LA-SUB--93-229

NONLINEAR RESPONSE OF PLAIN CONCRETE SHEAR WALLS WITH ELASTIC-DAMAGING BEHAVIOR

by

S. Yazdani and H. L. Schreyer

Departments of Civil and Mechanical Engineering
University of New Mexico
Albuquerque, New Mexico 87131

ABSTRACT

This report summarizes the theoretical and computational efforts on the modeling of small scale shear walls. Small scale shear walls are used extensively in the study of shear wall behavior because the construction and testing of full size walls are rather expensive. A finite element code is developed which incorporates nonlinear constitutive relations of damage mechanics. The program is used to obtain nonlinear load-deformation curves and to address the initial loss of stiffness due to shrinkage cracking. The program can also be used to monitor the continuous degradation of the fundamental frequency due to progressive damage.

ng

TABLE OF CONTENTS

I. Introduction

Shear walls are important structural members which support gravity loads and can be designed to resist horizontal forces due to wind or seismic action. Since the full size construction and testing of shear walls are expensive, many studies have been carried on scaled down models. One such investigation as a part of an ongoing research program has been conducted by Endebrock, et al. (1985) at the Las Alamos National Laboratory. In their work, shear wall models with high surface to volume ratios were used. The test results show that small scale models exhibit nonlinear behavior with an initial loss of stiffness and lower fundamental frequency than those computed with conventional linear elastic formulation or those observed in prototype walls.

The observed nonlinearities in the behavior of non-homogeneous cementitious materials arise from two distinct microstructural changes. One is the development of plastic flow along preferred dislocation planes under a high confining pressure. The second pattern involves the nucleation and propagation of microvoids and microcracks. Since the design and behavior of most conventional structural members, including shear walls, are under low or zero confining pressure it is plausible to assume that the nonlinear behavior of shear walls is strongly influenced by microcracking.

The presence of nonhomogeneities in the form of aggregate or reinforcement has microstructurally two conflicting effects. It is known for example that weak links develop at the interface of mortar and aggregate due to the accumulation of water lenses. These weak interface bonds are the sources of nucleation and propagation of microcracks. On the other hand, aggregate particles serve as crack arresters and help improve the strength and apparent ductility of mortar. The idea that aggregate particles act as energy barriers have been used to explain why high strength concrete behaves in a relatively more brittle fashion than normal strength concrete.

It is believed that reinforcement plays similar roles as manifested by numerous experiments. For most effective reinforcing action, it is essential that concrete and reinforcement deform together which in turn implies the necessity of developing strong bonds between them. The bond strength between the reinforcment and concrete is developed through (a) chemical adhesion, (b) natural roughness of the rebar, and (c) closely spaced rib-shaped deformation that provides a high degree of interlocking of the two materials. The majority of the bond strength develops through the interlocking of the two materials. The

experimental investigations by Ervin and Jundi (1969) have shown that bonding of rebars is nonlinear and in particular looses stiffness with alternating loading. With this nonlinear characterization of the bond and the nonlinear behavior of concrete it is not surprising to see that a finite element program with a linear elastic material model overestimates that response stiffness. Since the reinforcement plays little role during the early stages of loading, continuum damage mechanics can be used to predict the associated anisotropic degradation in stiffness of the shearwalls.

Another aspect of the behavior that may warrant consideration is the shrinkage cracking of concrete. Shrinkage is caused by the evaporation of the free water over what is needed for the hydration of cement gel. The rate and completeness of drying depends on ambient temperature, humidity, and the surface area that is available for the heat outflux. The work of Troxell, et al. (1958) has shown that, with the geometry of the mold unchanged, a chief factor which determines the amount of final shrinkage is the water content of the fresh concrete. In their investigation the same aggregates were used for all tests, but in addition to and independently of water content, the amount of cement was also varied from four to eleven sacks per cubic yard of concrete. This very large variation of cement content had only very minor effects on the amounts of shrinkage, compared with the effect of water content. Similar findings were reported by Mckeen and Ledbetter (1970). On the other hand, Hanssen and Mattock (1966) investigated the shrinkage characteristics of structural menbers made from the same batch but with different surface areas and volumes. They considered the surface to volume ratio to be the design parameter and concluded that members with higher surface to volume ratios showed higher shrinkage deformations.

