SAND2004-4079
Printed August 2004

Salinas–Theory Manual

Garth Reese* Manoj K. Bhardwaj† Timothy Walsh‡

Sandia National Laboratories
Albuquerque, NM 87185-0847

August 16, 2004

*Phone: 845-8640
†Phone: 844-3041
‡Phone: 284-5374

Revision: 1.7

Date: 2004/07/19 14:44:51

Latest Software Release: 1.3

Abstract

This manual describes the theory behind many of the constructs in Salinas. For a more detailed description of how to use **Salinas** , we refer the reader to *Salinas, User's Notes*.

Many of the constructs in **Salinas** are pulled directly from published material. Where possible, these materials are referenced herein. However, certain functions in **Salinas** are specific to our implementation. We try to be far more complete in those areas.

The theory manual was developed from several sources including general notes, a *programer_notes* manual, the user's notes and of course the material in the open literature.

(this page intentionally blank)

Salinas Theory Manual

Contents

1 Solutions

One thing which makes **Salinas** somewhat unique among the many mechanics codes developed at *Sandia National Labs* is that **Salinas** combines a variety of different solution procedures. These range from modal superposition based solutions to nonlinear transient. As described in the *User's Notes* , these solutions can be combined (or chained) in solution cases. This section of the manual describes the theory behind these individual solutions. For details about particular finite elements, see section 2.

1.1 Time integration

1.2 Linear transient analysis

The equations of motion of the structure are

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

where F^{ext} is the external load, 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. [1]

The time integration scheme is defined as follows

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

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

[1] As specified in the user's manual, α_f and α_m are specified by the parameter RHO in the time integrator. Where,

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

and velocity in terms of displacement, which yields

$$\begin{aligned} a_{n+1} &= \frac{1}{\beta_n \Delta t^2}\left[d_{n+1} - d_n - v_n \Delta t\right] - \frac{1-2\beta_n}{2\beta_n} a_n \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \\ &= v_n + \Delta t\left[(1-\gamma_n)a_n + \frac{\gamma_n}{\beta_n \Delta t^2}\left[d_{n+1} - d_n - v_n \Delta t\right] - \gamma_n \frac{1-2\beta_n}{2\beta_n} a_n\right] \end{aligned} \tag{3}$$

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.

1.3 Nonlinear transient analysis

This section follows closely the nonlinear transient procedure given by Belytschko et al,[1] 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

$$\begin{aligned} M\left[(1-\alpha_m)a_{n+1} + \alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1} + \alpha_f v_n\right] + \\ (1-\alpha_f)F^{int}_{n+1} + \alpha_f F^{int}_n &= (1-\alpha_f)F^{ext}(d_{n+1}) + \alpha_f F^{ext}(d_n) \end{aligned} \tag{4}$$

where F^{int}_{n+1} and F^{int}_n are the internal forces at the current and previous time steps, respectively. Note that we have written the external loads as functions of displacement, since in the most general case they could be follower loads.

Before preceeding, we note that there are two possible approaches for implementing the generalized alpha method, and in equation 4 we have taken one of these approaches. The difference lies in the treatment of the internal and external forces. The first approach is to evaluate them as follows

$$\begin{aligned} F^{int}((1-\alpha_f)d_{n+1} + \alpha_f d_n) \\ F^{ext}((1-\alpha_f)d_{n+1} + \alpha_f d_n) \end{aligned} \tag{5}$$

and the second is to evaluate two separate terms

$$\begin{aligned} (1-\alpha_f)F^{int}(d_{n+1}) + \alpha_f F^{int}(d_n) \\ (1-\alpha_f)F^{ext}(d_{n+1}) + \alpha_f F^{ext}(d_n) \end{aligned} \tag{6}$$

When both F^{ext} and F^{int} are linear functions, the two approaches are identical. For nonlinear problems, both F^{ext} and F^{int} could be nonlinear functions, and thus the two procedures are different. In the limit of very small time steps, these nonlinear functions effectively linearize and the two approaches again become the same. Thus the limiting behaviour of the two approaches is the same.

We note that in most cases, the external load F^{ext} is treated as a piecewise linear function of time, and in those cases the two approaches yield the same result for the external load, though a couple of exceptions are worth mentioning. First, if two consecutive time steps lie within two different linear segments, then the two approaches above yield different loads. Second, although they are seldom used, polynomial and loglog interpolation functions are available in Salinas in addition to the commonly used linear interpolation, and in those cases different load vectors result from the above procedures. For problems with very large time steps and involving polynomial interpolation, different results are to be expected.

In Salinas we have chosen the second option, which evaluates both the internal force and external force at both times of interest, and forms a linear combonation of the two. Comparisons have shown little difference in the results on simple test problems.

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

$$F^{int}_{n+1} = F^{int}_n + K_t \Delta d \tag{7}$$

where K_t is the tangent stiffness matrix, defined as $K_t = \partial F_{internal}/\partial u$. Also, we use equations 3, which are the same as in the linear case.

First, we substitute equations 3 and 7 into equation 4. This results in the following equations, which are almost identical to the ones from the linear case

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

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

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

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

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

$$\begin{aligned}\left[M\frac{(1-\alpha_m)}{\beta_n\Delta t^2}+\hat{C}(1-\alpha_f)\frac{\gamma_n}{\beta_n\Delta t}+K_t(1-\alpha_f)\right]\Delta d = \\ F_{n+1+\alpha_f}-(1-\alpha_f)\hat{F}^{int}-\alpha_f F_n^{int}-C\left[(1-\alpha_f\hat{v}+\alpha_f v_n\right] \\ -M\left[(1-\alpha_m)\hat{a}+\alpha_m a_n\right]\end{aligned} \quad (8)$$

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 8 reduces to

$$\left[M\frac{4}{\Delta t^2}+\hat{C}\frac{2}{\Delta t}+K_t\right]\Delta d = F_{n+1}-\hat{F}^{int}-C\hat{v}-M\hat{a} \quad (9)$$

which is the same equation given by Belytschko et al.[1]

We note that equation 8 can be written as

$$A\Delta d = res \quad (10)$$

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 4) are not satisfied by the current iterate.

1.4 Time integration with viscoelastic materials

Here we describe the integration of viscoelastic structures using the generalized alpha method. For the proper choice of the parameters of the generalized alpha method, the results below reduce to those corresponding to the Newmark-beta method.

1.4.1 Equations of motion

The equations of motion of elastodynamics in three dimensions are given by

$$u_{tt} - \nabla \cdot \sigma = f(x,t) \quad \Omega \tag{11}$$
$$u(x,t) = 0 \quad x \in \Gamma_D \tag{12}$$
$$\sigma(x,t) = g(x,t) \quad x \in \Gamma_N \tag{13}$$
$$\tag{14}$$

where $u = (u_x, u_y, u_z)$ is the vector of displacements, σ is the stress tensor, and $f(x,t)$ is the body force. The boundary of Ω is divided into Dirchlet Γ_D and Neumann Γ_N subregions.

The Dirichlet conditions lead to the space of admissible functions

$$V = \left[v \in H^1(\Omega), v(x) = 0, x \in \Gamma_D \right] \tag{15}$$

The equation of motion, along with boundary conditions, is cast into the weak form in the standard way

$$\int_\Omega u_{tt} \cdot v + \int_\Omega \sigma \cdot \nabla_s v dx = \int_\Omega f(x,t) \cdot v dx + \int_{\Gamma_N} g(x,t) \cdot v ds \quad \forall v \in V \tag{16}$$

where an integration by parts has been carried out on the middle term, and $\nabla_s = \frac{1}{2}(\nabla + \nabla^T)$ denotes the symmetric part of the gradient operator.

1.4.2 Constitutive equations

The representation of the time-dependent moduli for a viscoelastic material is commonly written in the form of a Prony series

$$G(t) = G_{\text{inf}} + (G_0 - G_{\text{inf}})\zeta_G(t) \tag{17}$$
$$\zeta_G(t) = \sum_i c_i e^{-\frac{t}{s_i}} \tag{18}$$

where G_0 is the glassy modulus, G_{inf} is the rubbery modulus, and c_i, s_i are coefficients used to fit the Prony series representation to the experimentally measured

relaxation curve. A similar expression holds for $K(t)$, with different values for the constants, and possibly a different number of terms in the series. Assuming an isotropic viscoelastic constitutive law, we only need to consider two rate-dependent material properties. In this presentation, we will work in terms of the bulk K and shear G moduli, since experimental data is typically given in terms of these two parameters.

The constitutive model for an elastic material can be written in terms of the shear and bulk moduli

$$\sigma = D\epsilon = (KD_K + GD_G)\epsilon \tag{19}$$

where D_K, D_G are given in equation 9.4.7 in,[2] and K, G are the bulk and shear moduli. This constitutive law can be generalized to a linear viscoelastic material as follows

$$\begin{aligned}\sigma(x,t) = (G_0 - G_{\inf})D_G \int_0^t \zeta_G(x, t-\tau)\frac{\partial \epsilon(x,\tau)}{\partial \tau}d\tau + G_{\inf}D_G\epsilon(x,t) + \\ (K_0 - K_{\inf})D_K \int_0^t \zeta_K(x, t-\tau)\frac{\partial \epsilon(x,\tau)}{\partial \tau}d\tau + K_{\inf}D_K\epsilon(x,t)\end{aligned} \tag{20}$$

The above expression is then used to represent the stress in the weak form of the equations of motion, 16.

Given a finite dimensional subspace $V_h \subset V$, we represent the approximate solution in the standard way

$$u_h(x,t) = \sum_{i=1}^{n} \phi_i(x)\eta_i(t) \tag{21}$$

where $V_h = span(\phi_i)$, and $\eta(t)$ represents the unknown time dependence. We also denote $\Phi(x) = [\phi_i(x)]$ as the matrix having ϕ_i as the i^{th} column. Inserting this into the equations of motion, and rearranging, we obtain

$$\begin{aligned}M\ddot{\eta}(t) + (G_0 - G_{\inf})K_1 \int_0^t \zeta_G(t-\tau)\dot{\eta}(t)d\tau + \\ (K_0 - K_{\inf})K_1 \int_0^t \zeta_K(t-\tau)\dot{\eta}(t)d\tau + K_2\eta(t) = f(t)\end{aligned} \tag{22}$$

where

$$M = \int_\Omega \rho(x)\Phi^T(x)\Phi(x)dx \tag{23}$$

is the mass matrix,

$$K_1 = (G_0 - G_{\inf})\int_\Omega B^T D_G B dx + (K_0 - K_{\inf})\int_\Omega B^T D_K B dx \tag{24}$$

$$K_2 = G_{\inf}\int_\Omega B^T D_G B dx + K_{\inf}\int_\Omega B^T D_K B dx \tag{25}$$

are the stiffness matrices, and

$$f(t) = \int_\Omega f(x,t) \cdot v(x) dx + \int_{\Gamma_N} g(x,t) \cdot v(x) ds \tag{26}$$

is the right hand side. The corresponding element matrices are defined simply by breaking the integrals into elementwise contributions.

Equation 22 represents a system of Volterra integro-differential equations. Without the inertial term, 22 represents a system of Volterra integral equations of the first kind. We now consider implicit schemes for integrating these equations in time. The goal is to reduce the system of equations 22 to a system in standard form

$$M\ddot{\eta}(t) + C\dot{\eta}(t) + K\eta(t) = \hat{f}(t) \tag{27}$$

where C is a *constant* damping matrix, and $\hat{f(t)}$ is a modified right hand side that will include a portion of the viscoelastic convolution term. We demand that C be independent of time, since this will eliminate the need for refactoring the left hand side at each time step. The damping (integral) term in equation 22 is certainly time-dependent. However, we will show that it is possible to split this integral term into a time-dependent and a time-independent part. The time-independent parts remain on the left hand side and become the damping matrix, whereas the time-dependent parts can be carried to the right hand side, since they are known quantities. Once the equations 22 are reduced to the system 27, the standard time integrators for structural dynamics can be employed.

For simplicity, we consider the case of only a single Prony series term. The results for more terms can be obtained by adding together the results for a single term. The integral in equation 22 can be split into two parts (considering only a single Prony series term)

$$\int_0^t e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau = \int_0^{t_i} e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau + \int_{t_i}^t e^{\frac{t-\tau}{a}} \dot{\eta}(t) d\tau \tag{28}$$

$$= e^{\frac{\Delta t}{s}} \int_0^{t_i} e^{\frac{t_i-\tau}{s}} \dot{\eta}(t) d\tau + \int_{t_i}^t e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau \tag{29}$$

where the first term is a loading history term that is *known* at time t_i. Consequently, it can be treated as an additional load and brought to the right hand side. The remaining term can be split into two terms, one containing coefficients of $\dot{\eta}$, and the other containing coefficients of $\dot{\eta}_i$. The former is unknown and thus becomes $C\dot{\eta}$, whereas the latter is known and thus also contributes to the right hand side.

In order to evaluate the term

$$\int_{t_i}^t e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau \tag{30}$$

we first need a representation for the velocity $\dot{\eta}(t)$ in the interval $t \in [t_i, t]$. We present two choices, both of which are second order accurate.

1.4.3 Linear Representation of Velocity

The first is consistent with the Newmark-beta method, which presumes a constant acceleration within the time step. With this assumption, the velocity must vary linearly within the time step. Thus,

$$\dot{\eta}(t) = \dot{\eta}(t_i) + \frac{\ddot{\eta} + \ddot{\eta}(t_i)}{2}(t - t_i) \tag{31}$$

where $\ddot{\eta}$ is the (unknown) acceleration at current time t, and $\ddot{\eta}(t_i)$ is the previous acceleration. Although equation 31 is the correct representation for velocity, it is inconvenient in that it would lead to (after inserting into equation 30) a contribution to the mass matrix. This is undesrirable, since it would interfere with the use of a lumped mass matrix. Thus, we re-write the velocity distribution in an equivalent form

$$\eta(t) = \eta(t_i) + \frac{\dot{\eta} - \dot{\eta}(t_i)}{\Delta t}(t - t_i) \tag{32}$$

We note that equations 31 and 32 are equivalent representations of the velocity. By inserting equation 32 into equation 30 we obtain

$$\int_{t_i}^{t} e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau = \left[s + \frac{s^2}{\Delta t}\left(e^{\frac{\Delta t}{s}} - 1 \right) \right] \dot{\eta} + \left[-s e^{\frac{-\Delta t}{s}} + \frac{s^2}{\Delta t}\left(1 - e^{\frac{-\Delta t}{s}} \right) \right] \dot{\eta}_i \tag{33}$$

The first term involves a coefficient times the unknown $\dot{\eta}$, which is the unknown velocity at the current time, and thus it must remain on the left hand side as a damping term contribution. The damping matrix implied by this term is

$$C = c_K(s_K + \frac{s_K^2}{\Delta t}(e^{\frac{-\Delta t}{s_K}} - 1))\mathbf{B^T D_K B} + c_G(s_G + \frac{s_G^2}{\Delta t}(e^{\frac{-\Delta t}{s_G}} - 1))\mathbf{B^T D_G B} \tag{34}$$

The second term is known, and thus it can be added to the load vector.

1.4.4 Midpoint Representation of Velocity

A second implicit scheme can be derived simply by using the midpoint rule on the velocity in the viscoelastic term. The only difference from the linear approach described above is in equation 33.

$$\dot{\eta}(t) = \frac{\dot{\eta} + \dot{\eta}(t_i)}{2} \tag{35}$$

This leads to

$$\int_{t_i}^{t} e^{\frac{t-\tau}{s}} \dot{\eta}(t) d\tau = \frac{s}{2}\left(1 - e^{\frac{\Delta t}{s}}\right)\dot{\eta} + \frac{s}{2}\left(1 - e^{\frac{\Delta t}{s}}\right)\dot{\eta}_i \tag{36}$$

In the same way as for the linear velocity approach, we use the term involving $\dot{\eta}$ to construct a damping matrix, and the remaining known terms are carried to the right hand side.

