb beam.exo
prompt> cvs commit beam.inp beam0_test beam.exo
```

29.4 Explanation of Various Files

To run the set of test problems, change directories to test_problems1 in the test_tool subdirectory and type 'TestSuite serial' or 'TestSuite parallel' or 'TestSuite parallel_sgi'. This will run the tests and print out results in the file 'Log_Test_Suite' and print to stdout the number of passes and failures.

The 'TestSuite' script runs the following pieces of scripts/executables:

```
1: clean_up (Shell script)
2: begin (C++ executable)
3: createdirs_script (Shell script created by 'begin')
4: readtest (C++ executable)
5: the_script (Shell script created by 'readtest')
6: final (C++ executable)
7: the_script1 (Shell script created by 'final')
8: qsubit (Shell script run only when doing parallel tests, but not
parallel_sgi runs)
```

1: The script 'clean_up' cleans up any temporary files, lingering Exodus II output files, etc., that might still exist from a previous test run.
2: 'begin' creates temp files(for recording passes and failures), a Log file (for recording various information during the test), and starts 'the_script' file for later use by 'readtest'. It also creates a script 'createdirs_script' which will create necessary directories for running on janus, the ASCI Option Red Supercomputer.
3: 'createdirs_script' creates /pfs_grande/tmp_1/$USER directories on janus, the ASCI Option Red Supercomputer, for running parallel tests. Also used when running parallel_sgi tests on Atlantis.
4: The C++ code 'readtest' parses the test problem input file and creates the necessary Salinas input file, nem_spread input file (if needed), and nem_join input file (if needed). It appends to 'the_script' for running salinas, nem_spread, and nem_join for each test problem.

5: 'the_script' runs the actual code salinas, and any other codes necessary to set up the necessary files, i.e. nem_spread and nem_join. It also runs the utility 'grope' to extract the information it needs to determine a pass or failure. After 'grope' is executed, 'the_script' will execute 'compare_values' to compare the expected and actual values to determine if they match within the specified tolerance. The results will be written into 'Log_Test_Suite' and the temporary passes and failures file.
6: 'final' simply outputs to stdout the total number of passes and failures.
7: 'the_script1' created by final will clean up certain temporary files if there are no failures. Otherwise, it does nothing.
8: 'qsubit' created by 'readtest' will submit 'the_script' script to the snl.day queue to be run. An e-mail will be sent to the submitter when the job is done. This will run all the parallel test problems when doing parallel tests on the ASCI Option Red Supercomputer.

30 Offset Beams and Shells

September 28, 2000. Garth Reese

Beams are rather unique in that everything for beams is solved in the beam coordinate systems, and they are then transformed back to the basic system. I think we can easily incorporate the offset into the beam formulation. But, in what follows, I believe that the offset would apply equally well to offset shells, provided only that there is only one offset vector which is applied uniformly to all the nodes on the element.

Consider two coordinate systems, X and X' where the prime denotes a coordinate system actually on the element, and the unprimed denotes an offset coordinate. The strain energy is represented naturally in the primed coordinate.

$$E_{strain} = \frac{1}{2} U'^T K' U'$$

Now the transformation from the primed coordinate to the unprimed coordinate is a simple linear coordinate transformation, i.e. $U' = LU$. Also, we require that the strain energy be invariant to the transformation. Thus,

$$E_{strain} = \frac{1}{2} (LU)^T K' (LU)$$

or

$$E_{strain} = \frac{1}{2} U^T K U$$

where K is defined as $K = L^T K' L$.

A similar relation can be worked out for the mass matrix of the element. The velocity transforms with the same transformation relation as the displacement. Since the kinetic energy must be invariant, we end up with an unprimed mass matrix, $M = L^T M' L$, where M' is the mass matrix in the primed system. Note that for lumped masses, we should be able to check this relation against the parallel axis relation, $I = ml^2$.

30.1 Stress and Strain Recovery

Once the equations of motion have been solved, the displacements in the unprimed system are determined. These may be transformed directly to the primed system using $U' = LU$. Standard tools can be used to determine both stress and strain in the primed system, which is the appropriate response to report. Of course, for beams we report neither stress nor strain, but this should apply to any type of offset element.

30.2 Coordinate Tranformation $U' = LU$

This transformation is purely geometric. For almost all elements, it is required that the offset vector be normal to the element. In what follows, that is assumed. Each node pair contributes a 6x6 block to the transformation matrix L. The general transformation for single node pair is,

$$\begin{pmatrix} u'_t \\ u'_r \end{pmatrix} = \begin{pmatrix} I & R \\ 0 & I \end{pmatrix} \begin{pmatrix} u_t \\ u_r \end{pmatrix}$$

Thus, rotations are directly tied between the systems and translations are the sum of the translational terms and the rotational components. We can show that $u'_t = u_t - O \times u_r$. Here O is the offset vector and $\times$ is the cross product, as shown in the figure.. Obviously, only the submatrix R is of much interest. It could either be stored or recomputed each time it is needed.

30.3 Force and Pressure Loading

Loadings applied directly to the element are properly applied in the primed coordinate system. They must be transformed back to the unprimed system. This is the inverse transform. The force on the unprimed system requires a a transformation similar to the displacements. Specifically,

$$F_t = F'_t - RF'_r$$

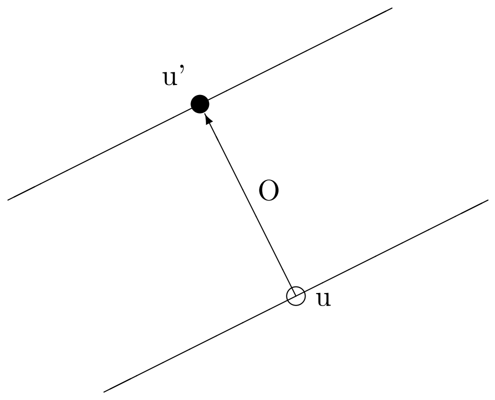

Figure 6: Offset Nodal Pair Geometry

where R is defined above, and,

$$F_r = F_r'$$

30.4 Implementation Details

We have several options in offsets.

1. We could require all elements to have an offset, but provide a default of 0. This would require us to add the attributes to the exodus options and the input file. It would require addition of a transformation matrix to the element. This could be implemented at the element level and percolate down to all elements. There is some need at the TwoDim level, a strong need at the OneDim level, but no need at the isosolid level.

2. We could implement offsets for beams only. They would incur the extra overhead of the offset, but no other element would.

3. We could generate a offset-beam. Then only beams that have offsets would have the overhead. This type of beam could inherit from standard beam formulations.

4. We could do some tricks so all 1D and 2D elements have the option of offsets, but they are turned on selectively. This would mean some funky things in constructors, and a variable number of attributes, i.e. the number of attributes for an element will depend on whether it has an offset or not. Since that is not known at the time of the constructor, it could be tricky.

I don't like 1 because of the extra overhead. Options 2 and 3 are rather restrictive. I think that we can implement option 4 without too much trouble. We would use the standard constructor, but add to the constructor a method to check for offsets. If the keyword "offset" is found in the input, or if the number of attributes is right for offsets, then they will be used. There could be a general element level option to check on offsets. In fact, most of the algorithms should be written at the element level so they are inherited for all elements. This would get around the offset shell problem that Howard has been wanting for some time.

Note that offsets MUST be supported both in the .inp file and in the exodus file. They will have to be added to the translators. This is because in a cylindrical geometry the offsets of the beam must vary as we go around the structure. Note also that offset shells in nastran are not vectors, but only a signed float. They must be normal to the surface. Only the normal component would be used in Salinas, but it seems that the offset should be specified the same for all types of elements. Thus, there will inevitably be complications in the translator.

Another issue is just what must be stored. Clearly there must be access to the offset, as attribute type data. However, it is pretty straightforward to compute a normal vector from the offset, and the rotation matrix R is easily computed. Thus, I don't see a need to store this matrix. The terms of the R matrix are simply,

$$R_{ii} = 0$$
$$R_{xy} = O_z$$
$$R_{xz} = -O_y$$
$$R_{yz} = O_x$$

And, R is antisymmetric.

It would also be nice if there is a way to save the normalized terms in the attributes of the element. Then, further calculations of the normalization need not be performed.

30.5 Test Cases

1. An offset beam by itself must have the same eigen properties as the un-offset structure. Use beam_100.exo for this.

2. Build two models, one with offset beams, one with MPC structures to accomplish the same thing. They should have identical mass and

stiffness matrices (and identical eigenproperties as well). Ideally such models would have an arbitrary rotation. I would suggest two beams side by side, for a total of 4 elements.

3. For completeness, we should look at beam stiffeners on plates. This is done just as the previous case with an offset MPC in one case and an offset beam in the other. This would be done on a single quad and single beam, in plane.

4. We should look at a stiffening ring on a cylinder, and compare it with a stiffener ring made by a shell model. This model should be rotated arbitrarily. It would serve as the regression and coverage test for offset beams.

5. We could look at offset T stiffeners with plates. They would serve as the regression test for offset shells. We should do stress recovery on these elements. Perhaps both a modal and a gravity loaded static solution would be appropriate.

31 Thermal Structural Response

In this section we describe the governing equations and corresponding finite element formulation for structural response in the presence of a steady state temperature distribution. By 'steady state' we mean the temperature distribution is constant in time, and thus is a solution of the Laplace equation. This induces a corresponding static thermal stress field that can simply be superposed with the remaining stress field, which may be time varying.

31.1 governing equations

In the general case we have the following equations of motion

$$\rho\ddot{u}_i - \frac{\partial\sigma_{ij}}{\partial x_j} = \rho b_i \tag{106}$$

where standard index summation is used, i.e. two j's in the same term implies summation $j = 1..3$, u_i is the i^{th} component of displacement, and b_i is the i^{th} component of the body force.

Multiplying by a test function we obtain

$$(\rho\ddot{u}_i, v) - (\frac{\partial\sigma_{ij}}{\partial x_j}, v) = (\rho b_i, v) \tag{107}$$

Integrating the middle term by parts, we obtain

$$(\rho\ddot{u}_i, v) - (\sigma_{ij}, \frac{\partial v_i}{\partial x_j}) = (\rho b_i, v) \tag{108}$$

We now decompose the stress field into two parts as follows

$$\sigma_{ij} = \sigma_{ij}^0 + \sigma_{ij}(t) \tag{109}$$

where σ_{ij}^0 is the constant thermal stress, and $\sigma_{ij}(t)$ is the remaining stress, which is time-varying.

Using this decomposition, equation 108 can be written as

$$(\rho\ddot{u}_i, v) - (\sigma_{ij}(t), \frac{\partial v_i}{\partial x_j}) = (\rho b_i, v) - (\sigma_{ij}^0, \frac{\partial v_i}{\partial x_j}) \tag{110}$$

The second term on the right hand side is treated as in internal thermal force that is constant for all time, and thus need only be computed once in the simultation. Then, at each time step, this constant internal force is subtracted from the right hand side.

Notationally, the internal force term can be written as

$$(\sigma_{ij}^0, \frac{\partial v_i}{\partial x_j}) = \int_\Omega B^T \sigma_{ij}^0 \tag{111}$$

where B is the standard strain displacement matrix. Thus, this internal force can be computed on the element level simply be integrating the product of the transpose of the strain displacement matrix with the thermal stress tensor. In terms of strains, the thermal stress tensor can be written as

$$\sigma_{ij}^0 = -C_{ijkl}(\epsilon_{kl}) = -C_{ijkl}(\alpha_{kl}\Delta T) \tag{112}$$

32 File Naming Conventions

File naming conventions can be something of a religious war. We'll try to avoid that here. I will try to outline the different file names, and provide justification for the naming.

Salinas outputs two kinds of files, exodus outputs, and the `.rslt` file which is an ASCII text file. There may be up to three types of exodus files.

standard exodus output These are standard exodus files with full detail included. They will be designated by the **.exo** extension.

history output These are subsets of the exodus files generated using the history command. As a subset, they may not have all the data normally associated with an exodus file. For example, they may not have element block information. They use the `.h` extension.

frequency output These files are developed using the "frequency" block in the input. They are similar to history output, but data is written specifically in the frequency domain. They use the `.frq` extension.

Each file is built up of three parts, 1) a base file name, 2) case descriptor, and 3) the extension.

The base file name is determined by the parent file. Thus, the exodus output files all derive their base file name from the exodus parent. This means that they will all reside in the same directory as the parent. This is critically important on parallel machines where the I/O performance is optimized. The text `.rslt` file, uses the base name derived from the input text file (i.e. the `.inp` file).

The case descriptor describes the solution itself. Only exodus files have a case descriptor part to the file name. In multicase solutions, the case descriptor will match the case descriptor in the input. This allows us to write more than one exodus file for a multicase solution. For single case solutions, the case descriptor will be "out". The case descriptor is always preceeded by a dash "-". For example, `junk-eig.exo` has a case descriptor of "eig".

The extension indicates the file type. The extensions are described above, and will be "exo", "h", "frq" or "rslt".

32.1 File Locations

In a serial solution, it is expected that the files will all be located in the same working directory. While that is not required, it is generally a good idea. However, in parallel systems, this cannot always be readily accomplished because of the need to store the parallel exodus files on multiple disk RAIDS. The files will be stored in the following directories.

standard exodus output These are always stored in the same directory as the spread exodus file.

history and frequency output These are stored in the same directory as the first spread exodus file. These files are "joined" on the fly inside of Salinas. However, to insure that the files are written to the RAID disks, we have opted to store them in the same directory as the first spread file.

results These files are typically small and should be written by only a subset of the processors. For this reason, and for convenience, they are written to the directory where the input is found. This is usually the working directory.

33 Assembling element to system matrices

At this time we are using lmatrices to assemble the stiffness and mass mass matrices. These are inefficient, and a pain. I'd like to get away from supporting these matrices. To do this, we need a procedure to assemble the system matrices directly into a sparse format. This outline of the process comes from Clark Dohrman.

In a nutshell, the assembly follows these steps.

1. generate a list of elements connected to node k, `A1`.
2. from this list, and from the connectivity, generate a list of nodes adjacent to node k, `B1`.
3. This list provides the node sparsity pattern. A dof sparsity can be easily generated from that.
4. The number of terms in the dof graph is the NNZ of the stiffness matrix.
5. Clark uses an A, I, J array, each of length NNZ to store the data. That could be easily converted to the standard CSR format.

The pseudo code he presented, is listed below. It is a mix of matlab, C and fortran