This study addresses the issues outlined above. A finite element program incorporating continuum damage mechanics is described which can predict the nonlinear load-deformation curve and changes in the fundamental frequency of the structure due to damage. The study also shows that if initial damage attributed to shrinkage cracking is taken into account, a good correlation with experimental data can be obtained.

II. A Continuum Damage Mechanics Model

Continuous damage mechanics is concerned with the progressive weakening of solids due to the development of microcracks and microvoids. As the nucleation and growth of microcracks create voids in the system, the load carrying capacity and stiffness are reduced and material is termed damaged. There is a strong directionality associated with the process of damage in general. The detail of the formulation will not be given here as it has been already compiled (Yazdani and Schreyer, 1988).

Consider small and isothermal deformations and note that for small deformations the additive decomposition of tensors of any rank is permissible. Let $\mathbf{C}$ be the fourth-order material compliance tensor which is solely affected by the extent of microcracking. If the initial material compliance is denoted by $\mathbf{C}^0$ and the added flexibility is given by $\mathbf{C}^c$ the following relation holds:

$$\mathbf{C}(k) = \mathbf{C}^0 + \mathbf{C}^c(k) \tag{1}$$

where k is a scalar parameter defining the extent of accumulated damage. The Cauchy stress tensor and the compliance of the material are related by means of Gibbs potential energy, G, as follows:

$$\frac{\partial^2 G(\boldsymbol{\sigma},k)}{\partial\boldsymbol{\sigma}\partial\boldsymbol{\sigma}} = \mathbf{C}(k) \tag{2}$$

The integration of Equation (2) with respect to $\boldsymbol{\sigma}$ and the use of Equation (1) yields:

$$\boldsymbol{\epsilon} = \frac{\partial G(\boldsymbol{\sigma},k)}{\partial\boldsymbol{\sigma}} = \mathbf{C}^0{:}\boldsymbol{\sigma} + \mathbf{C}^c(k){:}\boldsymbol{\sigma} + \boldsymbol{\epsilon}^r(k) \tag{3}$$

where $\boldsymbol{\epsilon}$ denotes the total strain tensor and $\boldsymbol{\epsilon}^r(k)$ arises as a variable of integration and is interpreted as the inelastic component of the deformation. If $\boldsymbol{\epsilon}^r(k)$ is set to zero, the conventional definition of an elastic-perfectly damaging behavior is retained. This type of behavior is called elastic damage (Yazdani and Schreyer, 1988) to distinguish it from any inelasticity that may develop from inelastic fracturing. This inelastic feature may arise due to imperfect microcracking and crack tip process zones. To understand the structure of Equation (3) better, the rate form of this equation is needed:

$$\dot{\epsilon} = \mathbf{C}:\dot{\sigma} + \dot{\mathbf{C}}^{c}:\sigma + \dot{\epsilon}^{r}(k)$$
$$= \dot{\epsilon}^{e} + \dot{\epsilon}^{D}(k) + \dot{\epsilon}^{r}(k) \tag{4}$$

where $\dot{\epsilon}^{e}$ is the ratio of the elastic deformation in the absence of any further microcracking, $\dot{\epsilon}^{D}(k)$ is the rate of deformation due to damage, and $\dot{\epsilon}^{r}(k)$ defines the rate of inelastic strain tensor. In order to complete the theory two kinetic relations must be postulated and developed. One formulation is needed to describe the evolution of $\dot{\epsilon}^{D}(k)$ and a second relation is required to define $\dot{\epsilon}^{r}(k)$. The first task is the identification of a kinetic equation for $\dot{\mathbf{C}}^{c}(k)$ satisfying the following relation

$$\dot{\epsilon}^{D}(k) = \dot{\mathbf{C}}^{c}(k) : \sigma \tag{5}$$

Following the approach by Ortiz (1985) the stress tensor and the associated rate of added flexibility tensor, $\dot{\mathbf{C}}^{c}$, are decomposed into modes I and II of microcracking as follows:

$$\dot{\epsilon}^{D}(k) = (\dot{\mathbf{C}}^{c}{}_{I}(k) + \dot{\mathbf{C}}^{c}{}_{II}(k)) : (\sigma^{+} + \sigma^{-}) \tag{6}$$

where σ^{+} and σ^{-} are the positive and negative cones of the stress tensor and subscripts I and II refer to modes I and II of fracturing, respectively. Mode I refers to the cleavage type cracking and is shown in Figure 1a. Mode II is a more complicated type of cracking and involves crack sliding and surface opening simultaneously (Figure 1b). It is implicitly assumed that these modes are decoupled in the sense that σ^{+} leads only to mode I cracking and mode II is activated if σ^{-} is nonzero and satisfies the loading condition. Two fourth-order response functions are further developed such that

$$\dot{\mathbf{C}}^{c}{}_{I} = k\, \mathbf{R}_{I} \tag{7}$$

and

$$\dot{\mathbf{C}}^{c}{}_{II} = k\, \mathbf{R}_{II} \tag{8}$$

These tensorial functions determine the direction of incurring damage. Guided by experimental and other theoretical studies, the following form was

adopted for $\mathbf{R}_I$:

$$\mathbf{R}_I = \frac{\boldsymbol{\sigma}^+ \otimes \boldsymbol{\sigma}^+}{\boldsymbol{\sigma}^+ : \boldsymbol{\sigma}^+} \tag{9}$$

where the symbol $\otimes$ denotes the tensor product and : indicates a C_{12} contraction operation. The compressive mode of cracking or mode II is the result of simultaneous action of shear sliding of an existing inclined microcrack and the opening of crack sides. It is also understood that the mean normal pressure impedes the formation of mode II fracturing, while no damage can take place under purely hydrostatic pressure. To accomodate these features and to address the increase in the apparent Poisson's ratio that cementitious materials exhibit, $\mathbf{R}_{II}$ is decomposed into two parts:

$$\mathbf{R}_{II} = \mathbf{R}^d{}_{II} + \mathbf{R}^h{}_{II} \tag{10}$$

The first part is postulated to be:

$$\mathbf{R}^h{}_{II} = w \frac{\tilde{\boldsymbol{\sigma}} \otimes \tilde{\boldsymbol{\sigma}}}{\tilde{\boldsymbol{\sigma}} : \tilde{\boldsymbol{\sigma}}} \tag{11}$$

where

$$\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma}^- - \lambda \mathbf{I} \tag{12}$$

and where λ is the maximum eigenvalue of $\boldsymbol{\sigma}^-$. The material parameter w accounts for the relative strength of concrete in tension and compression. The form for $\mathbf{R}^h{}_{II}$ is given by Yazdani and Schreyer (1988) to be:

$$\mathbf{R}_{II}{}^h = w \, \alpha \, H(-\tilde{\lambda})(\mu \underline{\mathbf{I}} - \mathbf{I} \otimes \mathbf{I}), \qquad \mu \geq 1 \tag{13}$$

where α is a parameter that brings in the effect of lateral pressure on the stress-strain response of concrete and $\tilde{\lambda}$ and $H(\cdot)$ are the minimum eigenvalue of $\tilde{\boldsymbol{\sigma}}$ and the Heaviside function, respectively. $\underline{\mathbf{I}}$ is the fourth-order identity tensor and μ is a scalar coefficient which is determined from experimental observations. These formulations imply that damage occurs in the direction of minimum eigenvalues (maximum absolute values) of $\boldsymbol{\sigma}^-$ and that no damage is predicted for a purely

hydrostatic compression path. Yazdani and Schreyer also proposed the following simple evolution equation for the inelastic part of the strain tensor:

$$\dot{\epsilon}^{r}(k) = kw\mathbf{S}^{-} + kw\beta\mathbf{S}^{+} \tag{14}$$

where β is a scalar coefficient that is determined from experimental records and is greater than one for permanent deformation. The negative and positive cones of σ^{-d}, the deviatoric component of σ^{-}, are denoted by $\mathbf{S}^{-}$ and $\mathbf{S}^{+}$, respectively. In this approach it is assumed that the inelastic damage is only associated with mode II cracking. If $\beta = 1$ no inelastic volumetric deformation is predicted. A particular damage surface can be obtained by using the special forms of $\mathbf{R}_I$ and $\mathbf{R}_{II}$ and Equation 14 as follows:

$$\Psi(\sigma,k) = \frac{1}{2}\sigma^{+}:\sigma^{+} + \frac{w}{2}\sigma^{-}:\frac{\tilde{\sigma}\otimes\tilde{\sigma}}{\tilde{\sigma}:\tilde{\sigma}}:\sigma^{-} + w(\mathbf{S}^{-} + \beta\mathbf{S}^{+}):\sigma + \frac{w}{2}\alpha\mu H(-\lambda)\sigma:\sigma$$
$$- \frac{9w\alpha}{2}H(-\lambda)\,P^2 - \frac{1}{2}\,t^2(k) = 0 \tag{15}$$

where P is the mean pressure and t(k) denotes the damage function. The damage function may also be called the critical stress (Ortiz, 1985). It is implicit in the formulation that the form for t can be established from experimental data for any stress path. An exponential function proposed by Smith and Young (1955) is used in obtaining a continuous damage function as follows:

$$\sigma_t = f_t\frac{\epsilon_t}{\epsilon_u}\,EXP(\,1 - \frac{\epsilon_t}{\epsilon_u}) \tag{16}$$

where f_t and ϵ_u are the tensile strength and the associated tensile strain of concrete, respectively. Since mode I cracking is assumed to be perfectly brittle, the damage parameter ,k, can be interpreted as the added flexibility under uniaxial stress so that for any point on the stress-strain curve the following relation holds:

$$\sigma_t = \frac{\epsilon_t}{\frac{1}{E_0} + k} \tag{17}$$

Equations (16) and (17) can be combined to yield the damage function. The limit damage surface is obtained by setting $t =. f_t$. For biaxial stress paths this surface is shown in Figure 2 which also contains the average test values of Liu, et al. (1972) and Andenaes, et al. (1977).

It should be mentioned that the alteration of elastic properties and its effect on the structural stiffness is solely predicted by the elastic damage relations. For this reason, the finite element program is developed for the elastic-perfectly damaging material. The inclusion of the inelastic damage relation in the damage surface is for the completeness of the theory.

Initial Stiffness and Shrinkage Cracking

The theoretical and experimental investigations by Picket (1946) have shown that a reasonable agreement between measured shrinkage strains and values calculated from the diffusion equation can be achieved. Picket has suggested that shrinkage deformation of concrete follows approximately the laws of diffusion similar to those expressing the flow of heat. Hanssen and Mattock (1964) followed the approach by Pickett and produced experimentally obtained design curves as is shown in Figures 3a and 3b. They concluded that although volume/surface ratio does not reflect perfectly variations of both sizes and shapes, nevertheless, the degree of correlation found between the theory and experiment is satisfactory for purposes of engineering modeling and design.

These investigations have not addressed the effect of shrinkage cracking on the material and structural stiffnesses. At the present time the material parameters obtained from the standard cylinder tests are used by analysts to calibrate the theoretical models. There are convincing experimental results to indicate that the use of values from cylinder tests (volume/surface area = 1.5) may become inappropriate. One such attempt was made by Endebrock, et al. (1985) to analyze the deformation characteristics of model shear walls. The finite element prediction of the structural stiffness has overestimated the measured response by as much as three times. Such discrepencies are believed to lie in the use of inappropriate material parameters. The shear wall used by Endebrock, et al. (1985) is sketched in Figure 4 and has a volume to surface ratio of 0.47. The design curves of Hannsen and Mattock (1966), Figures 3a and 3b, indicate that for this ratio, the shrinkage deformation is 1.4 times greater than that of standard cylindrical specimens after 50 days of casting. This can, at least partially, explain the observed discrepencies.