It should be noted that the midpoint scheme is inconsistent in that a different discretization scheme is used for the viscoelastic term than was used for the overall time integration. The linear represenation of velocity is a consistent scheme. However, both approaches are second order accurate.

1.5 Random Vibration

Details of random vibration analysis are included in a number of papers[2]. These few paragraphs document what was implemented.

1.5.1 algorithm

The first step in the calculation is computation of a modal spatial contribution, 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,

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

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.

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

[2] see for example, reference 3.

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.

1.5.2 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', and the superscript '*' denotes complex conjugate. 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.

$$G_{mn}(f) = \sum_{i,j} \phi_{im} \phi_{jn} \mathcal{G}_{ij}(f) \tag{37}$$

where

$$\mathcal{G}_{ij}(f) = q^*_i(f) q_j(f) \sum_{a,a'} Z^i_a Z^j_{a'} S^{a,a'}(f) \tag{38}$$

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 38 are completely known on each subdomain.

1.5.3 RMS Output

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

$$\begin{aligned} X_{rms} &= \sqrt{\int G_{mm}(f)df} \\ &= \sqrt{\int \sum_{i,j} \phi_{im}\phi_{jm}\mathcal{G}_{ij}(f)df} \\ &= \sqrt{\sum_{i,j} \phi_{im}\phi_{jm}\Gamma_{ij}} \end{aligned}$$

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 $G_{mm}(\omega)\omega^4$. Subsection 2.18.5 provides details about transforming power spectra to an output coordinate system.

1.5.4 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

1.5.5 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 $Q^{\dagger}BQ$ 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.

1.5.6 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, $X[(\texttt{Cdim})]/\texttt{X[0]} > \texttt{10}^{-\texttt{14}}$. 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.

1.6 Modal Frequency Response Methods

The Salinas implementation of the modal acceleration method is described in this section. Separate cases are considered when the structure does and does not have rigid body modes.

1.6.1 No Rigid Body Modes

We first consider the frequency domain version of the equations of motion.

$$(-\omega^2 M + j\omega C + K)\hat{u} = \hat{f} \tag{39}$$

Consider the modal approximation

$$\hat{u} \approx \sum_{i=1}^{N} \phi_i q_i \tag{40}$$

where N is the number of retained modes, ϕ_i is the i'th mode shape, and q_i is the i'th modal dof. For modal damping, one obtains the uncoupled equations

$$(-\omega^2 m_i + j\omega c_i + k_i)q_i = \phi_i^T \hat{f} \tag{41}$$

for $i = 1, \ldots, N$ where

$$\begin{aligned} m_i &= \phi_i^T M \phi_i && (42)\\ c_i &= \phi_i^T C \phi_i && (43)\\ k_i &= \phi_i^T K \phi_i && (44)\\ & && (45) \end{aligned}$$

are the modal mass, modal damping, and modal stiffness of the i'th mode. Solving equation 41 for q_i leads to

$$q_i = (\phi_i^T \hat{f})/(-\omega^2 m_i + j\omega c_i + k_i) \tag{46}$$

Replacing $(-\omega^2 M + j\omega C)\hat{u}$ in equation 39 with the modal approximation

$$(-\omega^2 M + j\omega C) \sum_{i=1}^{N} \phi_i q_i \tag{47}$$

leads to

$$K\hat{u} = \hat{f} + (\omega^2 M - j\omega C) \sum_{i=1}^{N} \phi_i q_i \tag{48}$$

Recall that the mode shapes satisfy the eigenproblem

$$K\phi_i = \omega_i^2 M\phi_i \tag{49}$$

where ω_i is the circular frequency of the i'th mode. Provided $\omega_i \neq 0$, one obtains

$$K^{-1}M\phi_i = \phi_i/\omega_i^2 \tag{50}$$

In addition, see Eq. (18.14) of Craig, the damping matrix C can be expressed as

$$C = \sum_{i=1}^{N} \left(\frac{2\zeta_i \omega_i}{m_i} \right) (M\phi_i)(M\phi_i)^T \tag{51}$$

where ζ_i is the damping ratio of the i'th mode. Substituting equations 50 and 51 into equation 48 and solving for $\hat{u}$ leads to

$$\hat{u} = K^{-1}\hat{f} + \sum_{i=1}^{N} (\omega^2/\omega_i^2 - 2\zeta_i j\omega/\omega_i)\phi_i q_i \tag{52}$$

The acceleration frequency response, $\hat{a}$, can be obtained by multiplying equation 52 by $-\omega^2$.

1.6.2 Rigid Body Modes

The procedure outlined here describes how the modal acceleration method can be used in the case when the structure has rigid body modes. The main difference between the approach presented here and Craig's method[4] (pp. 368-371) is in the way that the flexible response is computed using the singular stiffness matrix. Craig removes the rigid body modes from the stiffness matrix using constraints. In our approach, we first orthogonalize the right hand side with respect to the rigid body modes, and then use an iterative solver such as FETI to solve the singular system directly. Although the two methods are equivalent the latter is much more convenient from the implementation point of view. Note, however, that the implementation is likely to fail on a single processor since the direct solvers in Salinas are unable to manage a singular stiffness matrix.

The equations of interest are the frequency domain equations of motion

$$-\omega^2 Mu + j\omega Cu + Ku = f \tag{53}$$

Since the stiffness matrix may be singular, we first split the solution into a rigid body part and a flexible part.

$$u(\omega) = u_R(\omega) + u_E(\omega) \tag{54}$$
$$= \Phi_R q_R(\omega) + \Phi_E q_E(\omega) \tag{55}$$

where the subscript R refers to rigid body mode contributions, and E refers to contributions from flexible modes. We define N as the total number of degrees of freedom, N_R as the number of rigid body modes and N_E the number of flexible modes, where $N = N_R + N_E$. Then, Φ_R is an NxN_R matrix of rigid body eigenvectors, Φ_E is an NxN_E matrix of flexible eigenvectors, q_R is a vector of dimension N_R, and q_E is a vector of dimension N_E. We assume mass normalized eigenvectors.

We now substitute equation 55 into equation 53, and premultiply both sides by Φ_R^T and Φ_E^T. This yields two sets of equations, after using orthogonality and the fact that $K\Phi_R = 0$.

$$-\omega^2 q_R + j\omega C_R q_R = \Phi_R^T f \tag{56}$$
$$-\omega^2 q_E + j\omega C_E q_E + K_E q_E = \Phi_E^T f \tag{57}$$

where C_R, C_E are diagonal matrices containing the modal damping contributions, and K_E is a diagonal matrix containing the eigenvalues. In particular, the ith diagonal entry of C_E is $2\omega_i\zeta_{E_i}$, and the ith diagonal entry of C_R is $2\omega_i\zeta_{R_i}$. For most applications, C_R is null. Solving these equations we obtain the component-wise values of the coefficients

$$q_{R_i} = \frac{\Phi_{R_i}^T f}{-\omega^2 + j\omega C_{R_i}} \tag{58}$$

$$q_{E_i} = \frac{\Phi_{E_i}^T f}{-\omega^2 + j\omega C_{E_i} + \omega_E^2} \tag{59}$$

Equation 57 can be solved for q_E, and substituting this into equation 55, we obtain

$$u = \Phi_R q_R + \Phi_E K_E^{-1} \Phi_E^T f + \omega^2 \Phi_E K_E^{-1} q_E - j\omega \Phi_E K_E^{-1} C_E q_E \tag{60}$$

The first term in equation 60 is known. The third and fourth terms of equation 60 can be computed by modal truncation, and in fact these are the same as the second and third terms of equation 52. The second term in equation 60 is the static correction, and is not readily computable in the present form since all of the flexible modes would have to be known to compute it.

In order to compute the second term in equation 60, we note that the matrix $a_E = \Phi_E K_E^{-1} \Phi_E^T$ is the inverse of the elastic stiffness matrix, that is, the stiffness

matrix without the rigid body components. Craig gives a procedure of constraining the rigid body modes in the stiffness matrix in order to compute the product $a_E f$. This procedure would require re-sizing the global stiffness matrix midway through the modalfrf solution procedure, and this is tedius from the code development standpoint.

A more convenient approach is to use FETI to solve the system $Ku = f_E$, where f_E is obtained by orthogonalizing the right hand side f with respect to the rigid body modes, via Gram Schmidt. We note that FETI can solve problems of the form $Ku = f$ even if K is singular, provided that the right hand side f is orthogonal to the rigid body modes.

The procedure is to first apply Gram Schmidt orthogonalization to obtain f_E. Then, we use FETI to solve the system $Ku_E = f_E$, where K is singular. Finally, to be sure u_E is orthogonal to the rigid body modes, we apply Gram Schmidt one more time to u_E. Though in theory u_E is already orthogonal to the rigid body modes after the FETI solve, numerical roundoff may result in a small loss of orthogonality (especially if the solver tolerance is loose), and thus we apply this final orthogonalization to u_E to be on the safe side. The resulting solution we again denote by u_E. Then,

$$u_E = \Phi_E K_E^{-1} \Phi_E^T f \tag{61}$$

and thus all of the terms in equation 60 are known. Thus the modal frequency response can be computed using equation 60.

We note that the orthogonalizations refered to above involve only the standard dot products. That is, in order to make f orthogonal to one rigid body mode ϕ_i, the Gram Schmidt factor is

$$\alpha = \frac{\phi_i^T f}{\phi_i^T \phi_i} \tag{62}$$

and then

$$f_E = f - \alpha\phi \tag{63}$$

The dot products appearing in these expressions do not involve the mass matrix. They are the standard dot products.

1.6.3 Example

Finally, we present an example of the performance of this method as compared to the stndard modal displacement method. The example is a beam composed of 320 hex8 elements. The beam is free-free, so that all rigid body modes are present. The frequency response is computed up to 9000 Hz, and 15 modes are used in the modal expansions. The 15th mode had a frequency of 11362 Hz. In Figure 1, the two

methods are compared with the direct frequency response approach. It is seen that the modal acceleration method gives a significantly improved performance over the modal displacement method.

1.7 Complex Eigen Analysis - Modal Analysis of Damped Structures

1.7.1 Modal Analysis of Damped Structures

Salinas will solve the eigenvalue problems for structures with some types of damping. The algorithms are designed for internally damped structures such as from viscoelastic materials. The package is called `Ceigen`, and the parameters to be aware of are `eig_tol`, `nmodes`, and `viscofreq`. The first two parameters, `eig_tol` and `nmodes` will be familiar to Salinas users that solve eigenvalue problem for undamped structures. `eig_tol` is the convergence tolerance for the eigenvalues, and `nmodes` is the number of requested eigenvalues. `viscofreq` approximates the first flexible mode of the structure. The default value for `eig_tol` is $1.e-8$.

The complex eigen value problem which we solve is also known as the quadratic eigenvalue equation.

$$\left[K + \lambda D + \lambda^2 M\right] \phi = 0 \tag{64}$$

where,

$$\begin{aligned} K &= \text{the stiffness matrix} \\ D &= \text{the damping matrix} \\ M &= \text{the mass matrix} \\ \lambda &= \text{the complex frequency.} \end{aligned}$$

All of the matrices are independent of frequency. Note that we are solving for $\lambda = i\omega + \gamma$, not ω^2.

1.7.2 Input File Specification

The Salinas input file specification is similar to the specification for transient simulations. To change a working Salinas input file for a transient problem into a Salinas input file for `Ceigen`, change the Solution and Parameters blocks. The example below illustrates how the Solution and Parameter blocks are modified for modal analyses.

```
SOLUTION
case ceig
ceigen nmodes 20
```

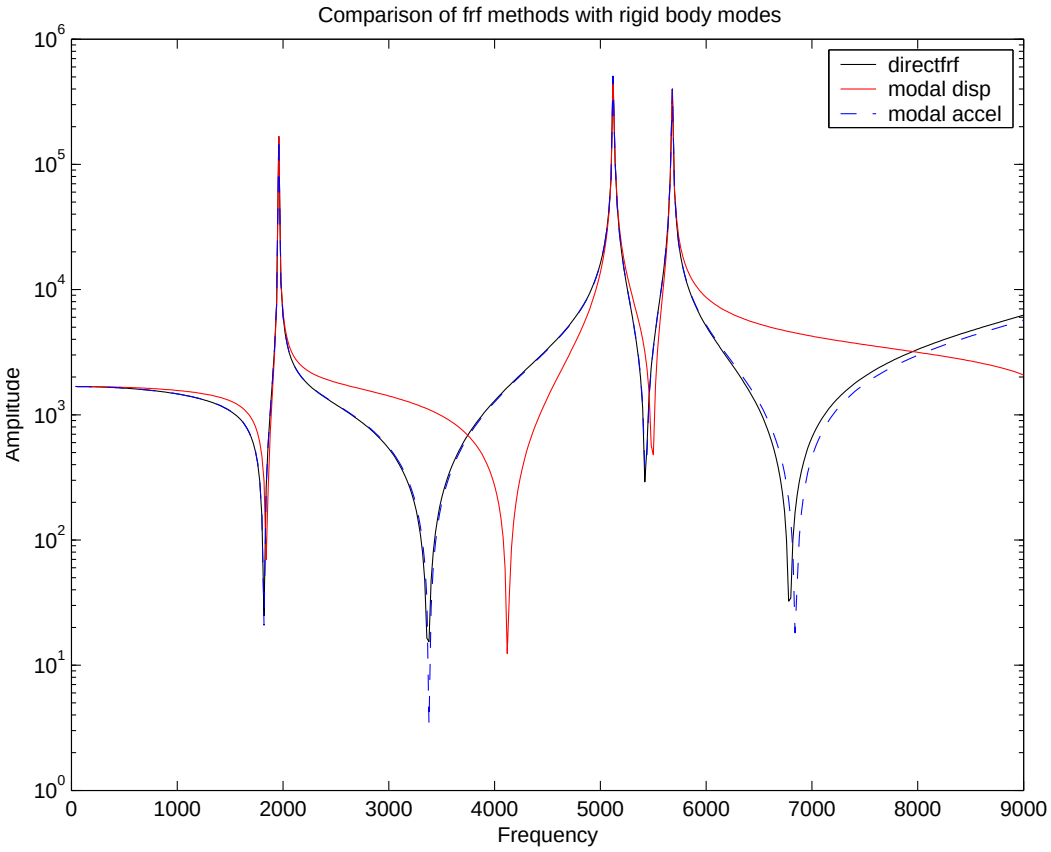

Figure 1: A comparison of the modal displacement, modal acceleration, and direct frequency response approaches. The modal acceleration method gives a better approximation to the direct approach than the modal displacement method.

```
viscofreq=1.e+4
END
PARAMETERS
eig_tol 1.E-5
wtmass=0.00259
END
```

The parameter wtmass is an example of a parameter that was was needed for the transient simulation, and is still needed for modal analyses.

1.7.3 Output File Format

The output is very similar to the output for the undamped eigenvalue problem. The results file contains any requested data. Supplemental information is written to the screen that is useful for algorithm development.

The Results file `foo.rslt` tabulates the values $\lambda/(2\pi)$ for (λ_i) that solve equation (65). Pure real eigenvalues are not written to the Results file.[3] If λ_i has been found with i in the range, $1 \leq i \leq 24, 27 \leq i \leq 34$, then the missing eigenvalues $(\lambda_i)_{25 \leq i \leq 26}$ are real eigenvalues that are omitted. The number of eigenvalues written in the Results file is less than or equal to `nmodes`.