```
elemnt i nodes
E1(E2(i):E2(i+1)-1) is the connectivity of element i.

count=zeros(1,nnode)
for i=1,nelem
   j(i) = nodes_for_element_i
   count(j(i))= count(j(i)+1
end
```

```
A2=zeros(nnodes)
A2(1)=1
A2(2)=A2(1)+count(1)
A2(3)=A2(2)+count(2)

A1=new A2(nnodes+1)-1

// copy the section with count in it, with slight modifications
// then, A1(A2(K):A2(K_1)-1) = elements connected to node k

// next, find nodes adjacent to node K, B1(B2(k):B2(k+1)-1)
zero(count,nnodes)

for i=1,nnodes
  nanode=0
  naelem=A2(i+1)-A2(i)
  for j=1,naelem
    elem=A1(A2(i)+j-1)
    for k=E2(elem+1)-E2(elem)
        node=E1(E2(elem)+K-1)
        if (count(node)==0) then
          nanode=nanode+1
          count(node)=1
          anode(nanode)=node
        end
   count(anode(1:nanode))=0  // no need to zero the whole thing
   end
  end
end
```

34 Time integration

34.1 Linear transient analysis

The equations of motion of the structure are

$$\begin{aligned} M\left[(1-\alpha_m)a_{n+1}+\alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1}+\alpha_f v_n\right]+ \\ K\left[(1-\alpha_f)d_{n+1}+\alpha_f d_n\right] &= F_{n+1+\alpha_f} \end{aligned}$$

$$\tag{113}$$

where α_f, α_m are the integration parameters for the generalized α method, and $\hat{C} = C + \alpha M + \beta K$. That is, the damping matrix is the sum of the standard damping matrix C plus the proportional damping terms. Also,

$$F_{n+1+\alpha_f} = F((1-\alpha_f)t_{n+1} + \alpha_f t_n) \tag{114}$$

The time integration scheme is defined as follows

$$\begin{aligned} d_{n+1} &= d_n + \Delta t v_n + \frac{\Delta t^2}{2}\left[(1-2\beta_n)a_n + 2\beta_n a_{n+1}\right] \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \end{aligned} \tag{115}$$

where γ_n, β_n are the integration parameters for the Newmark method. In order to have a displacement-based method, we solve these equations for the acceleration and velocity in terms of displacement, which yields

$$\begin{aligned} a_{n+1} &= \frac{1}{\beta_n \Delta t^2}\left[d_{n+1} - d_n - v_n\Delta t\right] - \frac{1-2\beta_n}{2\beta_n}a_n \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \\ &= v_n + \Delta t\left[(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n\Delta t^2}\left[d_{n+1} - d_n - v_n\Delta t\right] - \gamma_n\frac{1-2\beta_n}{2\beta_n}a_n\right] \end{aligned} \tag{116}$$

Substituting these equations into the equation of motion, and collecting terms, we obtain

$$\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2} + \hat{C}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t} + K(1-\alpha_f)\right]d_{n+1} =$$

$$F_{n+1+\alpha_f} - K\alpha_f d_n$$

$$-\hat{C}\left[\alpha_f v_n + (1-\alpha_f)\left[v_n + \Delta t(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n\Delta t}\left[-d_n - \Delta t v_n\right] - \frac{\gamma_n\Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right]$$

$$+M\left[-\alpha_m a_n + \frac{1-\alpha_m}{\beta_n\Delta t^2}\left[d_n + v_n\Delta t\right] + (1-\alpha_m)\frac{1-2\beta_n}{2\beta_n}a_n\right]$$

There are three matrix-vector products on the right hand side of this equation, one for each of the system matrices M, K, and C.

34.2 Nonlinear transient analysis

This section follows closely the nonlinear transient procedure given by Belytschko et al,[6] with the modification of using the generalized alpha integrator rather than the Newmark beta approach. In the case of a nonlinear transient analysis, the equation of motion is

$$\begin{aligned} M\left[(1-\alpha_m)a_{n+1}+\alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1}+\alpha_f v_n\right]+ \\ (1-\alpha_f)F^{int}_{n+1}+\alpha_f F^{int}_n &= F_{n+1+\alpha_f} \end{aligned} \tag{117}$$

where F^{int}_{n+1} and F^{int}_n are the internal forces at the current and previous time steps, respectively.

Using the tangent stiffness method, we replace F^{int}_{n+1} as

$$F^{int}_{n+1}=F^{int}_n+K_t\Delta d \tag{118}$$

where K_t is the tangent stiffness matrix. Also, we use equations 116, which are the same as in the linear case.

First, we substitute equations 116 and 118 into equation 117. This results in the following equations, which are almost identical to the ones from the linear case

$$\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2}+\hat{C}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t}+K_t(1-\alpha_f)\right]d_{n+1}=$$

$$F_{n+1+\alpha_f}-\alpha_f F^{int}_n-(1-\alpha_f)\left[F^{int}_n-K_t d_n\right]$$

$$-\hat{C}\left[\alpha_f v_n+(1-\alpha_f)\left[v_n+\Delta t(1-\gamma_n)a_n+\frac{\gamma_n}{\beta_n\Delta t}\left[-d_n-\Delta t v_n\right]-\frac{\gamma_n\Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right]$$

$$+M\left[-\alpha_m a_n+\frac{1-\alpha_m}{\beta_n\Delta t^2}\left[d_n+v_n\Delta t\right]+(1-\alpha_m)\frac{1-2\beta_n}{2\beta_n}a_n\right]$$

Finally, we want the unknown to be $\Delta d=d_{n+1}-\hat{d}$, where $\hat{d}$ is the current iterate of displacement. To accomplish this, we subtract the appropriate terms from both sides, which yields, after collecting terms

$$\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2}+\hat{C}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t}+K_t(1-\alpha_f)\right]\Delta d=$$

$$F_{n+1+\alpha_f}-(1-\alpha_f)\hat{F}^{int}-\alpha_f F^{int}_n-C\left[(1-\alpha_f\hat{v}+\alpha_f v_n\right] \tag{119}$$

$$-M\left[(1-\alpha_m)\hat{a}+\alpha_m a_n\right] \tag{120}$$

$$\tag{121}$$

where again hats denote current iterates of acceleration, velocity, etc. Upon using the Newmark beta time integrator ($\gamma_n = \frac{1}{2}$, $\beta_n = \frac{1}{4}$, $\alpha_f = \alpha_m = 0$, equation 120 reduces to

$$\left[M\frac{4}{\Delta t^2} + \hat{C}\frac{2}{\Delta t} + K_t\right]\Delta d = \\ F_{n+1} - \hat{F}^{int} - C\hat{v} - M\hat{a} \tag{122}$$

$$\tag{123}$$

which is the same equation given by Belytschko et al.[6]

We note that equation 120 can be written as

$$A\Delta d = res \tag{124}$$

where A is the dynamic matrix, Δd is the change in displacement from the previous Newton teration to the current Newton iteration, and res is the residual, i.e. the amount by which the equations of motion (equation 117) are not satisfied by the current iterate.

34.3 Modal Damping with the Generalized Alpha Method

Modal damping in transient analysis was originally implemented for the Newmark beta method, with $\gamma_n = \frac{1}{2}$ and $\beta_n = \frac{1}{4}$, and the implementation was based on the acceleration-based transient analysis. For more details on this, we refer to the original paper.[7]

This section describes the modifications necessary to change the modal damping implementation to a displacement based method, and for the generalized alpha method. The main approach is the same.

For the generalized alpha method with arbitrary Newmark parameters, the equations of motion of the structure are

$$\begin{aligned} M\left[(1-\alpha_m)a_{n+1} + \alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1} + \alpha_f v_n\right] + \\ K\left[(1-\alpha_f)d_{n+1} + \alpha_f d_n\right] &= F_{n+1+\alpha_f} \end{aligned} \tag{125}$$

where α_f, α_m are the integration parameters for the generalized α method, and $\hat{C} = C + C_\xi + \alpha M + \beta K = C_\xi + C_{\alpha\beta}$. That is, the damping matrix is the sum of the standard damping matrix C, plus the contribution from modal damping C_ξ, plus the proportional damping terms $C_{\alpha\beta}$. Also,