As is the case with the undamped eigenvalue problem, Salinas will print a table to the screen. The table is titled "Ritz values (Real, Imag) and direct residuals", and has four columns of real numbers. The number of eigenvalues that are actually computed may be larger or smaller than the number requested. Some real eigenvalues may appear among the converged eigenvalues. The table will contain any converged real eigenvalues (zero in column two). Columns three and four are two different residual norms for each eigenvalue. Eigenvalues with large residual norms are not converged. The residual norm in the third column is less sensitive to the linear system relative residual norm bound than the residual norm in the fourth column is After each implicit restart, all the approximate eigenvalues are printed to the screen.

1.7.4 Some Back Ground

The eigenvalue problem for an undamped structure

$$\mathbf{K}\Phi = \mathbf{M}\Phi\Omega^2, \quad \Phi^T\mathbf{M}\Phi = I,$$

$\Omega = \oplus_i \omega_i$, has been discussed elsewhere in this document. Salinas returns the frequencies $\omega/(2\pi)$. `Ceigen` solves a similar problem. `Ceigen` solves the quadratic

[3] Real modes correspond to an overdamped mode with no oscillatory component. These are usually generated from numerical artifacts discussed below, and are seldom of practical value

eigenvalue problem

$$[\mathbf{M}\lambda^2 + \mathbf{D}\lambda + \mathbf{K}]u = 0, \quad u^T u = 1. \tag{65}$$

In the undamped case, $\mathbf{D} = \mathbf{0}$, $\lambda = i\omega$.

A second order linear differential equation is the same as a first order system. Similarly a quadratic eigenvalue problem is the same as a matrix eigenvalue problem of twice the size.

Linear problems such as matrix eigenvalue problems are solvable in that it is possible to find all of the solutions. For matrix eigenvalue problems the key idea is deflation. One big subspace is used to compute all of the eigenvalues. Small eigenvalues tend to be computed early and are deflated from the problem. The reward for deflation is that the gravest remaining eigenvalues are much more likely to be computed next. For general nonlinear eigenvalue problems on the other hand, no robust algorithms are known to the author.

1.7.5 Viscoelasticity

The eigenvalue problem for viscoelastic problems[5] in the most simple case (one term Prony series) has the form

$$[\mathbf{M}s^2 + \mathbf{D}(s)s + \mathbf{K}]u = 0. \tag{66}$$

$$\mathbf{K} = \mathbf{B}E_\infty, \ \mathbf{D}(s)s = \mathbf{B}(E_g - E_\infty)f(s),$$
$$f(s) = s/(s+a) = 1 - (s/a+1)^{-1}.$$

Prony series damping in the time domain[5] creates a frequency domain problem with real eigenvalues that are not physical.[5] Some care is needed to avoid the real eigenvalues in computations.

Here is a sketch of justification that the Prony series problem has real eigenvalues. The eigenvalue problem has a closed form solution in terms of the eigenvalues of the undamped problem. The one term Prony series damping increases the degree of the characteristic equation from two to three, and the third root must be real.

1.7.6 Viscofreq

The eigenvalue problem in equation (66) is not a quadratic eigenvalue problem $(\mathbf{M}, \mathbf{D}, \mathbf{K})$. The obvious approximation is to evaluate $\mathbf{D}(s)$ at some fixed s_o near to the wanted eigenvalues. The user parameter `viscofreq`$= \omega$ is a real number such that $s_o = i\omega$. In a later release $s_o = r + i\omega$ for some internally computed value r.

Using a value of `viscofreq` that is much too small may degrade performance. As `viscofreq` increases, the eigenvalues do change, and Salinas converges more

quickly. The cluster of real eigenvalues moves left, away from zero, and it becomes possible to compute more of the complex eigenvalues. Over-estimates of `viscofreq` are safer than underestimates.

Suppose that $s_o = r + i\omega$. A different quadratic eigenvalue problem is used.[5] Both $\mathbf{D}$ and $\mathbf{K}$ are modified. The approximation is more accurate for problems in which r is much more accurate than ω. Also $(\mathbf{M}, \mathbf{D}, \mathbf{K})$ are all real matrices. The eigenvalues and eigenvectors come in complex conjugate pairs.

Important to be aware that no constant damping matrix inherits the property of $\mathbf{D}(s)$ that

$$\lim_{s\to\infty} \mathbf{D}(s) = 0.$$

Physically, this means that the eigenvalues in equation (65) that are far from `viscofreq` are over-damped. If for a given mode shape, s_o is closer to the real eigenvalue of equation (66) than either complex conjugate pair, then `Ceigen` may return the real eigenvalue. For example equation (66) has many real eigenvalues clustered left of $-a$.

1.7.7 Trust Regions and Real Modes

The eigenvalue problem is solved using ARPACK. The convergence criteria in the ARPACK package use a trust region. CEigen will compute the right-most eigenvalues of the eigenvalue problem in equation (qevp). If the k-th mode does not satisfy the convergence tolerance, and $k \leq$`nmodes`, then ARPACK is not converged, no matter how many other eigenvalues are converged.

The authors have gone to great lengths to filter out real eigenvalues. Nonetheless in problems with a cluster of real eigenvalues among the right-most eigenvalues, it is very difficult to compute eigenvalues high into the frequency range. If such a problem arises, increase `eig_tol` (multiply by ten), increase `nmodes` (add ten), and most importantly increase `viscofreq` (double).

1.7.8 ViscoFreq - Approximating the Response of Viscoelastics

The viscoelastic mass matrix can be considered to be independent of frequency. However, the damping and stiffness matrices can be functions of frequency, depending on the formulation. There are two possible formulations. The first one results in a complex, frequency dependent damping matrix, and a real-valued, frequency independent stiffness matrix. The second results in a frequency- dependent, real-valued damping matrix and a frequency-depenedent, real valued stiffness matrix. We chose the second formulation since the complex-valued damping matrix is somewhat difficult to deal with in quadratic eigensolvers. The two formulations are the same up to the order of the linearization error.

Consider the simplest possible viscoelastic material, characterized by a single term of the Prony series. The equation of motion for a 1D system with this material is given below. The full 3D case is similar, except that it has separate terms for the bulk and shear components.

$$\left[K_\infty + sD(s) - s^2 M\right] u = f(s) \tag{67}$$

Here, s is the Laplace transform frequency, $f(s)$ is the frequency dependent force, and the damping matrix is now a function of frequency.

$$D(s) = (E_G - E_\infty)\frac{1}{s + 1/\tau}B \tag{68}$$

with E_∞, the Young's modulus for high frequencies, E_G the modulus for low (or glassy) frequencies, τ is the Prony series relaxation time, and $K_\infty = E_\infty B$ is the stiffness at high frequencies.

We now return to equation 67, and consider different ways of linearizing the relation, since for the quadratic eigenvalue problem, we may only solve equations of the form in equation 64, i.e. quadratic in λ or s.

User Specified frequency of linearization We define viscofreq, ω and $s_\omega = r + i\omega$, which is the complex number about which the linearization takes place. In the current methodology, r is zero.

First, we split $D(s_\omega)$ into its real and imaginary components by multiplying by $\frac{(r+1)-i\omega\tau}{(r+1)-i\omega\tau}$.

$$\begin{aligned} D(s) &= (E_G - E_\infty)\frac{1}{s + 1/\tau}B && (69)\\ &= (E_G - E_\infty)\frac{\tau}{i\omega\tau + (r\tau + 1)}B && (70)\\ &= \frac{\tau((r\tau + 1) - i\omega\tau)}{(r\tau + 1)^2 + \omega^2\tau^2}(E_G - E_\infty)B && (71) \end{aligned}$$

Then we also temporarily replace the s in front of $sD(s)$ with s_ω. This gives,

$$\begin{aligned} sD(s) &= (i\omega + r)D(i\omega + r) && (72)\\ &= \frac{\tau(i\omega + r) + \omega^2\tau^2 + r^2\tau^2}{(r + 1)^2 + \omega^2\tau^2}(E_G - E_\infty)B && (73) \end{aligned}$$

Finally, we replace $i\omega + r$ with s to go back to the quadratic eigenvalue problem. This results in a contribution to the the stiffness matrix, and a real damping matrix.

$$\left[\left(E_\infty + (E_G - E_\infty)\frac{\omega^2\tau^2 + r^2\tau^2}{(r+1)^2 + \omega^2\tau^2}\right)B + s\left(\frac{\tau}{(r+1)^2 + \omega^2\tau^2}\right)(E_G - E_\infty)B + s^2 M\right]\phi = 0 \tag{74}$$

Thus we see that the damping matrix is purely real, but the stiffness matrix gets an additional (positive) real contribution.

Practically of course, the systems are far more complex. Typically there is more than one material, and that material has a number of Prony terms. Equation 74 is modified, but the overall effect is the same, i.e. the stiffness matrix is increased by a viscoelastic term, and the damping term is also modified. Effectively we have the following.

$$\tilde{K}(r + i\omega) = \sum_{elem} \tilde{K}_{elem}(r + i\omega) \tag{75}$$

where $\tilde{K}_{elem}$ is the modified stiffness matrix.

$$\tilde{K}_{elem}(r + i\omega) = K_{elem} + \text{imag}(D_{elem}(r + i\omega))$$

Likewise,

$$\tilde{D}_{elem}(r + i\omega) = \text{real}(D(r + i\omega)) \tag{76}$$

We now solve the *linearized* eigenvalue equation for λ,

$$\left[\tilde{K}(r + i\omega) + i\lambda\tilde{D}(r + i\omega) - \lambda^2 M\right]\phi = 0 \tag{77}$$

A Simple Error Estimate This question is now how well the eigenvalues computed from equation 74 approximate the true eigenvalues of equation 67.

First, we define the distance from a given computed eigenvalue, s_c, to the point of linearization, s_ω as δ.

$$\delta = s_c - s_\omega \tag{78}$$

Note that δ is a complex-valued quantity.

Next, we define the residual as the vector resulting from inserting s_c and the corresponding computed eigenvalue, ϕ_c, into equation 67.

$$\left(s_c^2 M + s_c D(s_c) + K\right)\phi_c = res \tag{79}$$

The residual, as defined in equation 79, is a computable quantity. Obviously, if the residual is large, then the error in the computed eigenvalue and eigenvector is large. However, the more interesting question from the analyst's perspective is how large may δ be for one to expect accurate eigenvalues.

1.8 Component Mode Synthesis

Component mode synthesis in Salinas follows the Craig-Bampton method. In this method the model is reduced using fixed interface modes and constraint modes. The method is outlined in some detail in Craig's book, (Chapter 19 of 4). It is summarized below. Note that in Salinas we do *not* permit any flexibility in the interface boundary options. Only fixed interface modes are supported.

CMS is typically applied to eigenvalue analysis, but it may be used in other solution methods as well. Here we describe only the eigen analysis application. Within Salinas only a subset of the standard CMS method is available. Salinas may reduce *an entire model* to a set of interface degrees of freedom with the corresponding system matrices and transfer matrices. Salinas will also (eventually) be capable of reading in a reduced system for solution within its framework.

CMS by these methods is always a linear model, with supports only linear elasticity. The reduction is based on an eigen reduction and linear superposition.

1.8.1 Reduction of superelement matrices

The entire model of a structure may be reduced to the interface degrees of freedom and generalized degrees of freedom associated with internal modes of vibration. Consider the general eigenvalue problem, with the system matrices partitioned into interface degrees of freedom, C, and the complement, V.

$$\left(\begin{bmatrix} K_{vv} & K_{vc} \\ K_{cv} & K_{cc} \end{bmatrix} - \lambda \begin{bmatrix} M_{vv} & M_{vc} \\ M_{cv} & M_{cc} \end{bmatrix} \right) \begin{bmatrix} u_v \\ u_c \end{bmatrix} = 0 \tag{80}$$

Within Salinas we consider only the cases where K_{vv} is nonsingular. For the Craig-Bampton method this implies that clamping the interface degrees of freedom removes all zero energy modes from the structure.

The Craig-Bampton method reduces the physical degrees of freedom, u, to generalized coordinates, p, using a set of preselected component modes, Ψ.

$$u = \Psi p \tag{81}$$

The component modes are selected as follows. We let $\Psi = [\Phi \ \psi]$, where Φ is a set of eigen modes of the fixed interface, i.e.,

$$(K_{vv} - \lambda M_{vv})\, \Phi = 0$$

We retain only a subset of the modes in this system. In addition, we define the constraint modes, ψ, as the static condensation of the problem. Each column of ψ is the solution of the static problem where one interface degree of freedom has

unit displacement, and all other interface degrees of freedom are fixed. As shown in Craig,

$$\psi = -K_{vv}^{-1} K_{vc} \tag{82}$$

Note that since we require that K_{vv} be positive definite, all these solutions are well defined. The matrix need be factored only once for all the modes.

Reduced System

As shown in *Craig*, the reduced system matrices can be written as follows.

$$\mu = \begin{bmatrix} \mu_{kk} & \mu_{kc} \\ \mu_{ck} & \mu_{cc} \end{bmatrix} \tag{83}$$

and,

$$\kappa = \begin{bmatrix} \kappa_{kk} & \kappa_{kc} \\ \kappa_{ck} & \kappa_{cc} \end{bmatrix} \tag{84}$$

where,

$$\begin{aligned} \mu_{kk} &= I_{kk} \\ \mu_{kc} &= \mu_{ck}^T = \phi^T (M_{vv}\psi + M_{vc}) \\ &= \phi^T M_{vv}\psi + (M_{cv}\phi)^T \\ \mu_{cc} &= \psi^T (M_{vv}\psi + M_{vc}) + M_{cv}\psi + M_{cc} \\ &= \psi^T M_{vv}\psi + (M_{cv}\psi)^T + M_{cv}\psi + M_{cc} \end{aligned} \tag{85}$$

and,

$$\begin{aligned} \kappa_{kk} &= \Lambda_{kk} \\ \kappa_{kc} &= \kappa_{ck} = 0 \\ \kappa_{cc} &= K_{cc} - K_{cv} K_{vv}^{-1} K_{vc} \\ &= K_{cc} + K_{cv}\psi \end{aligned} \tag{86}$$

Note that the coupling between the modal and interface portion of the system matrix occurs only in the mass matrix.

Parallelization Issues

The discussion above applies simply for direct solvers for which a system matrix is generated. Parallelization issues are straightforward, and cover 3 main areas 1) computaion of fixed interface modes, 2) computation of constraint modes, and 3) matrix vector products.

1. **Fixed Interface Modes.** Since the process of cmputation of the eigensystem is independt of the particular solver, there are no parallelization issues with respectt to the eigenvalue problem. It is easily shown that parallel solvers result in the same eiben pairs as serial solvers. There is no reason to expect that any finite precision issues would be more important here than in other modal based solutions.

2. **Constraint Modes.** The constraint modes are different, in that we do not currently have a capability to compute enforced displacement in parallel. Recall that the constraint mode is the displacement on space "V" that is computed when a unit displacement is applied to as single degree of freedom on the interface. The serial equations are as follows.

$$\begin{bmatrix} K_{vv} & K_{vc} \\ K_{cv} & K_{cc} \end{bmatrix} \begin{bmatrix} u_v \\ u_c \end{bmatrix} = \begin{bmatrix} 0 \\ R \end{bmatrix} \tag{87}$$

   Equation 82 uses the first of these only to solve for $u_v = \psi$. For a domain decomposition problem, the system matrices are written differently. We examine a two subdomain problem for clarity.

$$\begin{bmatrix} K_{1vv} & K_{1vc} & 0 & 0 & C_{1v}^T \\ K_{1cv} & K_{1cc} & 0 & 0 & C_{1c}^T \\ 0 & 0 & K_{2vv} & K_{2vc} & C_{2v}^T \\ 0 & 0 & K_{2cv} & K_{2cc} & C_{2c}^T \\ C_{1v} & C_{1c} & C_{2v} & C_{2c} & 0 \end{bmatrix} \begin{bmatrix} u_{1v} \\ u_{1c} \\ u_{2v} \\ u_{2c} \\ \mu \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ R \end{bmatrix} \tag{88}$$

   We extract only the first and third rows to arrive at,

$$\begin{bmatrix} K_{1vv} & 0 & C_{1v}^T \\ 0 & K_{2vv} & C_{2v}^T \end{bmatrix} \begin{bmatrix} u_{1v} \\ u_{2v} \\ \mu \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{89}$$

   Here $f_i = K_{ivc} u_{ic}$. This system is the standard system of equations that is solved by the domain decomposition solver. The RHS is just the sum of the individual subdomain terms.