$$F_{n+1+\alpha_f} = F((1-\alpha_f)t_{n+1} + \alpha_f t_n) \tag{126}$$

We note that from this point on, we will assume that $C = 0$, that is, that the damping matrix only includes contributions from modal and proportional damping.

The time integration scheme is defined as follows

$$\begin{aligned} d_{n+1} &= d_n + \Delta t v_n + \frac{\Delta t^2}{2}\left[(1-2\beta_n)a_n + 2\beta_n a_{n+1}\right] \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \end{aligned} \tag{127}$$

where γ_n, β_n are the integration parameters for the Newmark method. In order to have a displacement-based method, we solve these equations for the acceleration and velocity in terms of displacement, which yields

$$\begin{aligned} a_{n+1} &= \frac{1}{\beta_n \Delta t^2}\left[d_{n+1} - d_n - v_n\Delta t\right] - \frac{1-2\beta_n}{2\beta_n}a_n \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \\ &= v_n + \Delta t\left[(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n\Delta t^2}\left[d_{n+1} - d_n - v_n\Delta t\right] - \gamma_n\frac{1-2\beta_n}{2\beta_n}a_n\right] \end{aligned} \tag{128}$$

Substituting these equations into the equation of motion, and collecting terms, we obtain

$$\begin{aligned} &\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2} + \hat{C}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t} + K(1-\alpha_f)\right]d_{n+1} = \\ &F_{n+1+\alpha_f} - K\alpha_f d_n \\ &-\hat{C}\left[\alpha_f v_n + (1-\alpha_f)\left[v_n + \Delta t(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n\Delta t}\left[-d_n - \Delta t v_n\right] - \frac{\gamma_n\Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right] \\ &+ M\left[-\alpha_m a_n + \frac{1-\alpha_m}{\beta_n\Delta t^2}\left[d_n + v_n\Delta t\right] + (1-\alpha_m)\frac{1-2\beta_n}{2\beta_n}a_n\right] = g \end{aligned} \tag{129}$$

We note that the vector g is defined as the right hand side of the above equation.

We define $\hat{g}$ as follows

$$\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2} + C_{\alpha\beta}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t} + K(1-\alpha_f)\right]d_{n+1} =$$

$$F_{n+1+\alpha_f} - K\alpha_f d_n$$

$$-C_{\alpha\beta}\left[\alpha_f v_n + (1-\alpha_f)\left[v_n + \Delta t(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n \Delta t}\left[-d_n - \Delta t v_n\right] - \frac{\gamma_n \Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right]$$

$$+M\left[-\alpha_m a_n + \frac{1-\alpha_m}{\beta_n \Delta t^2}\left[d_n + v_n \Delta t\right] + (1-\alpha_m)\frac{1-2\beta_n}{2\beta_n}a_n\right] = \hat{g}$$

That is, we define $\hat{g}$ to be the right hand side corresponding to the case when the damping matrix only consists of proportional damping terms.

Next, we consider the case $\hat{C} = C_\xi + C_{\alpha\beta}$ in equation 129. In analogy with the approach in,[7] we carry the term $C_\xi v$ to the right hand side.

$$\left[M\frac{(1-\alpha_m)}{\beta_n \Delta t^2} + C_{\alpha\beta}(1-\alpha_f)\frac{\gamma_n}{\beta_n \Delta t} + K(1-\alpha_f)\right] d_{n+1} =$$

$$\begin{aligned} \hat{g} - C_\xi v &= \\ g - C_\xi \frac{\gamma_n}{\beta_n \Delta t} d_{n+1} \end{aligned} \tag{130}$$

We note that

$$\begin{aligned} g &= \hat{g} - C_\xi\left[\alpha_f v_n + (1-\alpha_f)\left[v_n + \Delta t(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n \Delta t}\left[-d_n - \Delta t v_n\right] - \frac{\gamma_n \Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right] \\ &= \hat{g} - C_\xi \tilde{v} \end{aligned} \tag{131}$$

where $\tilde{v}$ is defined as

$$\tilde{v} = \left[\alpha_f v_n + (1-\alpha_f)\left[v_n + \Delta t(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n \Delta t}\left[-d_n - \Delta t v_n\right] - \frac{\gamma_n \Delta t(1-2\beta_n)}{2\beta_n}a_n\right]\right]$$

Following the same approach as in,[7] we represent the matrix C_ξ as

$$C_\xi = M\Phi\Sigma\Phi^T M$$

We also represent the unknown current displacement d_{n+1} as

$$d_{n+1} = \Phi x + z \tag{132}$$

where z is the residual contribution to d_{n+1}, which is mass-orthogonal to Φ. We next note that

$$C_\xi \frac{\gamma_n}{\beta_n \Delta t} d_{n+1} = \frac{\gamma_n}{\beta_n \Delta t} M \Phi \Sigma x \tag{133}$$

The right hand side of equation 133 is the contribution to the force equation due to modal damping. To compute $\frac{\gamma_n}{\beta_n \Delta t} M \Phi \Sigma x$, we simply need to determine x.

x can be found by inserting equation 132 into equation 129, premultiplying by Φ^T, and using the fact that z is mass-orthogonal to Φ. We also have to use equation 131.

$$\left[I \frac{(1-\alpha_m)}{\beta_n \Delta t^2} + \left[\Sigma + \alpha I + \beta \Omega^2 \right] (1-\alpha_f) \frac{\gamma_n}{\beta_n \Delta t} + \Omega^2 (1-\alpha_f) \right] x =$$

$$\begin{aligned} &\Phi^T \hat{g} - \Phi^T C_\xi \tilde{v} = \\ &\ \Phi^T \hat{g} - \Sigma \Phi^T M \tilde{v} \end{aligned}$$

Since the right hand side of the above equation is known, and the left hand side is a diagonal matrix, the solution x can be found easily.

Once x is known, the right hand side of equation 133 can be computed. Then, the right hand side of equation 130 can be computed, which completes the derivation of the right hand side contribution for modal damping.

35 Matrix dimensions in Salinas

There are number of different dimensions in Salinas. These will be summarized here with a focus on using the data within the matlab framework. Examples of how to convert data from one dimensionality to another will be given.

The subject of matrix dimensions is an important one. Salinas has a fairly simple set of dimensions compared to more complex systems like Nastran. However, it is critical that these be well understood if we wish to manipulate the data.

As an example, I consider an eigen analysis of a structure with 9938 nodes. This structure is made of shells and solids. There are no boundary conditions, but there are 9 mpcs applied. I look at only the serial file sizes.

To get the required maps and other m-files, we must select 'mfiles' in the output section. To get the eigenvector data, we must also write the exodus file with 'disp' selected in the output section.
For this model, we have the following important dimensions.

1. #nodes=9938
2. external set= #nodes * 6 dofs/node = 59628
3. `G-set` = # active dofs before boundary conditions = 42708
4. `A-set` = analysis set = # equations to be solved = 42699
5. reduced external set = #nodes * 3 = 29814

There are 3 dofs/node for solid elements, but shells and beams have 6. In aggregate, the total dofs is 42708 before boundary conditions and mpcs are applied. There are no BCs in the model, but there are 9 MPC equations, each of which eliminates 1 dof, so the Aset is reduced to 42699.

Unfortunately, the `eigen_disp*.m` files are written in the reduced external set since this is what the analysts typically want. The bad news is that these m-files are useless to us. The good news is that all the data is available in either `m-files` or in the `exodus` output.

The matrices `Mssr` and `Kssr` contain the mass and stiffness matrices in the `A-set`. They are symmetric matrices and only one half of the off diagonal is stored. To get the complete matrix within `matlab`,

```
>>> K = Kssr + Kssr' - speye(size(Kssr)).*Kssr;
```

The full eigenvectors (in the external set) are available in the output exodus file. To get them use the `seacas` command `exo2mat`.

```
> exo2mat example-out.exo
```

Within `matlab`, the data can be converted to a properly shaped matrix.

```
>>> load example-out
>>> phi = zeros(nnodes*6,nsteps);
>>> tmp = (0:nnodes-1)*6;
>>> phi(tmp+1,:)=nvar01;
>>> phi(tmp+2,:)=nvar02;
>>> phi(tmp+3,:)=nvar03;
>>> phi(tmp+4,:)=nvar04;
>>> phi(tmp+5,:)=nvar05;
>>> phi(tmp+6,:)=nvar06;
```

We now have phi as a matrix with each column corresponding to an eigenvector. However, phi is dimensioned at 59628 x 10 for this example. We clearly can't multiply phi by K for example - the dimensions don't match. To do this we need a map.

We have two maps in our directory. `FetiMap_a.m` is the map from the external set to the A set. Thus we can reduce `phi` to the `A-set` by combining it with `Fetimap_a`. If the `G-set` is desired instead of the `A-set`, replace `FetiMap_a` with `FetiMap`.

```
>>> p2=zeros(max(max(FetiMap_a)),nsteps);
>>> for j=1:nnodes*6
>>>  i=FetiMap_a(j);
>>>  if ( i > 0 )
>>>    p2(i,:)=phi(j,:);
>>>  end
>>> end
```

This is slow. A faster, but less straightforward method is shown here.