3. **Matrix Vector Products.** Clearly matrix vector and vector-vector products must be computed differently in parallel. However, the code to do this is already being used in the eigensolver.

1.9 A posteriori error estimation for eigen analysis

The purpose of this section is to summarize two different approaches for a posteriori error estimation of eigen analysis. The first is an explicit error estimator,[6],[7] and the second is a quantity of interest approach.[8] The explicit approaches are described in chapter 2 of,[9] and the quantity of interest approaches are described in chapter 8 of the same book. However, since we are interested in the eigenvalue problem, the methodologies are somewhat different than the approaches described in,[9] though there are many similarities. Both the explicit and the quantity of interest approaches have the same goal - to use the computed solution to compute upper and lower bounds on the discretization error for the eigenvalues and eigenvectors. A drawback to the explicit approach is that unknown constants are present in the bounds, making final determination of the error more difficult. Because of this, explicit estimators are more frequently used as element indicators to drive adaptivity algorithms, rather than as error estimators. The quantity of interest approach avoids the unknown constants, but is more work in terms of implementation.

1.9.1 Preliminaries

We seek a posteriori bounds on the error of the finite element solution of the eigenvalue problem for elasticity

$$-\rho\lambda u = (\Lambda + \mu)\nabla(\nabla \cdot u) + \mu\nabla^2 u = \nabla \cdot \sigma(u) \tag{90}$$

or

$$A_1(u) = -\lambda A_2(u) \tag{91}$$

where where $A_1(u)$ and $A_2(u)$ are the partial differential operators implied by equation 90, λ and u are the unknown eigenvector and eigenvalue, and Λ and μ are the Lamé elasticity constants. We note that the right hand side of equation 90 can be written either in terms of displacement, as in the first representation, or in terms of stress, as in the second representation of the right hand side of the equation. The weak formulation of equation 90 is constructed by multiplying by a test function, and integrating by parts, with homogeneous boundary conditions. This leads to the weak formulation: Find $(\lambda, u) \in V \times R$ such that

$$B(u, v) = \lambda M(u, v) \quad \forall v \in V \tag{92}$$

where

$$B(u, v) = \int_\Omega \sigma(u)\epsilon(v)dx \tag{93}$$

and

$$M(u, v) = \int_\Omega \rho uv dx \tag{94}$$

After defining a finite element discretization, this reduces to: Find (u_h, λ_h) such that

$$Ku = \lambda Mu \tag{95}$$

where (u_h, λ_h) are the finite element approximations of the eigenvector and eigenvalue, and K, M, are the assembled stiffness and mass matrices.

1.9.2 Approach I - explicit error estimator

In *Larsen*[6] and *Rannacher*,[7] two independently derived error estimates are presented for the Laplace equation. While the two estimates differ slightly, both incorporate an unknown constant, C, an element diameter term, h_e, and an element residual function, $\bar{\rho}$. In what follows we extend these estimates to the elasticity problem. The following two error estimates are given in[6] and[7] respectively. In what follows we use Larsen's results (equation 96) exclusively. [4]

$$|\lambda - \lambda_h| \leq c\lambda C_{e,0} \left(\sum_{e=1}^{N_e} h_e^4 \bar{\rho}(u_h, \lambda_h)^2 \right)^{\frac{1}{2}} \tag{96}$$

$$|\lambda - \lambda_h| \leq C_2 \sum_{e=1}^{N_e} h_e^2 \bar{\rho}(u_h, \lambda_h)^2 \tag{97}$$

where h_e is the element diameter, and

$$\bar{\rho}(u_h, \lambda_h)^2 = \int_{\Omega_e} \left(|A_1 u_h + \lambda_h A_2 u_h| + R_{flux} \right)^2 d\Omega_e \tag{98}$$

The first term on the right hand side is the interior element residual, which is the differential stiffness operator A_1, defined in equation 91, applied to the computed element displacement combined with the computed eigenvalue times the differential mass operator A_2, also defined in equation 91, applied to the computed element displacement. This term is computed by representing the eigenvector as a summation

$$u_h(x) = \sum_{i=1}^{N} a_i N_i(x) \tag{99}$$

where a_i is the i^{th} entry in the eigenvector, and $N_i(x)$ is the i^{th} shape function, and then simply applying the gradient and divergence operators from equation 90 to the summation in equation 99.

[4] Equation 96 applies to elements with linear shape functions. The more general expression may be found in equation 146 or the reference.

We note that the quantity $A_1 u_h + \lambda_h A_2 u_h$ is expressed in the strong form, and thus is not the same as $Ku_h - \lambda_h M u_h$, though both expressions are on the element level. The difference can be seen by observing the first term $A_1 u_h$

$$A_1 u_h = \nabla \cdot \sigma(u_h) \tag{100}$$

That is, $A_1 u_h$ is the divergence of the stress (which is computed from the finite element displacement u_h). This is not the same as Ku_h, since Ku_h is in the weak form, and has been evaluated by integrating over the element against a test function. For example, if we consider linear elements, we have $A_1 u_h = \nabla \cdot \sigma(u_h) = 0$, since the stress is constant over the element. On the other hand, Ku_h is not zero.

The second term is the boundary or flux residual.

$$R_{flux} = (h_e vol(e))^{-1/2} \left[\int_{\Gamma_e} R^2 d\Gamma_e \right]^{1/2} \tag{101}$$

It has two different integrands depending on whether the face in question lies on a part of the boundary where traction or pressure boundary conditions are applied, or whether it is an interior face. When it lies on a boundary loaded face,

$$R = g - \sigma_{ij} n_j \tag{102}$$

where g is the applied traction or pressure load. Note that $g = 0$ for eigen problems. When the face is an interior face,

$$R = [\sigma_{ij} n_j] = \sigma^a_{ij} n_j - \sigma^b_{ij} n_j \tag{103}$$

where σ^a and σ^b are the stress tensors in the two adjacent elements, element 'a' and element 'b'. Note that because the integrand is squared, computing the flux residual in parallel requires parallel communication.

We note the intuitive nature of the upper bound in equation 96. As the element size h_e tends to zero, the right hand sides of the estimate goes to zero, due to the multiplication by the element sizes h_e. Keep in mind also that the $\bar{\rho}$ term includes an integral over a volume and that $\sum_{e=1}^{N_e} \|\text{const}\|$ is a constant.

There are two important issues in applying the results in Larsen's reference to general elasticity problems. The first of these is the the extension to elasticity. The second is the extension to multiple materials. These are covered in the following sections.

1.9.3 Extension of Estimators to Elasticity

This section was provided by Ulrich Hetmaniuk to help us with problems in scaling the Laplace equation to the elasticity problem. It addresses issues of both mass

and stiffness scaling. A similar development was provided by Clark Dohrman. The development herein builds upon Larsen's development 6, and uses quantities defined there.

We consider the eigenvalue problem

$$-\mu\Delta\mathbf{u} - (\Lambda+\mu)\nabla(\nabla\cdot\mathbf{u}) = -\nabla\cdot\sigma(\mathbf{u}) = \theta\rho\mathbf{u} \quad \text{in } \Omega \tag{104}$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } \partial\Omega \tag{105}$$

where the Lamé constants Λ and μ satisfy

$$\Lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)} \tag{106}$$

We define also a weak formulation: find $(\mathbf{u},\theta)\in\mathbb{V}\times\mathbb{R}$

$$a(\mathbf{u},\mathbf{v}) = \theta b(\mathbf{u},\mathbf{v}), \quad \forall\ \mathbf{v}\in\mathbb{V} \tag{107}$$

$$b(\mathbf{u},\mathbf{u}) = 1 \tag{108}$$

where

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega \sigma(\mathbf{u})\cdot\epsilon(\mathbf{v})dx \tag{109}$$

and

$$b(\mathbf{u},\mathbf{v}) = \int_\Omega \rho\mathbf{u}\cdot\mathbf{v}dx \tag{110}$$

We follow the approach in the paper by M. Larson to derive a posteriori error estimators. We use most of his notation.

Residual

The definition (3.7) for the residual becomes, on a triangle τ,

$$R(\mathbf{u}_h,\theta_h)_{|\tau} = \frac{1}{\sqrt{\rho}}|\nabla\cdot\sigma(\mathbf{u}_h)+\theta_h\rho\mathbf{u}_h| + \sqrt{\frac{1}{h\ vol(\tau)}\int_{\partial\tau\setminus\partial\Omega}\left(\mathbf{n}\cdot\left[\frac{\sigma(\mathbf{u}_h)}{2\sqrt{\rho}}\right]\right)^2} \tag{111}$$

Note that we have

$$R(\mathbf{u}_h,\theta_h) \equiv R(\mathbf{u}_h,\theta_h,\rho,E,\nu) \tag{112}$$

and that R satisfies the following scaling properties

$$R(\frac{\mathbf{u}_h}{\sqrt{\alpha}},\frac{\theta_h}{\alpha},\alpha\rho,E,\nu) = \frac{1}{\alpha}R(\mathbf{u}_h,\theta_h,\rho,E,\nu) \tag{113}$$

$$R(\mathbf{u}_h,\alpha\theta_h,\rho,\alpha E,\nu) = \alpha R(\mathbf{u}_h,\theta_h,\rho,E,\nu) \tag{114}$$

Stability estimates

The equation (3.10) becomes

$$||D^{2+s}\mathbf{v}|| \leq C_{e,s}\sqrt{b\left(\left(\frac{-1}{\rho}\nabla\cdot\sigma\right)^{1+s/2}(\mathbf{v}),\left(\frac{-1}{\rho}\nabla\cdot\sigma\right)^{1+s/2}(\mathbf{v})\right)} \tag{115}$$

Note that

$$\Lambda+\mu=\frac{E}{2(1+\nu)(1-2\nu)} \quad , \quad \frac{\mu}{\Lambda+\mu}=1-2\nu \tag{116}$$

Then, we get

$$C_{e,s}=c\frac{\rho^{(1+s)/2}}{(\Lambda+\mu)^{(2+s)/2}} \tag{117}$$

Note that we have

$$C_{e,s}\equiv C_{e,s}(\rho,E,\nu) \tag{118}$$

and that $C_{e,s}$ satisfies the following scaling properties

$$C_{e,s}(\alpha\rho,E,\nu) = \alpha^{(1+s)/2}C_{e,s}(\rho,E,\nu) \tag{119}$$

$$C_{e,s}(\rho,\alpha E,\nu) = \frac{1}{\alpha^{(2+s)/2}}C_{e,s}(\rho,E,\nu) \tag{120}$$

A posteriori estimates

We make also the assumption (2.6) : there are $0\leq\delta<1$ and $h_0>0$ such that

$$\max_{\theta_i\notin\Theta}\left|\frac{\theta_h-\theta}{\theta_i-\theta}\right|\leq\delta \quad , \quad ||Q_\Theta\mathbf{u}_h||^2\leq\delta \tag{121}$$

for all meshes such that $\max h(x)\leq h_0$. Using $p=1$, $k=2$, $\beta_0=0$, and $\beta_1=1$, the final estimate on the eigenvalues becomes

$$\frac{\theta_h-\theta}{\theta}\leq\frac{c}{1-\delta}C_{e,0}\sqrt{\rho}||h^2R(\mathbf{u}_h,\theta_h)|| \tag{122}$$

The estimates on the error in the discrete eigenvector are now

$$\sqrt{b(\mathbf{e}_\Theta,\mathbf{e}_\Theta)} \leq \frac{c}{1-\delta}C_{e,0}(1+\max_{\theta_i\notin\Theta}\frac{\theta}{|\theta_i-\theta|})\sqrt{\rho}||h^2R(\mathbf{u}_h,\theta_h)|| \tag{123}$$

$$\sqrt{a(\mathbf{e}_\Theta,\mathbf{e}_\Theta)} \leq \frac{c\sqrt{\rho}}{1-\delta}(C_c+C_{e,0}\max_{\theta_i\notin\Theta}\frac{\theta\theta_i^{1/2}}{|\theta_i-\theta|}h_{max})||hR(\mathbf{u}_h,\theta_h)|| \tag{124}$$

where C_c is related to the coercivity constant

$$||D\mathbf{v}|| \leq C_c\sqrt{a(\mathbf{v},\mathbf{v})} \tag{125}$$

In Ciarlet's book *("The finite element method for elliptic problems")*, the coercivity constant is given

$$a(\mathbf{v},\mathbf{v}) \geq 2\mu||D\mathbf{v}|| \quad \Rightarrow \quad C_c = \frac{c}{\sqrt{2\mu}} \tag{126}$$

1.9.4 Explicit Estimator - Multiple Materials

To date, we have not seen any publication which extends the explicit error estimator to multiple materials. We don't believe that there are signficant issues, and present the approach used in Salinas here. There are two main constraints from the explicit error estimator formulations that must be maintained.

1. The eigenvectors, u_h must be unit normalized, i.e.$\|u_h\| = 1$. This is important for mass scaling so that a change of units does not affect the fractional error in the solution. It is an essential part of both Larsen's development and Ulrich's extension to elasticity. See equation 108.

2. The extensions must maintain finite element consistency so that as h goes to zero there is no inconsistency.

The second of these can be evaluated by examination of the residuals (as in equation 98). Both the internal and the flux terms of the residuals are unaffected by most scaling operations provided that materials remain constant within an element. Note that the evaluation of the flux jump (equation 101) is unaffected by multiple materials since the normal component of stress discontinuity should go to zero even for disparate materials.

Eigenvector normalization could be addressed in several ways. The eigenvectors computed in Salinas are mass normalized, i.e. $u^T M u = I$. We renormalize for error estimation in the following manner.

1. A unitless mass matrix, $\bar{M}$ is computed using unit density material.

2. We compute a scale factor
$$m_\alpha = u^T \bar{M} u \tag{127}$$

3. The eigenvectors are renormalized as $u \leftarrow u/\sqrt{m_\alpha}$.

In addition to eigenvector renormalization, we move the evaluation of the scaling constant, C_e, s, from equation 117 inside the summation of equation 96. This maintains the proper scaling with respect the element stiffness terms.

A recent paper by Bernardi and Verfurth[10] has shown that explicit estimators can be used in the presence of multiple materials. For static Laplace equation, he derived multiplicative constants for the interior and flux contributions that make the multiplicative constant in front of the estimator independent of jumps in material properties. In what follows we extend this approach to the eigenvalue problem, and to elasticity problems. We will follow the same approach as in that paper, i.e. first constructing the lower bound, and then the upper bound. The proper choices for the coefficients will result from the upper and lower bound estimates.

First, we note a commonly used form for explicit estimators.

$$\|\mathbf{u_h} - \mathbf{u}\|_\alpha \leq c \sum_K \left(h \|R_i(\mathbf{u_h}, \theta_h)\|^2_{L^2(K)} + \sqrt{h} \| \frac{[\sigma_n(\mathbf{u_h})]}{2} \|^2_{L^2(\partial K)} \right)^{\frac{1}{2}} \tag{128}$$

where $R_i(\mathbf{u_h}, \theta_h) = |\nabla \cdot \sigma(\mathbf{u}_h) + \theta_h \rho \mathbf{u}_h|$, $[\sigma_n(\mathbf{u_h})]$ is the jump in stress across the element boundary ∂K, and $\| \cdot \|_\alpha$ is the energy norm. This estimator can be shown to give both an upper and a lower bound on the error. As written, this estimator does not fully account for discontinuous material properties, since the constant c in front of the estimator would depend on the jumps in material properties.