```
>>> mapp1=FetiMap_a+1;
>>> tmp=zeros(max(max(mapp1)),nsteps);
>>> tmp(mapp1,:)=phi;
>>> p2=tmp(2:max(max(mapp1)),:);
```

Now we can do all the neat things like `p2'*K*p2`.

To get back to the external set, we again use this map. For example, if we have a vector of dimension 42699,

```
>>> x=1:42699';
>>> XX = zeros(59628,1);
>>> for i=1:59628
>>>   if ( FetiMap_a(i)>0 )
>>>     XX(i)=x(FetiMap_a(i));
>>>   end
>>> end
```

Obviously, similar shortcuts can be made to make this more efficient. One that appears to work is shown here.

```
>>> xtmp=[ 0 x'];
>>> X2=xtmp(mapp1);
```

36 Isotropic Material Parameters and Sensitivities

Issue: There are 4 commonly used parameters for input to isotropic materials (E, ν, K and G). Only two of these are appropriate at a time, i.e. exactly 2 are required for complete material specifications.

Within Salinas, only K and G are stored. Thus, we must provide relations to derive these from the other two parameters. From these relations, we must also develop the relations for sensitivity.

36.1 Material Relations

Relations for K and G for given input are provided below.

Case 1: E and ν are provided.

$$K = \frac{E}{3(1-2\nu)} \tag{134}$$

$$G = \frac{E}{2(1+\nu)} \tag{135}$$

Case 2: G and ν are provided.

$$K = \frac{2G(1+\nu)}{3(1-2\nu)} \tag{136}$$

Case 3: G and E provided.

$$K = \frac{GE}{3(3G-E)} \tag{137}$$

Case 4: K and ν provided.

$$G = \frac{3K(1-2\nu)}{2(1+\nu)} \tag{138}$$

Case 5: K and E provided.

$$G = \frac{3KE}{9K-E} \tag{139}$$

36.2 Sensitivities

Sensitivies are computed using the above relations. For example, if E and ν are provided (as in case 1), we compute deriviatives using equation 134

$$\left.\frac{\partial K}{\partial E}\right|_{\nu=0} = \frac{d}{dE}\left(\frac{E}{3(1-2\nu)}\right)$$

Each case must be considered separately because different variables are held fixed during differentiation. We store dK/dp and dG/dp (where dp is the change in a parameter). We then use the chain rule to compute the total derivative.

$$\frac{\partial \lambda}{\partial p} = \frac{\partial \lambda}{\partial K}\frac{\partial K}{\partial p} + \frac{\partial \lambda}{\partial G}\frac{\partial G}{\partial p} \tag{140}$$

Case 1: E and ν provided.

$$\text{using } K = \frac{E}{3(1-2\nu)}$$

$$\frac{dK}{dE} = \frac{K}{E} \tag{141}$$

$$\frac{dK}{d\nu} = \frac{2K}{1-2\nu} \tag{142}$$

$$\text{using } G = \frac{E}{2(1+\nu)}$$

$$\frac{dG}{dE} = \frac{G}{E} \tag{143}$$

$$\frac{dG}{d\nu} = \frac{-G}{1+\nu} \tag{144}$$

Case 2: G and ν are provided.

$$\text{using } K = \frac{2G(1+\nu)}{3(1-2\nu)}$$

$$\frac{dK}{dG} = \frac{K}{G} \tag{145}$$

$$\frac{dK}{d\nu} = \frac{2G}{(1-2\nu)^2} \tag{146}$$

Case 3: G and E provided.

$$\text{using } K = \frac{GE}{3(3G-E)}$$

$$\frac{dK}{dG} = \frac{-E^2}{3(3G-E)^2} \tag{147}$$

$$\frac{dK}{dE} = \frac{G^2}{(3G-E)^2} \tag{148}$$

Case 4: K and ν provided.

$$\text{using } G = \frac{3K(1-2\nu)}{2(1+\nu)}$$

$$\frac{dG}{dK} = \frac{G}{K} \tag{149}$$

$$\frac{dG}{d\nu} = \frac{-9K}{2(1+\nu)^2} \tag{150}$$

Case 5: K and E provided.

$$\text{using } G = \frac{3KE}{9K-E}$$

$$\frac{dG}{dK} = \frac{-3E^2}{(9K-E)^2} \tag{151}$$

$$\frac{dG}{dE} = \frac{27K^2}{(9K-E)^2} \tag{152}$$

37 Stochastic FE Integration

At this time, stochastic FE have been introduced in Salinas only in a very unobtrusive way, i.e. we have generated an element that reads parameters from a file and computes the result. Salinas is treated only as a black-box function evaluator. This has limitations, the most important of which is that the statistics are sampled. The benefit is that all the real stochastic work is done outside of the Salinas framework.

Roger Ghanem has suggested a more intrusive approach which would allow greater power in application of the stochastic method. The drawback is that Salinas must absorb more of the modifications. I here try to summarize the ideas behind these modifications.

The variation is written as an expansion in terms of orthogonal basis functions.

$$u(x,\xi) = \sum_i u_i(x)\psi_i(\xi) \tag{153}$$

Where u represents the response,
u_i represents the deterministic coordinates,
ψ_i represents the random variable basis functions, and
ξ is the basic random variable on which everything depends. For example, ξ might represent the variation in Youngs modulus in a given finite element.

The random variable basis functions are orthogonal (but not orthonormal), ie.

$$< \psi_i\psi_j >= \delta_{ij} < \psi_i^2 > \tag{154}$$

We can combine these relations to solve for u_j,

$$u_j = \frac{< u\psi_j >}{< \psi_j^2 >} \tag{155}$$

Now take the equation for statics (it appears that each problem must be examined individually in this method).

$$Ku = f \tag{156}$$

Using the KL optimal basis expansion (ϕ_i), K may be expanded in terms of the basic random variables (other expansions also exist including an expansion in terms of ψ).

$$\sum_i \xi_i K_i(\phi_i)u \quad = \quad f \tag{157}$$

$$\sum_{i,j} < \xi_i \psi_j \psi_k > K_i u_j \;=\; < f \psi_k > \tag{158}$$

or,

$$\sum \sum_j \left(\sum_i C_{ijk} K_i \right) u_j \;=\; f_k \tag{159}$$

or,

$$\sum_j \tilde{K}_{jk} u_j \;=\; f_k \tag{160}$$

Where $C_{ijk} =< \xi_i \psi_j \psi_k >$, and $\tilde{K}_{jk} = \sum_i C_{ijk} K_i$. One of the important aspects of this approach is that the C_{ijk} terms can be computed exactly.

This is a *huge* block diagonal system, where each block in the system is the $\tilde{K}_{jk}$ matrix associated with the expansion of the random variable ψ_k. These submatrices have some interesting properties. The $\tilde{K}_{00}$ term is just the deterministic solution, and is thus positive definite. The other terms may not be positive definite, but the sum of the terms always is, provided only that the variations in ξ are physical (e.g. we don't want negative Young's modulus).

There may be interesting ways to solve these coupled systems, which are based on the fact that each of the $\tilde{K}_{jk}$ are similar to the deterministic one. One such method would write $\tilde{K}_{ij} = \tilde{K}_{00} \delta_{ij} + \Delta_{ij}$. The Δ terms are moved to the right hand side of the equation and we solve iteratively,

$$\tilde{K}_{00} u_i^{(r+1)} = f_i + g^{(r)} \tag{161}$$

where g contains a product of the Δ terms with $u^{(r)}$. The nice part of this kind of solution is that we are always solving the same left hand side. The down-side is that the solution is iterative. I don't know if we can prove under what conditions it is guaranteed to converge.

Symbolically,

$$\begin{bmatrix} K_{00} & & \\ & \cdot & \\ & & \cdot \\ & & & K_{00} \end{bmatrix} \begin{bmatrix} u_0^{(r+1)} \\ \vdots \\ u_n^{(r+1)} \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix} + \begin{bmatrix} \Delta_{00} & \Delta_{01} & \dots & \Delta_{0n} \\ \Delta_{10} & \Delta_{11} & \dots & \Delta_{1n} \\ . & . & . & . \\ \Delta_{n0} & \Delta_{n1} & \dots & \Delta_{nn} \end{bmatrix} \begin{bmatrix} u_0^{(r)} \\ \vdots \\ u_n^{(r)} \end{bmatrix} \tag{162}$$

37.1 Solution Proceedure for Stochastic FE

The issues involved in this kind of solution are significant. It would be very helpful to break things out into different tasks, especially if some of these tasks could be handled by a library callable by the FE application. Goals of this effort include.

1. As much work as possible should be abstracted and placed in a callable library, so as to reduce the impact on the FE code.
2. The solution must be structured to maintain parallel distributed computing. There may be more than one way to exploit parallelism here. For example, a set of processors could be dispatched to solve each of the blocks in the system. We will focus on using parallelism the same way as it is used in the deterministic system, i.e. our first focus will solve K_0 in a distributed manner.
3. The FE code should be able to provide a linear solver to the statistical library.
4. We anticipate that even on a distributed machine, we will have trouble with memory.

A first cut at the task breakdown follows.

1. The finite element program reads and manages a list of the stochastic information. I envision this to be similar to current sensitivity information. I'll call this `stoch_info` for convenience.
2. The KL expansion must be computed. This is a big deal because the matrices that are generated are full. It is not hopeless because in a typical application, only a small part of the model is involved in a KL expansion. But, it does mean that a separate partition must be established for KL eigenfunction evaluations. The terms to be computed are,
$$\int_\Omega R(x,y)\phi(y)dy = \lambda\phi(x)$$
This will require connectivity and coordinates from the FE application, but should be otherwise independent of the application. I suggest that this could be put into a library very nicely, but it represents a lot of work.
3. Assemble the $K_i(\phi_i)$ from equation 157. This is clearly the FE application's reponsibility once ϕ has been provided.
4. The FE application reserves space for u_i. It computes $u_0 = K_0^{-1}f$, and $u_i = 0$.

5. The library solves the matrix system. A strawman interface is,
   `Solve StochAxb(K0,Ki,U0,Ui,stoch_info,tol,LinearSolver())`
   Obviously the Ki and Ui terms have to be pointers to arrays. However, it may be better to have the FE application provide a function that will compute $K_i u_j$ since that is all that is needed for the solution. This follows since,

$$\begin{aligned}
\Delta_{ik} u_i &= \left(\tilde{K}_{00} - \tilde{K}_{ik}\right) u_i \\
&= \left(\tilde{K}_{00} - \sum_j C_{jik} K_j\right) u_i \\
&= \tilde{K}_{00} u_i - \sum_j C_{jik} (K_j u_i)
\end{aligned}$$

6. Some routines need to be provided for output. The library could provide these. There is a concern about secondary variables (such as stress) that are very much application dependent.

38 Eigen Accuracy

Rich and Ulrich,

I very much appreciated the time spent last week discussing the accuracy of our eigen solutions. I have a number of issues and/or questions that I hope you could address.

- It would be very helpful if I could get a copy of the slides that Rich presented. My notes are rather incomplete. Could you shoot me the powerpoint presentation?
- I understand that we need to be evaluating the M^{-1} norm. There are a number of issues relating to this norm that you may be able to provide insight to.
  1. I thought I understand Rich to say that the ARPACK package *indirectly* computes the proper M^{-1} norm. Is that correct? If so, is there a reasonable way to extract this computed norm from the ARPACK results?
  2. In the textbooks, M is positive definite. In our models, this is often not the case. Let me list a few of the common issues. Do you have any suggestions on how to handle these issues?
     (a) Multipoint constraints change the system of equations that we are looking at. With multipoint constraints, we are looking at a system of equations like the following.

     $$\left[\begin{pmatrix} K & C^T \\ C & 0 \end{pmatrix} - \lambda \begin{pmatrix} m & 0 \\ 0 & 0 \end{pmatrix} \right] \begin{bmatrix} u \\ \mu \end{bmatrix} = 0 \qquad (163)$$

     Obviously, as here presented, M is singular. Clearly we could do some kind of Schurr complement, but for a large system of equations, this would be cost prohibitive (even if m were diagonal).

     (b) The inertia terms for rotational degrees of freedom tend to zero faster than the translational terms. These leads to ill conditioned mass matrices (though they are seldom truly singular).

     (c) Mass lumping. Sometimes analysts tend to reduce the mass of the model, and store it on concentrated masses on the nodes. Those masses often have no rotational inertias. While

I wouldn't think that is always the best modeling practice, the analysts have excellent reasons for doing this, and I want to support this if possible. This approach was used in the Newport News model of an aircraft carrier.

   (d) Specialized elements such as joint elements (which act like springs) have no mass associated with them. Generally, there is mass associated with the other elements, but that is not always the case. For example, in a typical joint, a collection of massless beams (or more typically MPCs) will reduce the degrees of freedom on a face of solid elements to a single node. The matching face (on the other side of the joint) will likewise be reduced to a single node. These two nodes are then joined with a rather complex Iwan element (which is, and should be, massless).

3. Given that direct computation of the M^{-1} norm can be rather challenging, is there any benefit in modifying our existing (inadequate) $L2$ norm? For example, would a computation of $||r||_2/||M||_\infty$ be better than $||r||_2$ alone? Are there other, simple things we could do when the computation of the correct norm may be too expensive?

39 Eigenvalue Error Estimators

A number of error estimators have been defined for the computation of uncertainty in eigenvalues. We will be evaluating explicit estimators. Most of these have a form like the following.

$$\begin{aligned} |\lambda_{true} - \lambda_{computed}| &= |error| && (164) \\ &= \sum_e \rho_e^{internal} + \rho_e^{boundary} && (165) \end{aligned}$$

where, $\rho_e^{internal}$ is the contribution of element, e, due to errors in the computation of the eigenvalue equation, and $\rho_e^{boundary}$ is the contribution due to jumps in quantities through the boundaries of the elements.

Typically,

$$\rho_e^{internal} = \int_V x^T(Kx - \lambda Mx)dV$$

and,

$$\rho_e^{boundary} = \int_{Surface} \hat{n} \cdot (Kx - \lambda Mx)dS$$

Here $\hat{n}$ is the surface normal through the element. Two main quantities are of interest. The global error, equation 165, indicates the uncertainty in the computation of the global eigenvalue. The local error involves the element contributions by themselves, but also includes the flux into the element by neighbors. Because the normals are reversed, a uniform flux would result in cancelation of the boundary term.
We consider only the contributions from solid elements.

39.1 Issues in computing $\rho^{internal}$

There are few issues in computing the contribution from internal components. Essentially all the components are in place. We must resolve only these issues. There are no parallelization issues.

1. What must we integrate. Usually this would be $x^T(Kx - \lambda Mx)$, but other options are possible.