We note that the estimator, written in this form, is essentially the same as the one proposed by Larson. For example, by writing the boundary term as an integral of a constant function, scaled by the volume of the element, then we can write equation 128 in the form

$$\|u_h - u\|_\alpha \leq c \sum_K \left(\| h R_i(u_h, \theta_h) + \frac{\sqrt{h}}{Vol(K)} \frac{[\sigma_n(\mathbf{u_h})]}{2} \|^2_{L^2(K)} \right)^{\frac{1}{2}} \tag{129}$$

which is the same expression given by Larson in the case of linear elements. We note that this estimator is in terms of the energy norm, whereas Larson gives his results in terms of the L^2 norm. This results in the difference of one power of h in equation 129.

The approach in Bernardi is to replace the estimator in equation 128 by

$$\|\mathbf{u_h} - \mathbf{u}\|_\alpha \leq c \sum_K \left({\mu_K}^2 \|R_i(\mathbf{u_h}, \theta_h)\|^2_{L^2(K)} + \mu_e \| \frac{[\sigma_n(\mathbf{u_h})]}{2} \|^2_{L^2(\partial K)} \right)^{\frac{1}{2}} \tag{130}$$

where μ_K and μ_e are chosen in such a way that the resulting estimator is both an upper and lower bound on the error, and the constant c is independent of the jumps in material properties.

Before beginning, we redefine the original PDE as follows

$$\frac{-\nabla \cdot \sigma}{\rho} = \theta \mathbf{u} \tag{131}$$

the corresponding bilinear forms as

$$a(\mathbf{u}, \mathbf{v}) = \int_\Omega \frac{1}{\rho} \sigma(\mathbf{u}) \cdot \epsilon(\mathbf{v}) d\mathbf{x}$$
$$b(\mathbf{u}, \mathbf{v}) = \int_\Omega \mathbf{u} \cdot \mathbf{v} d\mathbf{x}$$

and the corresponding interior residual as

$$R_i(\mathbf{u_h}, \theta_h) = |\frac{\nabla \cdot \sigma(\mathbf{u}_h)}{\rho} + \theta_h \mathbf{u}_h| \tag{132}$$

By dividing through by ρ, we include the density in the energy norm. This will be important later on when the coefficients in equation 130 are selected.

As in Bernardi, we need the following identities, which follow from equation 92

$$a(\mathbf{u} - \mathbf{u_h}, \mathbf{v}) \quad = \quad \theta b(\mathbf{u}, \mathbf{v}) - a(\mathbf{u_h}, \mathbf{v}) \tag{133}$$

$$\theta b(\mathbf{u}, \mathbf{v}) - a(\mathbf{u_h}, \mathbf{v}) = \sum_K \int_K \left(\theta \mathbf{u} + \frac{1}{\rho} \nabla \cdot \sigma(\mathbf{u}_h) \right) \mathbf{v} d\mathbf{x} - \sum_e \int_e \left[\frac{1}{\rho} \sigma_n(\mathbf{u_h}) \right] \cdot \mathbf{v} d\tau \tag{134}$$

where the summation $\sum_e$ is over all edges (in 2D) or over all faces (in 3D). We also use equations 2.11 in Bernardi's paper.

The lower bound will be considered first. We set $w_K = \Psi_K R_i(\mathbf{u_h}, \theta_h)$, where Ψ_K comes from equation 2.11 in Bernardi's paper. We will also make use of the following inequality for the bilinear form

$$a(\mathbf{u}, \mathbf{v})_K \quad \leq \quad \|\mathbf{u}\|_\alpha \|\mathbf{v}\|_\alpha \tag{135}$$
$$\leq \quad \alpha_K \|\mathbf{u}\|_1 \|\mathbf{v}\|_1 \tag{136}$$

where $\alpha_K = \frac{C_K}{\rho_K}$, and C_K is the maximum eigenvalue of the material property matrix, and ρ_K is the density of the element.

For the interior part of the residual, we have

$$\begin{aligned}
\|R_i(u_h,\theta_h)\|^2_{L^2(K)} &\leq \gamma_1^2 \int_K \left[\frac{1}{\rho}\nabla\cdot\sigma(\mathbf{u_h}) + \theta_h \mathbf{u_h}\right]\cdot \mathbf{w_K} d\mathbf{x} \\
&= -\gamma_1^2 \int_K \frac{1}{\rho}\sigma(\mathbf{u_h})\cdot\epsilon(\mathbf{w_K}) d\mathbf{x} + \gamma_1^2 \int_K \theta_h \mathbf{u_h}\cdot\mathbf{w_K} \\
&= \gamma_1^2 a(\mathbf{u}-\mathbf{u_h},\mathbf{w_K})_K - \gamma_1^2\theta\int_K \mathbf{u}\cdot\mathbf{w_K} d\mathbf{x} + \gamma_1^2\theta_h\int_K \mathbf{u_h}\cdot\mathbf{w_K} d\mathbf{x} \\
&\leq \gamma_1^2\left[\|\mathbf{u}-\mathbf{u_h}\|_{\alpha(K)}\gamma_2 h_K^{-1}\alpha_K^{\frac{1}{2}} + \|\theta_h\mathbf{u_h}-\mathbf{u}\theta\|_{L^2(K)}\right] \\
&\times \|R_i(u_h,\theta_h)\|_{L^2(K)} \qquad (137)
\end{aligned}$$

where we note that, since Ψ_K is a bubble function, the boundary terms vanish in the integration by parts on the second line of the above equation.

This implies that

$$\|R_i(u_h,\theta_h)\|_{\alpha(K)} \leq \gamma_1^2\left[\|\mathbf{u}-\mathbf{u_h}\|_{\alpha(K)}\gamma_2 h_K^{-1}\alpha_K^{\frac{1}{2}} + \|\theta_h\mathbf{u_h}-\mathbf{u}\theta\|_{L^2(K)}\right]$$

or, multiplying through by μ_K,

$$\mu_K\|R_i(u_h,\theta_h)\|_{\alpha(K)} \leq \gamma_1^2\left[\|\mathbf{u}-\mathbf{u_h}\|_{\alpha(K)}\mu_K\gamma_2 h_K^{-1}\alpha_K^{\frac{1}{2}} + \mu_K\|\theta_h\mathbf{u_h}-\mathbf{u}\theta\|_{L^2(K)}\right]$$

Now is where a critical assumption comes into play. We assume here that the computed eigenvalue θ_h and eigenvector $\mathbf{u_h}$ are closer to the exact solution θ and $\mathbf{u}$ than any other eigenvalue/eigenvector pair. This assumption is also made by Larson, in equation 2.6. With this assumption, the term $\|\theta_h\mathbf{u_h}-\mathbf{u}\theta\|_{L^2(K)}$ is a higher order term compared with $\|\mathbf{u}-\mathbf{u_h}\|_{\alpha(K)}$, and thus will decay to zero at a faster rate. This was also shown in the paper by Duran.[11] Thus, we select μ_K based on the term $\|\mathbf{u}-\mathbf{u_h}\|_{L^2(K)}$ only. If we select $\mu_K = h_K\alpha_K^{-\frac{1}{2}}$ then the right hand side is independent of the jumps in material properties.

For the boundary term, we first choose $w_e = \Psi_e\left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right]$, where again Ψ_e comes from equation 2.11 in Bernardi. Then, using equation 137 we have

$$
\begin{aligned}
\| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|^2_{L^2(e)} &\leq \gamma_3^2 \int_e \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \cdot \mathbf{w_e} d\tau \\
&= \gamma_3^2 \sum_K \int_K \left(\nabla \cdot \frac{1}{\rho}\sigma(\mathbf{u}_h) + \theta_h \mathbf{u}_h\right) \cdot \mathbf{w_e} - \gamma_3^2 \sum_K a(\mathbf{u}-\mathbf{u_h}, \mathbf{w_e}) \\
&+ \gamma_3^2 \sum_K \int_K (\theta \mathbf{u} - \theta_h \mathbf{u_h}) \cdot \mathbf{w_e} \\
&\leq c\gamma_3^2 \left(\sum_K \gamma_5 h_e^{\frac{1}{2}} \|R_i(u_h, \theta_h)\|_{L^2(K)} + \sum_K \gamma_4 h_e^{-\frac{1}{2}} \alpha_K^{\frac{1}{2}} \|\mathbf{u}-\mathbf{u_h}\|_\alpha \right. \\
&+ \left. \gamma_5 h_e^{\frac{1}{2}} \sum_K \|\mathbf{u}\theta - \mathbf{u_h}\theta_h\|_{L^2(K)} \right) \| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|_{L^2(e)} \\
&\leq c\gamma_3^2 \left[\sum_K h_e^{-\frac{1}{2}} \alpha_K^{\frac{1}{2}} \|\mathbf{u}-\mathbf{u_h}\|_\alpha + \sum_K h_e^{\frac{1}{2}} \|\theta_h \mathbf{u_h} - \theta \mathbf{u}\|_{L^2(K)} \right] \\
&\times \| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|_{L^2(e)} \qquad (138)
\end{aligned}
$$

where in the above equation, $\sum_K$ denotes a summation over elements, but only those elements that border the edge e. Also, in the previous estimate we collected constants involving γ and combine with the constant c, where possible.

This implies that

$$
\mu_e^{\frac{1}{2}} \| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|_{L^2(e)} \leq c\gamma_3^2 \mu_e^{\frac{1}{2}} \left[\sum_K h_e^{-\frac{1}{2}} \alpha_K^{\frac{1}{2}} \|\mathbf{u}-\mathbf{u_h}\|_\alpha + \sum_K h_e^{\frac{1}{2}} \|\theta_h \mathbf{u_h} - \theta \mathbf{u}\|_{L^2(K)} \right]
$$

We see that if we choose $\mu_e = h_e \max(\alpha_{K1}, \alpha_{K2})^{-1}$, where subscripts 1 and 2 denotes the two neighboring elements that contain the edge or face e, then the right hand side (neglecting the higher order term) is independent of the jumps in material properties.

Now we construct the upper bound. We start with a few identities that will be needed along the way.

$$
\begin{aligned}
\int_\Omega \left(\frac{1}{\rho}\nabla \cdot \sigma(\mathbf{u}_h) + \theta \mathbf{u}\right) \cdot (\mathbf{w}-\mathbf{w_h}) = -a(\mathbf{u_h}, \mathbf{w}-\mathbf{w_h}) + \\
\sum_e \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \cdot (\mathbf{w}-\mathbf{w_h}) + \int_\Omega \theta \mathbf{u}(\mathbf{w}-\mathbf{w_h})
\end{aligned} \qquad (139)
$$

This implies that

$$a(\mathbf{u_h}, \mathbf{w} - \mathbf{w_h}) = \sum_e \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \cdot (\mathbf{w} - \mathbf{w_h})$$
$$+ \int_\Omega \theta \mathbf{u} \cdot (\mathbf{w} - \mathbf{w_h}) - \int_\Omega \left(\frac{1}{\rho}\nabla \cdot \sigma(\mathbf{u}_h) + \theta\rho\mathbf{u}\right) \cdot (\mathbf{w} - \mathbf{w_h}) \tag{140}$$

We will use the previous result in the upper bound on the energy norm of the error. Let $\mathbf{w} = \mathbf{u} - \mathbf{u_h}$. Then

$$\|\mathbf{u} - \mathbf{u_h}\|_\alpha^2 = a(\mathbf{u} - \mathbf{u_h}, \mathbf{w}) = a(\mathbf{u} - \mathbf{u_h}, \mathbf{w} - \mathbf{w_h}) \tag{141}$$

where the last equality follows from Galerkin orthogonality. Breaking the previous expression into element-wise quantities, and using equation 140, we obtain

$$\begin{aligned}
\|\mathbf{u} - \mathbf{u_h}\|_\alpha^2 &= \sum_K a(\mathbf{u} - \mathbf{u_h}, \mathbf{w} - \mathbf{w_h}) \qquad (142)\\
&= \sum_K a(\mathbf{u}, \mathbf{w} - \mathbf{w_h}) - \sum_e \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \cdot (\mathbf{w} - \mathbf{w_h})\\
&- \sum_K \int_K \theta\mathbf{u} \cdot (\mathbf{w} - \mathbf{w_h}) + \sum_K \int_K \left(\nabla \cdot \frac{1}{\rho}\sigma(\mathbf{u}_h) + \theta\mathbf{u}\right) \cdot (\mathbf{w} - \mathbf{w_h})\\
&= \sum_K \int_K \left(\nabla \cdot \frac{1}{\rho}\sigma(\mathbf{u}_h) + \theta\mathbf{u}\right) \cdot \mathbf{w} - \mathbf{w_h} - \sum_e \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \cdot (\mathbf{w} - \mathbf{w_h})\\
&\leq \sum_K \mu_K \|\nabla \cdot \frac{1}{\rho}\sigma(\mathbf{u}_h) + \theta\mathbf{u}\|_{L^2(K)} \mu_K^{-1} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(K)}\\
&+ \sum_e \mu_e^{\frac{1}{2}} \| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|_{L^2(e)} \mu_e^{\frac{1}{2}} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(e)}\\
&\leq \left[\sum_K \mu_K^2 \|\nabla \cdot \frac{1}{\rho}\sigma(\mathbf{u}_h) + \theta\mathbf{u}\|_{L^2(K)}^2 + \sum_e \mu_e \| \left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right] \|_{L^2(e)}^2\right]^{\frac{1}{2}}\\
&\times \left[\sum_K \mu_K^{-2} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(K)}^2 + \sum_e \mu_e^{-1} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(e)}^2\right]^{\frac{1}{2}}
\end{aligned}$$

We now use equation 2.16 in Bernardi's paper, which shows that

$$\left[\sum_K \mu_K^{-2} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(K)}^2 + \sum_e \mu_e^{-1} \|\mathbf{w} - \mathbf{w_h}\|_{L^2(e)}^2\right]^{\frac{1}{2}} \leq c\|\mathbf{w}\|_\alpha \tag{143}$$

With this result, we have

$$\|\mathbf{u}-\mathbf{u_h}\|_\alpha \leq c\left[\sum_K \mu_K^2 \|\nabla\cdot\frac{1}{\rho}\sigma(\mathbf{u}_h)+\theta\rho\mathbf{u}\|^2_{L^2(K)}+\sum_e \mu_e\|\left[\frac{1}{\rho}\sigma_n(\mathbf{u_h})\right]\|^2_{L^2(e)}\right]^{\frac{1}{2}} \quad (144)$$

which is the desired upper bound. We note that we would also obtain higher order terms in the above expression by adding and subtracting terms of the kind $\int_K \theta_h \mathbf{u_h} dx$, but the same argument could be made as before.

1.9.5 Explicit Estimator Summary

Summarizing, the implementation of the explicit error estimator involves the following steps. These steps have to be carried out for each eigenvalue separately.

1. Renormalize the eigenvectors as in section 1.9.4, equation 127.

2. Loop through all elements in the mesh. Compute the surface flux residuals for each face. Share that residual vector at each surface gauss point with neighboring elements to determine the stress jump 103. Integrate over all faces (by summing at surface gauss points) to determine R_{flux} (eq 101).

3. Loop through all elements in the mesh. At each interior gauss point of each element,

   (a) Compute the interior residual,

   $$a_1 = |A_1(u_h)+\lambda_h A_2(u_h)|$$

   (b) Compute the integrand,

   $$(a_1+R_{flux})^2$$

   Note that R_{flux} is a constant over the element.