2. What level of integration is required? We could integrate with the current shape functions, or a higher level of integration may be required.

39.2 Issues in computing $\rho^{boundary}$

There are more issues for the boundary terms, particularly since we need to know the neighbors to compute the local indicators.

1. What must we integrate, e.g. $\hat{n} \cdot (Kx - \lambda Mx)$?
2. What surface integration gauss quadrature is appropriate?
3. How do we evaluate the integrand at an arbitrary point?
4. How do we determine the neighbors and their faces? This has important parallelization issues.

39.3 Development Schedule

The following table indicates the tasks that we expect to accomplish, their priority, and who leads that effort.

Description	days	priority	who	parallel	next 6 weeks
Documentation	3	1	T/G	ok	yes
Input Specs	1	2	G	ok	yes
Computation of ρ^{int}	2	3	G	ok	yes
Complete the Gauss rule	4	4	T	ok	yes
Compute function on face	2	5	T	ok	yes
neighbor contributions	6	6	G/T	no	-
global sums	1	7	T	some	-
output routines	2	8	G	some	-

40 Acoustic Coupling

The coupling between an acoustic medium and a structure requires a definition of the *wet surface*, which is the common interface between the two media. A surface integral is performed on this interface in order to compute the acoustic/structural coupling matrix L, which is defined as

$$L = \sum_{i=1}^{NumE} L_e = \sum_{i=1}^{NumE} \int_{\Gamma_{wet}} N_s^T \mathbf{n} N_a ds \tag{166}$$

where Γ_{wet} is the wet surface, $NumE$ is the number of elements on the wet surface, $\mathbf{n}$ is the unit normal on the surface (dependent on position), N_s is the shape function matrix for the structural element, and N_a is the shape function matrix for the acoustical element. Note that both of the shape function matrices only involve the degrees of freedom of the element that lie on the wet surface. For example, for a hex8/hex8 interface, the L_e would be a $12x4$ matrix.

The information defining the wet surface must be stored in an elementwise manner. In parallel this is not a simple determination, since a potential acoustic neighbor may lie on a different subdomain. For those portions of the wet surface that are interior to a subdomain, the determination of the wet surface pairs is straightforward. For any portion of the wet surface lying on a subdomain boundary, the corresponding structural elements must know whether their neighbors on the matching subdomain are acoustical elements. If so, then those structural elements are on the wet surface and thus must be involved in the surface integral just described.

In order to overcome the difficulty of the wet surface definition in the parallel setting, the following procedure is used to augment the neighbor information on a subdomain, so that the subdomain has enough information to determine if any of its structural elements are on the wet surface. On each subdomain, the exodus utilities are used to determine which elements lie on the subdomain interface, and their subdomain interface connectivities. Then, for those elements that are acoustic, their subdomain interface connectivities are sent to the corresponding subdomain neighbors via point to point communications. The neighbors, upon receiving this information, will check the incoming connectivities against those of the local elements in their own subdomain. The matches that are found on structural elements are marked as being on the wet interface. Then, the node to dof maps for the nodes on these elements must be augmented with the acoustical degree of freedom. This latter step is not needed for the elements on the internal parts of the wet surface, whose nodes already know about the acoustical dof. These updates, augmented with the information already generated on each subdomain for the internal parts of the wet surface, completes the elementwise description of the wet surface in a parallel setting.

Two important considerations should be emphasized.

1. The coupling needs to be computed on only one side of the interface. We have chosen to do the computation on the structural side because acoustic elements are simple at this stage. Since all acoustic elements are solids (and we expect that for the foreseeable future) the shape functions are easily estimated without requiring a pointer to the element. The same can not be said for the structural elements which may be solids or shells. Thus, the structural element uses its own shape

functions and a well known acoustic solid shape function to compute the integral.

2. In parallel the coupling matrix appears on only one side of the interface. Again we choose to put it on the structures side. The coupling matrix L, couples the structural dofs on that subdomain to the acoustic dofs *on the same subdomain*. Lagrange multipliers in FETI then couple acoustic dofs across the boundary.

41 Shift-Invert Mode in ARPACK for the Right-Most Modes

David M. Day
October 2, 2003

41.1 Background

A prerequisite for reading these notes is a working knowledge of the contents of *the ARPACK*.[8] ARPACK implements Arnoldi's method with restarts. Arnoldi's method refers to finding V_k with k orthonormal columns, and upper Hessenberg H_k such that $OP(V_k) = V_k H_k + f_k e_k^T$

ARPACK approximates the wanted eigenvalues using a user specified number of restarts. After the user specified number of restarts, ARPACK returns the Ritz values that are wanted and converged.

An unsymmetric definite generalized eigenvalue problem (A, M) may be solved in shift-invert mode: $OP = (A - M\sigma)^{-1}M$ and $B = M$. If A is real, then so is σ. ARPACK supports computing the eigenvalues of (A, M) nearest to the shift. The gravest modes, the eigenvalues of (A, M) with largest real part, are usually wanted. If the gravest modes are not the closest to the shift, then a modified interface has advantages.

41.2 ARPACK-Cayley Code

LOCA contains an extended ARPACK-Cayley interface to ARPACK. The interface is primarily designed to operate in in Cayley mode, and also can operate in shift-invert mode. Is there a reference on the Cayley code?

The extension has two parts. Part one is six files that fit in the ARPACK library. Add to SRC `dnaup2c.f`, `dnaupc.f` and `dneupc.f`. Add to PARPACK/MPI/SRC `pdnaup2c.f`, `pdnaupc.f` and `pdneupc.f`. Part two is a suite of functions that support the interface to ARPACK. Part two will be extensively modified.

- $k = nev$, $m = ncv$
- Build V_k, H_k
- while not converged
  1. Extend to V_m, H_m
  2. spectrum(H_m)
  3. spectrum(H_m) $= U \cup W$
  4. Finished?
  5. Increment k,
     (a) (Exact shift strategy) $\forall \nu$: wanted and converged, as long as $k \leq (ncv - neq)/2$, $++k$.
     (b) For each unwanted Ritz value that is converged, $++k$.
     (c) Ensure that the conjugate of a shift is also a shift.
  6. shifts $u(1 : m - k)$ come from U
  7. Restart $(V_m,\ H_m) \to (V_k,\ H_k)$.

Table 4: IRAM with Exact shift strategy

The ARPACK Cayley interface is used here to solve an eigenvalue problem not in Cayley mode, but in shift-invert mode. The status of shift-invert mode in the ARPACK-Cayley code is documented. Changes and modifications are logged. I wrote comments. I reworked the code a little, and will continue to do so. Needed improvements are documented.