   (c) Sum at gauss points to obtain the element contribution,

   $$\begin{aligned}\bar{\rho}^2 &= \int_{\Omega_e}(a_1+R_{flux})^2 d\Omega_e \\ &\approx \sum_i^{N_{gauss}} w_i(a_1(x_i)+R_{flux})^2\end{aligned}$$

4. Compute the global contribution to the error. For elements with linear shape functions, this may be written,

$$\frac{|\lambda - \lambda_h|}{\lambda} \leq c \left(\sum_{e=1}^{N_e} (C_{e,0} h_e^2 \bar{\rho})^2 \right)^{\frac{1}{2}}. \tag{145}$$

Where (as shown in section 1.9.3, equation 117),

$$C_{e,0}^2 = \frac{\rho}{(\Lambda + \mu)^2}$$

and ρ, Λ and μ are the material density and the Lamé constants respectively. The more general expression for elements of order p is,

$$\frac{|\lambda - \lambda_h|}{\lambda^{(p+1)/2}} \leq c \left(\sum_{e=1}^{N_e} (C_{e,p-1} h_e^{(p+1)} \bar{\rho})^2 \right)^{\frac{1}{2}}. \tag{146}$$

We note that although the constant, c, in equation 145 is not known completely, it is usually estimated to be of order 1. The constant depends on the details of the mesh, and in particular on the minimum angle in the elements.

1.9.6 Approach II - quantity of interest estimator

In,[8] an error estimator is derived for the elasticity equation, using the eigenvalues as the quantity of interest. The estimate is of the form

$$\eta_{low}^{\lambda} = -\eta_{upp}^2 \tag{147}$$
$$\eta_{upp}^{\lambda} = -\eta_{low}^2 \tag{148}$$

where η_{low}^{λ} is a lower bound on $\lambda - \lambda_h$, and η_{upp}^{λ} is an upper bound on $\lambda - \lambda_h$. Note that both quantities are necessarily negative,[5] since the computed eigenvalues are always larger than the exact ones.

The quantities η_{upp} and η_{low} are computed using the so-called *element residual method*. This method involves solving a small linear system on each element to obtain an error representation for that element, and then the element contributions are accumulated to obtain the total errors. The element residual method involves solving the following linear system on each element

$$-B(\Phi_K, v) = R(v, 0) + \int_{\partial K} g_{\gamma,K} v ds \quad \forall v \in W_K \tag{149}$$

[5] for consistent mass only.

or

$$K_b a = f \tag{150}$$

where a is the vector of coefficients that represent the function Φ_K. In other words, $\Phi_K = \sum_{i=1}^{Nshape_{bubble}} a_i N_i$, where N_i is the i^{th} bubble shape function. The left hand side K_b is the element stiffness matrix, but evaluated using bubble functions rather than the standard element shape functions. This is necessary since the standard element stiffness matrix is singular and thus equation 150 would otherwise not be solvable. The right hand side consists of two terms, an interior residual term for the interior of the element, and a stress jump term on the element boundary. This is similar to the interior and boundary residual terms that were encountered in the explicit error estimator, though the exact formulas for these terms are somewhat different. The first term is simply

$$R(v, 0) = B(u_h, v) - \lambda_h M(u_h, v) \tag{151}$$

Equation 151 can be most efficiently evaluated using the following method.[12] We evaluate the first term first.

$$B(u_h, v) = \int_K B_{bubble}^T \sigma(x) dx \tag{152}$$

where B_{bubble}^T is the standard 'B' matrix, or the matrix of derivatives of the element shape functions, except that it is using the bubble shape functions rather than the standard shape functions. Note that the result of equation 152 is a vector of length $3xNshape_{bubble}$, where $Nshape_{bubble}$ is the number of bubble shape functions. We note that the routine ForceFromStress in IsoSolid.C already performs the computation needed for equation 152, with the only change being the use of the matrix B_{bubble}^T rather than the standard B^T, and thus this code could be re-used.

The second term can be evaluated in a similar way.

$$M(u_h, v) = \int_K u_h(x) v(x) dx \tag{153}$$

Note that $u_h(x)$ is a known function. This term is also a vector of length $3xNshape_{bubble}$. The three entries corresponding to the i^{th} bubble shape function are as follows

$$\int_K u_{1h}(x) \phi_i(x) dx \tag{154}$$

$$\int_K u_{2h}(x) \phi_i(x) dx \tag{155}$$

$$\int_K u_{3h}(x) \phi_i(x) dx \tag{156}$$

$$\tag{157}$$

where u_{1h}, u_{2h}, and u_{3h} are the x, y, and z components of u_h, and ϕ_i is the i^{th} bubble shape function.

The boundary term consists of the following. $g_{\gamma,K}$ is simply the traction on the element boundary, or

$$\int_{\partial K} g_{\gamma,K} v ds \quad = \quad \int_{\partial K} [\sigma_{ij} n_j]\, v ds \tag{158}$$

where $[\sigma_{ij} n_j]$ denotes the *averaged* stress on the element faces. For two adjacent elements, element 'a' and element 'b', it is the average of their stress traction vectors.

$$[\sigma_{ij} n_j] = \frac{1}{2}\left(\sigma^a_{ij} n_j + \sigma^b_{ij} n_j\right) \tag{159}$$

Again, the test (shape) function in this case, 'v' is the bubble function rather than the standard element shape function. We note that the boundary integral term in equation 149 and equation 158 is over all faces of the element in question. Thus, if the implementation of this term proceeds one face at a time, then there will be a nodal summation step to get the complete right hand side vector corresponding to the boundary integral term. We could also write this term as

$$\int_{\partial K} g_{\gamma,K} v ds = \sum_{i=1}^{N_{faces}} \int_{\partial K_i} g_{\gamma,K} v ds \tag{160}$$

where ∂K_i is the i^{th} face of element 'K'. Note that the test functions, v become the element shape functions when restricted to an element. Thus, for a given element bubble shape function ϕ_{bubble}, and a given face, we can write the previous equation as

$$\int_{\partial K_i} g_{\gamma,K} \phi_{bubble} ds \tag{161}$$

Note that $g_{\gamma,K}$ is a 3-vector, and so for a given bubble shape function, and a given face, $\int_{\partial K_i} g_{\gamma,K} \phi_{bubble} ds$ is also a 3-vector. We then take this 3-vector and project it into the element right hand side. After looping through all faces and all bubble shape functions, we end up with a vector that is of length $3 * Nshape_{bubble}$.

Once the linear systems 150 are solved on each element, the upper bound, η_{up} from equation 148 can be computed as follows

$$\eta_{upp} = \sqrt{\sum_K B(\Phi_K, \Phi_K)} \tag{162}$$

This equation can also be written as follows. If we represent the function Φ_K as a summation of coefficients multiplied by the bubble shape functions,

$$\Phi_K = \sum_{i=1}^{Nshape_{bubble}} a_i N_i \tag{163}$$

then

$$\eta_{upp} = \sqrt{\sum_K B(\Phi_K, \Phi_K)} = \sqrt{\sum_K a^T K_b a} \tag{164}$$

Finally, using equation 148, we have an upper bound on the error in the eigenvalue.

A lower bound on the error in the eigenvalue can also be computed. This is described in detail in,[8] and we summarize here.

First, we define a function $\chi \in V$, which we will define shortly. Once the function χ is defined, the lower bound can be computed as follows

$$\eta_{low} = \frac{|R_p(\chi, 0)|}{\sqrt{B(\chi, \chi)}} \tag{165}$$

The quantities in both the numerator and denominator can be computed by looping through all elements and computing the corresponding element-wise quantities (using equation 151), and then summing globally.

Summarizing, in order to implement the quantity of interest approach for eigenvalue error estimation, we have the following steps. These must be carried out for each eigenvalue.

1. Loop over all elements. Construct the bubble stiffness matrix, K_b in equation 150, in the same way that standard element stiffness matrix is constructed, but using the bubble shape functions.

2. Loop over all elements. Construct the right hand side of equation 150. This consists of the interior part, equation 151, and the boundary part, equation 158.

3. Loop over all elements and solve the linear systems 150, to obtain the error functions Φ_K.

4. Compute the upper bound on the error in the eigenvalue using equation 164.

5. Compute the lower bound on the error in the eigenvalue using equation 165.

2 Elements

Structural dynamics is a rich and extensive field. Finite element tools such as **Salinas** have been used for decades to describe and analyze a variety of structures. The same tools are applied to large civil structures (such as bridges and towers), to machines, and to micron sized structures. This has necessarily led to a wealth of different element libraries. Details of these element libraries are presented in this section. For information on the solution procedures that tie these elements together, please refer to section 1.

2.1 Isoparametric Solid Elements. Selective Integration

The following applies to any solid isoparametric element, but is implemented in the code on elements with linear shape functions (such as hex8 or wedge6). This discussion addresses calculation of relevant operators on the shape functions and eventual integration into the stiffness matrices. [6]

2.1.1 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 avoiding overstiffness with respect to bending.

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

$$p = \frac{1}{2}(2G\epsilon + \lambda tr(\epsilon)I) \bullet \epsilon \tag{166}$$

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

$$p = G\hat{\epsilon} \bullet \hat{\epsilon} + \frac{1}{2}\beta(tr(\epsilon))^2 \tag{167}$$

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.

[6] This development is based on work by Dan Segalman.

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:

$$\vec{u}(x) = \tilde{N}^{P_j}(x) u_{s_j}^{P_j} \vec{e}_{s_j} \tag{168}$$

where summation takes place over the degree of freedom j.

Similarly, the displacment gradient is:

$$\vec{\nabla}\vec{u}(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x) u_{s_j}^{P_j} \vec{e}_{s_j}\vec{e}_k \tag{169}$$

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

$$W^j(x) = (\frac{\partial}{\partial u_{s_j}^{P_j}})\vec{\nabla}\vec{u}(x) \tag{170}$$

which, with Equation 169 yields:

$$W^j(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x)\vec{e}_{s_j}\vec{e}_k \tag{171}$$

The symmetric part of this tensor is:

$$S^j(x) = \frac{1}{2}(W^j(x) + W^j(x)^T) \tag{172}$$

and the strain tensor is

$$\epsilon(x) = S^j(x) u_{s_j}^{P_j} \tag{173}$$

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

$$tr(\epsilon(x)) = b^j(x) u_{s_j}^{P_j} \tag{174}$$

where

$$b^j(x) = tr(S^j(x)) \tag{175}$$

Similarly,

$$\hat{\epsilon}(x) = \hat{B}^j(x) u_{s_j}^{P_j} \tag{176}$$

where

$$\hat{B}^j(x) = S^j(x) - \frac{1}{3} b^j(x) I \tag{177}$$

The stiffness matrix is evaluated using the consitutive equation (Equation 167) and the following definition:

$$K_{m,n} = \frac{\partial^2}{\partial u_{s_m}^{P_m} \partial u_{s_n}^{P_n}} \int_{volume} p(x) dV(x) \tag{178}$$

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

$$K_{m,n} = G \int_{volume} (\hat{B}^m(x))^T \bullet \hat{B}^n(x) dV(x) + \beta \int_{volume} b^m(x) b^n(x) dV(x) \tag{179}$$

2.2 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

$$W^j_{mn} = \vec{e}_m \cdot W^j \cdot \vec{e}_n \tag{180}$$

$$= \delta_{m,s_j} (\frac{\partial}{\partial x_n}) \tilde{N}^{P_j}(x) \tag{181}$$

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

$$(\frac{\partial}{\partial x_n})\tilde{N}^{P_j}(x(\xi)) = (\frac{\partial}{\partial \xi_\alpha}) N^{P_j}(\xi) J^{-1}_{\alpha,n} \tag{182}$$

where

$$J_{m,\gamma} = \frac{\partial}{\partial \xi_\gamma} x_m(\xi) \tag{183}$$

and

$$N(\xi) = \tilde{N}(x(\xi)) \tag{184}$$

The issue of selective integration in the elements is discussed in Appendix B. The formulation discussed there applies to all the isoparametric solid elements.

2.3 Quadratic Isoparametric Solid Elements

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 section 2.1 and Appendix B for definitions of G and β.) Therefore, G=0 and β=0 are good values for such elements.

2.4 Wedge elements

2.4.1 Shape Functions

The shape functions are given explictly in *Hughes*, (ref. 13). These are provided as bi-linear polynomials in r, s, t, and ξ, where r and s are independent coordinates of the triangular cross-subsections, $t = 1 - r - s$, 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:

$$N_k = A_0^k + A_1^k r + A_2^k s + A_3^k \xi + A_4^k r\xi + A_5^k s\xi \tag{185}$$

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
$N_1 = \frac{1}{2}(1-\xi)r$		$\frac{1}{2}$			-$\frac{1}{2}$	
$N_2 = \frac{1}{2}(1-\xi)s$			$\frac{1}{2}$			$-\frac{1}{2}$
$N_3 = \frac{1}{2}(1-\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$N_4 = \frac{1}{2}(1+\xi)r$		$\frac{1}{2}$			$\frac{1}{2}$	
$N_5 = \frac{1}{2}(1+\xi)s$			$\frac{1}{2}$			$\frac{1}{2}$
$N_6 = \frac{1}{2}(1+\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$

2.4.2 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/$\sqrt{3}$

2.5 Tet10 elements

The 4-point integration is given in *Hughes* (see 14), 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.)

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

Using a 3D Pascal's triangle, we can construct 20 polynomials of the form,

$$p_i = \varepsilon_1^{r_i} \varepsilon_2^{s_i} \varepsilon_3^{t_i}$$

where the r_i, s_i and t_i $(i = 1, \ldots, 20)$ are integers satisfying,

$$r_i^2 + s_i^2 + t_i^2 \leq 7$$

These terms may be constructed with the following loop.[7]

```
count=0
for I = 0 to 7
  for J = 0 to 7
    for K = 0 to 7
      if I^2 + J^2 + K^2 <= 7
        count = count + 1
          r(count) = I
          s(count) = J
          t(count) = K
      endif
    endfor
  endfor
endfor
```

We require 20 shape functions N_i, with $i = 1, \ldots, 20$, that satisfy the conditions that $N_i = 1$ at node i and $N_i = 0$ at every other node. This results in 20 equations at each node. Expressing the N_i as linear combinations of the p_i, we can write,

$$\vec{N} = A\vec{p} \tag{186}$$

where A is a 20x20 matrix. We want to find the 400 term $A-$matrix values. For each node, we have 20 equations and there are 20 nodes; so, there are 400 equations for the 400 unknowns. Let $\vec{\varepsilon_i}$ denote the natural coordinate value at the ith node. We have $A\vec{p}(\vec{\varepsilon_1}) = \vec{e}_1 \equiv (1, 0, 0, \ldots, 0)^T$, and, in general, $A\vec{p}(\vec{\varepsilon_i}) = \vec{e}_i$. So,

$$[\vec{\varepsilon}_1, \vec{\varepsilon}_2, \ldots, \vec{\varepsilon}_{20}] = [A][\vec{p}(\vec{\varepsilon}_1), \vec{p}(\vec{\varepsilon}_2), \ldots, \vec{p}(\vec{\varepsilon}_{20})]$$

[7] This is how the `rst` matrix in `Hex20.C` was created.

or,

$$I = AP$$

or,

$$A = P^{-1}$$

This matrix A is the matrix *"hc20"* in `Hex20.C`.

Not only can the shape functions be expressed as a linear combination of the p_i, but so can the derivatives, $\frac{\partial \vec{N}}{\partial \varepsilon_j}$, $(j = 1, 2, 3)$. Differentiating equation 186, we have

$$\frac{\partial \vec{N}}{\partial \varepsilon_j} = A \frac{\partial \vec{p}}{\partial \varepsilon_j}$$

but the $\partial \vec{p}/\partial \varepsilon_j$ may be written as a linear combination of the p_k via the following trhee equations.

$$\frac{\partial p_i}{\partial \varepsilon_1} = r_i \varepsilon_1^{r_i - 1} \varepsilon_2^{s_i} \varepsilon_3^{t_i} \tag{187}$$

$$\frac{\partial p_i}{\partial \varepsilon_2} = s_i \varepsilon_1^{r_i} \varepsilon_2^{s_i - 1} \varepsilon_3^{t_i} \tag{188}$$

$$\frac{\partial p_i}{\partial \varepsilon_3} = t_i \varepsilon_1^{r_i} \varepsilon_2^{s_i} \varepsilon_3^{t_i - 1} \tag{189}$$

while noting that equations 187, 188 and 189 are zero if r_i, s_i, or t_i is zero, respectively. We would like to find the matrix B_j with $j = 1, 2, 3$ such that,

$$\frac{\partial \vec{N}}{\partial \varepsilon_j} = B_j \vec{p}.$$

Evaluating $\partial \vec{N}/\partial \varepsilon_j$ and $\vec{p}$ at all 20 nodes, we have,

$$\left[\frac{\partial \vec{N}}{\partial \varepsilon_j}(\vec{\varepsilon}_1), \frac{\partial \vec{N}}{\partial \varepsilon_j}(\vec{\varepsilon}_2), \ldots, \frac{\partial \vec{N}}{\partial \varepsilon_j}(\vec{\varepsilon}_{20}) \right] = B_j \left[\vec{p}(\vec{\varepsilon}_1), \vec{p}(\vec{\varepsilon}_2), \ldots, \vec{p}(\vec{\varepsilon}_{20}) \right] \tag{190}$$

Matrix equation 190 can be inverted to solve for B_j with $j = 1, 2, 3$. In `Hex20.C`, `AB1` is B_1 , `AB2` is B_2, and `AB3` is B_3.

2.7 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.

2.8 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.

2.8.1 Allman's Triangular Element

Using the formulation given in Ref. 15 and replacing $\cos(\gamma_{ij}) = \frac{y_{ji}}{l_{ij}}$ and $\sin(\gamma_{ij}) = \frac{-x_{ji}}{l_{ij}}$, we get

$$u = u_1\psi_1 + u_2\psi_2 + u_3\psi_3 + \frac{1}{2}y_{21}(\omega_2 - \omega_1)\psi_1\psi_2 + \frac{1}{2}y_{32}(\omega_3 - \omega_2)\psi_2\psi_3 + \frac{1}{2}y_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{191}$$

$$v = v_1\psi_1 + v_2\psi_2 + v_3\psi_3 + \frac{1}{2}x_{21}(\omega_2 - \omega_1)\psi_1\psi_2 - \frac{1}{2}x_{32}(\omega_3 - \omega_2)\psi_2\psi_3 - \frac{1}{2}x_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{192}$$

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.

2.8.2 Discrete Kirchoff Element

As for the DKT[16] 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. 17, as

$$w = \alpha_1\psi_1+\alpha_2\psi_2+\alpha_3\psi_3+\alpha_4\psi_1\psi_2+\alpha_5\psi_2\psi_3+\alpha_6\psi_3\psi_1+\alpha_7\psi_1{}^2\psi_2+\alpha_8\psi_2{}^2\psi_3+\alpha_9\psi_3{}^2\psi_1 \tag{193}$$

For the AT and DKT elements, the stiffness and mass matrices are derived with the help of Maple. The consistent 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.

2.8.3 Verification and Validation

The AT element is verified by comparing calculated results with the results published by Allman in Ref. 15. The square plate in pure bending and a cantilvered beam with a parabolic tip load are used as verification examples. 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. 16 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. 18. Tables 1 and 2 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.

2.9 Triangular Shell - Tria3

The triangular shell used most in Salinas is the **Tria3** element developed by Carlos Felippa of the University of Colorado in Boulder. This element is very similar to the **TriaShell** element presented in section 2.8. Full details of the theory behind the element is out of the scope of this document, but details may be found in references 19, 20 and 21.

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 1: 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 2: Comparison of deflections at Node 3

2.10 Two Node Beam

This is the definition for a Beam element based on Cook's development (see pp 113-115 of reference 2).

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 subsectional 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).

2.11 Truss

This is the definition for a Truss element based on pages 214-216 of Cook (ref 2).

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 subsectional area of the truss (Attribute 1)

2.12 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. For an understanding of the various solution spaces (such as the A-set), see section 3.1.

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 subsection 2.17.

2.13 Multi-Point Constraints, MPCs

A description of *MPC*s is contained in the users manual. This subsection 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. These include a constraint transform method and Lagrange multipliers. Lagrange multipliers have not been implemented at this time.

2.13.1 Constraint Transforms

Constraints may be eliminated using the constraint transform method. This is described in detail in Cook, chapter 9 (ref 2). 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.)

2.14 Rigid Elements

Salinas supports 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).

2.14.1 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.

2.14.2 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.

2.14.3 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 multi-point 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.

2.15 Shell Offset

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,

$$\begin{pmatrix} \tilde{u} \\ \kappa \end{pmatrix} = [L] \begin{pmatrix} u \\ \kappa \end{pmatrix} \tag{194}$$

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,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} \tilde{u} \\ \kappa \end{array} \right\}^T \left[\tilde{K}\right] \left\{ \begin{array}{c} \tilde{u} \\ \kappa \end{array} \right\} \tag{195}$$

But with this substitution,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\}^T \left[L^T \tilde{K} L\right] \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\} \tag{196}$$

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

$$E_{strain} = 0.5 \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\}^T [K] \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\} \tag{197}$$

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

2.16 Notes on Consistent Loads Calculations

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

$$\{r_e\} = \int_{V_e} [B]^T [E] \{\epsilon_0\} dV - \int_{V_e} [B]^T \{\sigma_0\} dV + \int_{V_e} [N]^T \{F\} dV + \int_{S_e} [N]^T \{\Phi\} dS \tag{198}$$

where each of these terms are defined in Subsection 4.1 of the above mentioned reference. The load vector, $\{r_e\}$, is composed of four parts in Eqn. 198. In this document, only the last part, which is the contribution of the surface tractions to the load vector, will be considered. Rewritting,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS \tag{199}$$

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

$$\{\Phi\} = [\Phi_x \Phi_y \Phi_z]^T \tag{200}$$

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

$$\{\Phi\} = \left\{ \begin{array}{c} n_x \\ n_y \\ n_z \end{array} \right\} \Phi(x, y, z) \tag{201}$$

2.16.1 Salinas Element Types

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

- Hex8
- Hex20
- Wedge6
- Tet4
- Tet10
- Tria3
- TriaShell

- Tria6 (four Tria3s)
- QuadT (two Tria3s)
- Quad8T (1 QuadT and 4 Tria3s)

2.16.2 Pressure Loading

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

$$\{\Phi\} = \left\{ \begin{array}{c} n_x \\ n_y \\ n_z \end{array} \right\} \Phi(x, y, z) \tag{202}$$

where $[n_x, n_y, n_z]^T$ is the normal to the element face. Hence, the consistent loads can be calculated as,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS = \int_{S_e} [N]^T \Phi(x, y, z)(\vec{a} \times \vec{b}) dS_e \tag{203}$$

Here,

$$\vec{a} = [\frac{\partial x}{\partial r}, \frac{\partial y}{\partial r}, \frac{\partial z}{\partial r}]^T \tag{204}$$

$$\vec{b} = [\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s}, \frac{\partial z}{\partial s}]^T \tag{205}$$

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.

2.16.3 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.

2.16.4 Shell Elements - consistent loads

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

$$f^1{}_s = \frac{1}{20}(7q_{s1} + 3q_{s2})L_{21} \qquad f^2{}_s = \frac{1}{20}(3q_{s1} + 7q_{s2})L_{21} \tag{206}$$

$$f^1{}_n = \frac{1}{20}(7q_{n1} + 3q_{n2})L_{21} \qquad f^2{}_n = \frac{1}{20}(3q_{n1} + 7q_{n2})L_{21} \tag{207}$$

$$f^1{}_t = \frac{1}{20}(7q_{t1} + 3q_{t2})L_{21} \qquad f^2{}_t = \frac{1}{20}(3q_{t1} + 7q_{t2})L_{21} \tag{208}$$

$$m^1{}_s = m^2{}_s = 0 \tag{209}$$

$$m^1{}_n = -\frac{1}{60}(3q_{t1} + 2q_{t2})L^2{}_{21} \qquad m^2{}_n = \frac{1}{60}(2q_{t1} + 3q_{t2})L^2{}_{21} \tag{210}$$

$$m^1{}_t = -\frac{1}{40}(3q_{n1} + 2q_{n2})L^2{}_{21} \qquad m^2{}_t = \frac{1}{40}(2q_{n1} + 3q_{n2})L^2{}_{21} \tag{211}$$

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,

$$f^1{}_x = f^1{}_y = m^1{}_z = f^2{}_x = f^2{}_y = m^2{}_z = f^3{}_x = f^3{}_y = m^3{}_z = 0 \tag{212}$$

$$f^1{}_z = (\frac{8}{45}p_1 + \frac{7}{90}p_2 + \frac{7}{90}p_3)A \tag{213}$$

$$f^2{}_z = (\frac{7}{90}p_1 + \frac{8}{45}p_2 + \frac{7}{90}p_3)A \tag{214}$$

$$f^3{}_z = (\frac{7}{90}p_1 + \frac{7}{90}p_2 + \frac{8}{45}p_3)A \tag{215}$$

$$m^1{}_x = \frac{A}{360}[7(y_{31} + y_{21})p_1 + (3y_{31} + 5y_{21})p_2 + (5y_{31} + 3y_{21})p_3] \tag{216}$$

$$m^1{}_y = \frac{A}{360}[7(x_{13} + x_{12})p_1 + (3x_{13} + 5x_{12})p_2 + (5x_{13} + 3x_{12})p_3] \tag{217}$$

$$m^2{}_x = \frac{A}{360}[(5y_{12} + 3y_{32})p_1 + 7(y_{12} + y_{32})p_2 + (3y_{12} + 5y_{32})p_3] \tag{218}$$

$$m^2{}_y = \frac{A}{360}[(5x_{21} + 3x_{23})p_1 + 7(x_{21} + x_{23})p_2 + (3x_{21} + 5x_{23})p_3] \tag{219}$$

$$m^3{}_x = \frac{A}{360}[(3y_{23} + 5y_{13})p_1 + (5y_{23} + 3y_{13})p_2 + 7(y_{23} + y_{13})p_3] \tag{220}$$

$$m^3{}_x = \frac{A}{360}[(3x_{32} + 5x_{31})p_1 + (5x_{32} + 3x_{31})p_2 + 7(x_{32} + x_{31})p_3] \tag{221}$$

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.

2.17 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)
3. specification of material property rotations for anisotropic materials.
4. specification of spring directions (see subsection 2.12).

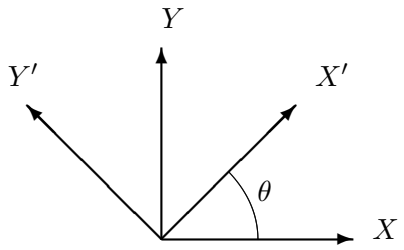

Figure 2: Original, and rotated coordinate frames

5. specification of output coordinate systems (in history files only).

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

1. specification of nodal locations,

2. 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 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:

$$X = R^T X' \tag{222}$$

and

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

As an example, consider a cartesian system as shown in Figure 2.

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,

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

2.18 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,

$$C'u' = Q \tag{223}$$

where the primes indicate a generalized coordinate frame. The frame may be transformed to the basic coordinate system using equation 222, and

$$u' = Ru \tag{224}$$

We can now rewrite equation 223,

$$\begin{aligned} C'Ru &= Q \\ Cu &= Q \end{aligned} \tag{225}$$

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.

2.18.1 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.

$$u = \begin{bmatrix} u_r \\ u_c \end{bmatrix} \tag{226}$$

The whole constraint equation may be similarly partitioned.

$$\begin{bmatrix} C_r & C_c \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = [Q] \tag{227}$$

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,

$$u_c = C_c^{-1} Q - C_c^{-1} C_r u_r \tag{228}$$

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.

$$\begin{bmatrix} u_r \\ u_c \end{bmatrix} = [T]\,[u_r] + \tilde{Q} \tag{229}$$

where

$$T = \begin{bmatrix} 1 \\ C_{rc} \end{bmatrix} \tag{230}$$

$$C_{rc} = -C_c^{-1} C_r \tag{231}$$

and

$$\tilde{Q} = \begin{bmatrix} 0 \\ C_c^{-1} Q \end{bmatrix} = \begin{bmatrix} 0 \\ \breve{Q} \end{bmatrix} \tag{232}$$

2.18.2 Transformation of Stiffness Matrix

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

$$\begin{bmatrix} K_{rr} & K_{rc} \\ K_{cr} & K_{cc} \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = \begin{bmatrix} R_r \\ R_c \end{bmatrix} \tag{233}$$

or,

$$[K]\,[T]\,u_r + [K]\left[\tilde{Q}\right] = R \tag{234}$$

and

$$T^T K T u_r = T^T \left\{ R - K\tilde{Q} \right\} = \tilde{R} \tag{235}$$

We can define the reduced equations,

$$\tilde{K} = T^T K T = K_{rr} + K_{rc} C_{rc} + C_{rc}^T K_{cr} + C_{rc}^T K_{cc} C_{rc} \tag{236}$$

and,

$$\begin{aligned} \tilde{R} &= T^T R - T^T \begin{bmatrix} K_{rc}\breve{Q} \\ K_{cc}\breve{Q} \end{bmatrix} \\ &= R_r + C_{rc}^T R_c - K_{rc}\breve{Q} - C_{rc}^T K_{cc}\breve{Q} \end{aligned} \tag{237}$$

The solution in the retained system is

$$\tilde{K} u_r = \tilde{R} \tag{238}$$

The system may now be solved using the reduced equations, and the constrained degrees of freedom may be solved using equation 228. 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 236. 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.

2.18.3 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 236 and 237 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 223).
2. Transform the constraint equation to the basic coordinate system (equation 224).
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 227 and 231.
5. Compute the corrections to the force vector from equation 237. 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 236.
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 228).

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). 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,

$$u_{\text{spc}} = \text{constant} + \sum_{i}^{\text{retained dofs}} u_i C_i \tag{239}$$

2.18.4 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. Subsection 2.13 deals more explicitly with MPCs.

2.18.5 Transformation of Power Spectral Densities

Note: The following is taken almost verbatim from Paez's book [22]. 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,

$$\tilde{H}_1(f) = \sum_{k=1}^{3} c_{ki} H_k(f) \tag{240}$$

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,

$$G_{ij}(f) = \alpha H_i^*(f) H_j(f) \tag{241}$$

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 240 to express $\tilde{G}$ in terms of the H_i. We may then use the spectral definition in equation 241 to provide the transformation of spectral densities.

$$\begin{aligned} \tilde{G}_{ij}(f) &= \alpha \left(\sum_{k=1}^{3} c_{ki} H_k^*(f) \right) \left(\sum_{m=1}^{3} c_{mj} H_m(f) \right) \\ &= \sum_{k=1}^{3} \sum_{m=1}^{3} c_{ki} c_{mj} G_{km} \end{aligned} \tag{242}$$

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

2.19 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 (see reference 23). 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
```

3 Linear Algebra Issues

3.1 Solution Spaces

There are a 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);
```

References

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

[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] Reese, G., Field, R., and Segalman, D. J., "A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments," *Shock and Vibration. Digest*, **vol. 32**, no. 6, 2000.

[4] Craig, R. R., *Structural Dynamics: An Introduction to Computer Methods*, John Wiley & Sons, 1981.