The interface extension has three components.

1. **Fulfil** (`polez3`, `sitest`) The user evaluates the number of converged Ritz values. In shift-invert mode, ARPACK-Cayley function `polez3` counts the number of converged Ritz values. `polez3` uses a function `sitest` to discard negligible eigenvalues.

2. **Shift** The user supplies shifts. $np < ncv$ if a leading block of H_{ncv} splits off. $np = |\{bounds([1 : kplusp]) > 0\}|$. ARPACK-Cayley uses zero shifts.

3. **Select** (`stslc3`, `sitest`) The user selects the desired eigenvalues and eigenvectors. ARPACK-Cayley uses `stslc3` to select the converged

Ritz values. Note that `polez3` passes some data to `stslc3` `polez3` such as a cutoff.

`polez3` passes a cutoff value to `stslc3`. **My polishing with the code might have broken this.**

41.3 Features

The relationship between the properties of the eigenvalues of H_k and the eigenvalues of (A, M) is different in the symmetric case. A dispute is ongoing in computational science over whether or not to call the eigenvalues of H_k Ritz values if A is unsymmetric. As the dimension of the Krylov subspace expands, a Ritz value will converge to an eigenvalue of OP. The information about the gravest modes provided by the Ritz values of OP is deceptive. It is not clear whether or not a Ritz value really comes from its reputed source.

`sitest` uses a culling formula that has proved robust in Cayley mode. The justification depends on the Cayley transformation. The rational basis begins with the fact that the error in a small eigenvalue of OP is amplified in the transformation to (A, M). Small Ritz values are rejected based on a criterion in the subroutine `sitest`. If $m = ncv$, and ν is an eigenvalue of OP, then

$$|\nu|^2 > \frac{.02}{m^2 + 100}$$

In shift-invert mode, `sitest` will not work. The criterion is not homogeneous with respect to a norm of OP. One approach is to include the spectral radius of OP in the formula.

Another point of view is that no inaccurate Ritz value has physical significance. User's may assess Ritz values as follows. Suppose that a Ritz value ν approximates an eigenvalue of OP with absolute error β and relative error ξ. That is, $\nu + z$ is an eigenvalue of OP for some $|z| < \beta$. The information defines a set that contains an eigenvalue of (A, M). Inaccurate approximations correspond to a large domain of uncertainty. It is the *left*-most point in the domain of uncertainty that indicates the graveness of the approximate mode (see Figure 1). A conservative approach is to use an estimate of the accuracy of the Ritz value that tends to over-estimate the accuracy. The Ritz error estimate tends to over-estimate the accuracy (assumes unit separation).

A way to throw into the scale the Ritz error bounds in the assessment of a Ritz value is to sort ν according to the minimum of the real part of

$1/(\nu + z)$. The image of the circle $|z - \nu| = \beta$ under inversion is another circle $|z - \mu| = \gamma$. If $\nu \leq \beta$, then inversion maps the neighborhood of ν to the exterior of a neighborhood of $1/\nu$, and the left-most real part is $-\infty$.

If $\nu > \beta$, then $\xi = \beta/|\nu| < 1$. The diametric points $\nu(1 \pm \xi)$ on the ν circle correspond to diametric points $1/(\nu(1 \pm \xi))$ on the image μ circle. The image circle has center

$$\mu = \frac{1}{2\nu}\left(\frac{1}{1-\xi} + \frac{1}{1+\xi}\right) = \frac{\nu^*}{|\nu|^2 - \beta^2},$$

and radius such that

$$\gamma = \frac{1}{2|\nu|}\left(\frac{1}{1-\xi} - \frac{1}{1+\xi}\right) = \frac{\beta}{|\nu|^2 - \beta^2}$$

Here the mode ν is sorted according to

$$\frac{\Re(\nu) - \beta}{|\nu|^2 - \beta^2} \tag{167}$$

The formula simplifies if ν is (nearly) real.

For shift-invert, add the function `Assess` that determines a permutation of the Ritz values so that equation 167 is non-decreasing and conjugate modes are adjacent.

The Fulfil function determines at what threshold are the *nev* gravest modes converged. For restarts, the shift are the least grave modes. The gravest modes are selected.

A different output would be useful. It would help to return the converged or wanted Ritz values in three blocks: the converged and wanted, the unconverged and wanted, and third the converged and unwanted.

41.3.1 Supplying Shifts

In Cayley mode, in order to update the shifts between restarts, zero shifts are used:

$$zero(workl[ipntr[13]-1], 2 * iparam[7]);$$

$$workl[ipntr[13]-1+iparam[7]-1] = 1.0;$$

In shift-invert mode, exact shifts are legal but not implemented.

Here is the specification for how to supply the shifts. Some notation helps to clarify the description, $i = \sqrt{-1}$ and $usp = ipntr(13)$. Supply up to $np = iparam(7)$ shifts, and set $iparam(7)$ to the number of shifts actually used. The real and imaginary parts of the complex shifts are returns in $workl$ at locations $usp - 1$ and $usp + np - 1$.

Only complex conjugate pairs of shifts may be applied, and the conjugate pairs must be placed in consecutive locations. If $np = ncv$, then location np may not contain a conjugate pair.

The ncv Ritz values and Ritz error estimates are in $workl$ too. $ipntr(5 : 7)$ points to real, imaginary, and bounds respectively. Complex conjugate pairs are consecutive.

41.4 Anasazi

Restarting in Anasazi is implemented as in *Stewart*.[9] The ARPACK-Cayley design for shift-invert corresponds to the following for Anasazi.

1. The user supplies a permutation of the Ritz values into Wanted and Unwanted parts. The code might supply a default permutation. The user needs access to the Ritz values and Ritz error estimates, and supplies a permutation.

2. The existing Anasazi iterate capability allows the user to terminate or resume the iteration, and compute the number of converged approximate eigenvalues.

3. The user selects some subset of Ritz values for which eigenvectors are needed. Here one might duplicate the tools developed for ARPACK-Cayley.

References

[1] 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.

[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] 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.

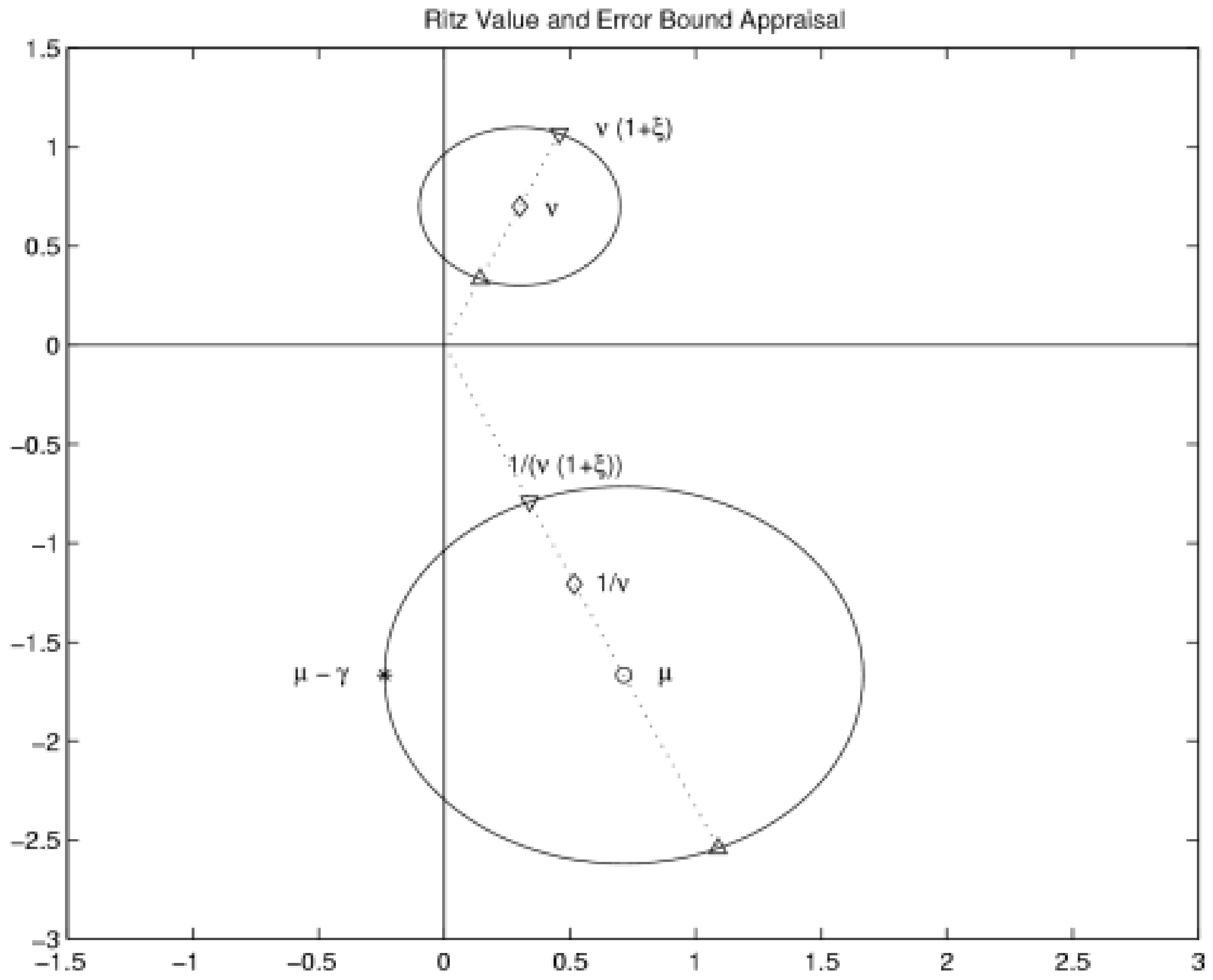

Figure 7: Interpretation of OP Ritz values and error estimates. A neighborhood of a small Ritz value ν of OP corresponds to a large neighborhood of an eigenvalue $1/\nu$ of (A, M).

[4] Zienkiewicz, O. C. and Taylor, R. L., *The Finite Element Method*, vol. 2, chap. 1, McGraw-Hill Book Company Limited, fourth edn., 1991, pp. 23–26.

[5] Ertas, A., Krafcik, J. T., and Ekwaro-Osire, S., “Explicit Formulation of an Anisotropic Allman/DKT 3-Node Thin Triangular Flat Shell Elements,” *Composite Material Technololgy*, **vol. 37**, 1991, pp. 249–255.

[6] T. Belytschko, W. K. L. and Moran, B., *Nonlinear Finite Elements for Continua and Structures*, John Wiley & Sons, first edn., 2000.

[7] Alvin, K. F., “Implementation of Modal Damping in a Direct Implicit Transient Algorithm,” April 2001.

[8] Lehoucq, R. B., Sorensen, D., and Yang, C., *ARPACK Users' Guide*, SIAM, Philadelphia, PA, USA, 1998.

[9] Stewart, G. W., “A Krylov-Schur algorithm for large eigenvalue problems,” **vol. 23**, no. 3, June 2001, pp. 601–614.