[5] Day, D. M. and Walsh, T., "Damped Structural Dynamics," Tech. Rep. SalinasDocuments/SandReport/qevp - Draft, Sandia National Laboratories, 2003.

[6] Larsen, M., "A Posteriori and a Priori Error Analysis for Finite Element Approximations of Self-Adjoint Elliptic Eigenvalue Problems," *SIAM Journal of Numerical Analysis*, **vol. 38**, no. 2, 2000, pp. 608–625.

[7] Heuveline, V. and Rannacher, R., "A Posteriori Error Control for Finite Element Approximations of Elliptic Eigenvalue Problems," *Advances in Computational Mathematics*, **vol. 15**, 2001, pp. 107–138.

[8] Oden, J. T. and Prudhomme, S., "Error Estimation of Eigenfrequencies for Elasticity and Shell Problems," *Mathematical Models and Methods in Applied Sciences*, **vol. 13**, no. 3, 2003, pp. 323–344.

[9] Ainsworth, M. and Oden, J. T., *A Posteriori Error Estimation in Finite Element Analysis*, John Wiley & Sons, Inc., first edn., 2000.

[10] Bernardi, C. and Verfurth, R., "Adaptive finite element methods for elliptic equations with non-smoooth coefficients," *Numerische Mathematik*, **vol. 85**, 2000, pp. 579–608.

[11] Duran, R., Padra, C., and Rodriguez, R., "A Posteriori Error Estimates for the Finite Element Approximation of Eigenvalue Problems," *Mathematical Models and Methods in Applied Sciences*, **vol. 13**, no. 8, 2003, pp. 1219–1229.

[12] Prudhomme, S., "personal communication," March 2004.

[13] Hughes, T. J. R., *The Finite Element Method–Linear Static and Dynamic Finite Element Analysis*, Prentice-Hall, Inc, 1987.

[14] Hughes, T. J. R., *The Finite Element Method–Linear Static and Dynamic Finite Element Analysis*, chap. Appendix 3.1, Prentice-Hall, Inc, 1987, p. 170.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Alvin, K., de la Fuente, H. M., Haugen, B., and Felippa, C. A., "Membrane triangles with corner drilling freedoms – I. The EFF element," *Finite Elements in Analysis and Design*, **vol. 12**, 1992, pp. 163–187.

[20] Felippa, C. A. and Militello, C., "Membrane triangles with corner drilling freedoms – II. The ANDES element," *Finite Elements in Analysis and Design*, **vol. 12**, 1992, pp. 189–201.

[21] Felippa, C. A. and Alexander, S., "Membrane triangles with corner drilling freedoms – III. Implementation and performance evaluation," *Finite Elements in Analysis and Design*, **vol. 12**, 1992, pp. 203–239.

[22] Wirsching, P. H., Paez, T. L., and Ortiz, K., *Random Vibrations, theory and practice*, John Wiley and Sons, Inc, 1995.

[23] Felippa, C. A., "The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation," Tech. Rep. CU-CAS-02-03, Univ. Colo. at Boulder, 2002.

[24] Auld, B. A., *Acoustic Fields and Waves in Solids, Second Edition*, vol. I, Robert E. Krieger Publishing Company, 1990.

A Anisotropic Materials

Here we discuss how anisotropic elasticity is implemented in Salinas.[8] The approach is reasonably standard, but a documentation here is necessary to specify which of the many conventions of material parameter numbering is used in Salinas. Further, it is useful to present the theoretical development for those who may do maintenance on this part of the code.

A.1 Linear Anisotropic Elasticity

Linear elasticity asserts that the stress is a linear function of the strain:

$$\sigma_{ij} = C^4_{ijkl}\epsilon_{kl} \tag{243}$$

Where C^4_{ijkl} are the Cartesian components of the fourth order constitutive tensor and the Einstein convention of summation on repeated indices is used.

A.2 Stress Vectors

By definition, the strain is symmetric. Further, we make the usual constitutive assumption that the stress is symmetric. This permits the representation of the 3x3 stress matrix and the 3x3 strain matrix each by a column vector having six rows.

$$s = \left\{ \begin{array}{c} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{array} \right\} \tag{244}$$

and,

$$e = \left\{ \begin{array}{c} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{array} \right\}.$$

This is the Voigt notation. Note that this mapping from σ to s and from ϵ to e is not universal. This is the numbering used in Malvern and seems to be popular in the materials science world, but it differs from the numbering used in NASTRAN and

[8] This is a transcription of Dan Segalman's framemaker document, "aniosConst.frm".

from the numbering in ABAQUS. Further, note that though the above are usually referred to as "stress vectors" and "strain vectors", they are not vectors in the sense that they map from one coordinate system to another as true vectors do. How that mapping is done is discussed in a later section.

We use the above to map the fourth-order tensor C^4_{ijkl} into a 6x6 matrix of material parameters. This is done with the aid of the matrices that formally map σ to s and from ϵ to e.

$$e_n = Enij\epsilon_{ij} \tag{245}$$

and

$$\epsilon_{ij} = e_n F_{nij} \tag{246}$$

where

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad E_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad E_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$E_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad E_5 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad E_6 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{247}$$

and

$$F_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad F_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$F_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1/2 \\ 0 & 1/2 & 0 \end{bmatrix} \quad F_5 = \begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 0 & 0 \\ 1/2 & 0 & 0 \end{bmatrix} \quad F_6 = \begin{bmatrix} 0 & 1/2 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & 0 \end{bmatrix} \tag{248}$$

We note that the stress mappings are also achieved with the above third order quantities:

$$s_n = F_{nij}\sigma_{ij} \tag{249}$$

and

$$\sigma_{ij} = snE_{nij} \tag{250}$$

From Equations 245 and 246 or Equations 249 and 250 we see that,

$$E_{mij}F_{nij} = \delta_{mn} \tag{251}$$

Substituting Equations 246 and 250 into Equation 243 and simplifying with Equation 251, we find

$$s_m = C_{mn}e_n \tag{252}$$

where

$$C_{mn} = F_{mij} C^4_{ijkl} F_{nkl} \tag{253}$$

Though above shows how to find the 6x6 matrix C_{ij} in terms of the fourth order tensor components C^4_{ijkl}, the material description is usually provided directly in terms of the components of C_{ij}.

A.3 Strain Energy and Orientation

We now address the situation where the matrix of material parameters of are provide in a Cartesian coordinate system different from the coordinate system (usually the global system) in which strains are calculated. Because stress and strain are tensors, they transfer from one coordinate system to another by:

$$\sigma_{ij} = R_{ai} \hat{\sigma}_{ab} R_{bj} \tag{254}$$

and

$$\epsilon_{ij} = R_{ai} \hat{\epsilon}_{ab} R_{bj} \tag{255}$$

where σ_{ij} and ϵ_{ij} are the stress and strain components calculated in some other (global) Cartesian system and R_{ai} are the components of the rotation matrix that rotates the basis vectors in that global system to that with respect to which the material properties are defined. A basis vector $\hat{b}_a$ in the local, material frame is expressed in terms of the basis vectors of the global system by:

$$\hat{b}_a = R_{ai} b_i \tag{256}$$

where b_1, b_2, and b_3 are the basis vectors of the global frame.

From Equations 249, 250, and 253, we find following

$$s_m = (F_{mij} E_{nab} R_{ai} R_{bj}) \hat{s}_n. \tag{257}$$

From Equations 245, 246, and 255, we find the more useful relationship

$$e_m = (E_{mij} F_{nab} R_{ai} R_{bj}) \hat{e}_n. \tag{258}$$

The above two transformations are simplified:

$$s = T^T \hat{s} \tag{259}$$

and

$$e = T \hat{e} \tag{260}$$

where the 6x6 transformation matrix, T, is defined

$$T_{nk} = E_{nij} F_{kab} R_{ai} R_{bj} = tr\left(E_n^T R F_k R^T\right) \tag{261}$$

Noting that

$$s = \hat{C}\hat{e}, \tag{262}$$

and substituting Equations 259 and 260 into Equation 262, we further find

$$s = T^T \hat{C} T e. \tag{263}$$

Comparing the above with Equation 252, we finally find that

$$C = T^T \hat{C} T \tag{264}$$

which was the main point of this exercise.

Note also that the components of arrays E_n and F_n are mostly zero, with the rest either 1 or 1/2. After using Maple to simplify the product matrix,we find that T has a fairly simple form.

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T22 \end{bmatrix} \tag{265}$$

where

$$T_{11} = \begin{bmatrix} R_{11}^2 & R_{12}^2 & R_{13}^2 \\ R_{21}^2 & R_{22}^2 & R_{23}^2 \\ R_{31}^2 & R_{32}^2 & R_{33}^2 \end{bmatrix}, \tag{266}$$

$$T_{12} = \begin{bmatrix} R_{13}R_{12} & R_{13}R_{11} & R_{13}R_{11} \\ R_{23}R_{22} & R_{23}R_{21} & R_{23}R_{21} \\ R_{33}R_{32} & R_{33}R_{31} & R_{33}R_{31} \end{bmatrix}, \tag{267}$$

$$T_{21} = \begin{bmatrix} 2R_{21}R_{31} & R_{22}R_{32} & R_{23}R_{33} \\ 2R_{11}R_{31} & R_{12}R_{32} & R_{13}R_{33} \\ 2R_{11}R_{21} & R_{12}R_{22} & R_{13}R_{23} \end{bmatrix}, \tag{268}$$

and

$$T_{22} = \begin{bmatrix} R_{23}R_{32} + R_{22}R_{33} & R_{23}R_{31} + R_{21}R_{33} & R_{22}R_{31} + R_{21}R32 \\ R_{13}R_{32} + R_{12}R_{33} & R_{13}R_{31} + R_{11}R_{33} & R_{12}R_{31} + R_{11}R32 \\ R_{13}R_{22} + R_{12}R_{23} & R_{13}R_{21} + R_{11}R_{23} & R_{12}R_{21} + R_{11}R22 \end{bmatrix}. \tag{269}$$

Note that T defined above is the transformation matrix N in of Equatoin 3.34 in Auld's *"Acoustic Waves in Solids, Volume I"* (reference 24), which is used in the same way.

The Maple code to perform the above calculations follows.

```
with(linalg);
E[1] := matrix(3,3,[ [1,0,0],[0,0,0],[0,0,0]]);
E[2] := matrix(3,3,[ [0,0,0],[0,1,0],[0,0,0]]);
E[3] := matrix(3,3,[ [0,0,0],[0,0,0],[0,0,1]]);
E[4] := matrix(3,3,[ [0,0,0],[0,0,1],[0,1,0]]);
E[5] := matrix(3,3,[ [0,0,1],[0,0,0],[1,0,0]]);
E[6] := matrix(3,3,[ [0,1,0],[1,0,0],[0,0,0]]);
F[1] := E[1];
F[2] := E[2];
F[3] := E[3];
F[4] := (1/2)*E[4];
F[5] := (1/2)*E[5];
F[6] := (1/2)*E[6];
R := matrix(3,3);

for k from 1 to 6 do
FRR[k] := matrix(3,3);
FRR[k] := evalm ( R &* F[k] &*transpose(R));
od;

T := matrix(6,6);
for k from 1 to 6 do
for n from 1 to 6 do
T[n,k] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
T[n,k] := T[n,k] +evalm(FRR[k][i,j])*E[n][i,j];
od; od;
od; od;

readlib(C);
C(T);

read("/home/djsegal/Maple/tools/maple2mif.mpl");
M := maple2mif();
fprintf("/home/djsegal/MPP/notes/temp.mif",'%s',M(eval(T))) ;
```

B Integration of Isoparametric Solids

We show below how one achieves effective selective integration of isoparametric solids in a manner that satisfies the standard conditions (such as the patch test) and also accommodates anisotropic materials.[9]

We begin with the definition of the strain vector. For computational convenience defines the stress and strain vectors:

$$s = \left\{ \begin{array}{c} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{array} \right\} \tag{270}$$

and,

$$\nu = \left\{ \begin{array}{c} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{array} \right\}. \tag{271}$$

These are related through the matrix of elastic constants.

$$s = C\nu \tag{272}$$

We now take a look at virtual work, since it is from virtual work that the stiffness matrix is derived.

$$\delta W = \int_V s^T \delta\nu dV = \int_V \nu^T C \delta\nu dV \tag{273}$$

If we select the above volume to be that of an element and use the strain-displacement matrices associated with each nodal degree of freedom,

$$\nu(x) = \sum_j B_j(x) u_j \tag{274}$$

where u_j is the j^{th} nodal degree of freedom, the virtual work becomes

$$\delta W = u_j \delta u_k \int_V B_j(x)^T C B_k(x) dV \tag{275}$$

[9] This is a transcription of Dan Segalman's framemaker document, "IsoInt.frm".

Since the element stiffness matrix is defined by

$$\delta W = u_j \delta K_{ij} \tag{276}$$

we conclude that

$$K_{ij} = \int_V B_j(x)^T C B_k(x) dV \tag{277}$$

The next step is to decompose the strain-displacement vectors into deviatoric and dilatational components.

$$B_j(x) = B_j^D(x) + B_j^V(x) \tag{278}$$

where,

$$B_j^V(x) = d_j(x) \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{279}$$

and $3d_j(x)$ is the sum of the first three rows of $B_j(x)$. $B_j^D(x)$ is defined by Equation 278. Substitution of Equation 278 into Equation 277 yields:

$$\begin{aligned} K_{ij} &= \int_V B_j^D(x)^T C B_k^D(x) dV + \int_V B_j^V(x)^T C B_k^V(x) dV + \cdots \\ &\quad + \int_V B_j^V(x)^T C B_k^D(x) dV + \int_V B_j^D(x)^T C B_k^V(x) dV \end{aligned} \tag{280}$$

For isotropic materials, the deviatoric and dilatational portions of the strain are orthogonal with respect to the matrix of material constants, so the last two integrals in the above equation are zero. It is sometimes common to integrate the contributions of each to the stiffness matrix using separate strategies. Such approaches can produce elements with slightly less susceptibility to parasitic shear. Such an approach does not work for elements of anisotropic material, so the following approach has been developed.

B.1 Uniform Strain-Displacement Matrices

At this point it is useful to define the element averaged strain displacement matrices.

$$\bar{B}_k = \frac{1}{V} \int_V B_k(x) dV \tag{281}$$

For hex elements, these are the strain-displacement matrices of the Flanagan and Belytschko, and are known as "uniform strain" elements. Elements formed by the above strain/displacement matrices are very "soft", having properties similar to elements formed by single point integration. Hex elements of this sort display extraneous zero-energy modes. In what follows, we consider linear combinations of this strain-displacement matrix formulation with the consistent formulation presented in Equation 274.

The uniform strain matrices are also separable into dilatational and deviatoric parts.

$$\bar{B}_k = \bar{B}_k^V + \bar{B}_k^D \tag{282}$$

B.2 Mixed Integration

The approach presented here builds on one presented by Hughes.[13] We can achieve the effect of softening elements by forming the strain displacement matrices from combinations of the consistent strain-displacement and the uniform strain displacement matrices.

$$\hat{B}_k(x) = \alpha\bar{B}_k^V + (1-\alpha)B_k^V(x) + \beta\bar{B}_k^D + (1-\beta)B_k^D(x) \tag{283}$$

(14) Note that for all values of α and β, the above correctly captures uniform strains. It is in how the non-uniform strains contribute to the stiffness matrix that the particular values of α and β make a difference. By setting values of α and β according to the following table, we recover the standard integration forms:

α	β	**Integration**
1	1	Flanagan and Belytschko
0	0	Full Integration
1	0	Selective Integration

We note that setting $\alpha = 1$ and using an intermediate value of β, we can achieve performance almost as good as that of the Flanagan and Belytschko element but without admitting hour-glass modes.

Index