To improve on this perceived problem, it is plausible to assume that shrinkage cracking has no preferential direction so that the degradation in the initial stiffness is isotropic. Let $\mathbf{C}^0$ be the initial compliance tensor to be defined as

$$\mathbf{C}^0 = (\beta_1 + \beta_2)\,\underline{\mathbf{I}} - \beta_2\,\mathbf{I} \otimes \mathbf{I} \qquad \text{if } tr(\epsilon) \geq 0 \tag{18}$$

where $\underline{\mathbf{I}}$ and $\mathbf{I}$ are the fourth order and second order identity tensors, respectively, and β_1 and β_2 denote material parameters to be determined from two different loading paths. The effect of these parameters on the structural responses are illustrated and discussed in section IV of the report.

III. The General Formulation of the FEM Program

It can be seen that the governing constitutive equations outlined in section II of this report are nonlinear and therefore an incremental approach should be taken in the solution algorithm. Let K^{i-1} be the global stiffness matrix of the structure corresponding to the ith loading step assembled from the information obtained at the previous step i-1. The governing matrix equation for the static problem is given by

$$[K^{i-1}]\,[\Delta u^{i}] = [\Delta P^{i}] \tag{19}$$

where Δu^{i} is the global incremental nodal displacement vector at the ith step and ΔP^{i} denotes the incremental loading vector also at the ith step. Equation 19 yields the nodal displacement vector, Δu^{i}, which can be used to obtain the corresponding incremental nodal strain vector, $\Delta\epsilon^{i}$, for each element as follows:

$$[\Delta\epsilon^{i}{}_{elem}] = [B^{i}{}_{elem}]\,[\Delta u^{i}{}_{elem}] \tag{20}$$

where $[B]$ is the strain-displacement matrix. Since the stiffness terms are computed numerically at the Gauss points, the program was modified to interpolate strains over the element using the element shape functions, N. The resulting expressions were evaluated at Gauss points:

$$[\Delta\epsilon^{i}{}_{elem,k}] = [\,N_{elem}\,]\,[\Delta\epsilon^{i}{}_{elem}] \tag{21}$$

where $\Delta\epsilon^{i}{}_{elem,k}$ denotes the incremental strain vector corresponding to an element "*elem*" and a Gauss point "k". The strains at the Gauss points are desired because damage is computed at Gauss points and the damage subroutine is a strain driven algorithm. Let A^{T} be the transformation matrix which can be used to transform a given matrix, M, to a diagonal matrix ,D, as follows:

$$[D] = [A^{T}]\,[\,M\,]\,[\,A\,] \tag{22}$$

Such an operation can be done with respect to the components of the strain tensor to obtain the corresponding eigenvalues to be used in the damage subroutine. With this, the following subsection explains the structure of the solution algorithm used in the numerical integration of the anisotropic damage model.

The structure of the damage solution algorithm

A numerical code for solving nonlinear constitutive relations is presented in this section. The code applies to an elastic-damaging material where the damage function has both hardening and softening features. The continuous alteration of elastic properties is a major feature of damage theory. For a given load increment, and after solving Equation 21, the governing set of equations that need to be solved are:

$$\epsilon = \epsilon_0 + \Delta\epsilon \tag{23a}$$

$$\sigma = D : \epsilon^e \tag{23b}$$

$$\epsilon^e = \epsilon - \epsilon^{*D} \tag{23c}$$

$$\epsilon^{*D} = \epsilon^{*D}{}_0 + \Delta\epsilon^D \tag{23d}$$

$$\Delta\epsilon^D = \Delta k \frac{\partial \Psi}{\partial \sigma} \tag{23e}$$

$$k = k_0 + \Delta k \tag{23f}$$

$$\Psi(\sigma, k) = 0 \tag{23g}$$

where ϵ^{*D} represents the components of the total strain tensor, ϵ, due to elastic damage. The elastic strain tensor for an undamaged material is denoted by ϵ^e and subscript 0 refers to the previous time step as before. An increment is indicated by Δ and k is a scalar measure of cumulative damage. The damage surface is given by Ψ, and σ and D represent the updated stress and stiffness tensors, respectively.

The underlining problem is that given an increment in total strain obtained form the previous step, ϵ^e ,ϵ^{*D} , σ ,D and Δk are unknown. An iterative approach for solving Equations 23 is explained as follows:

Step 1: Enter the damage subroutine with an increment in total strain.

Step 2: Assume step is elastic.

$$\Delta\epsilon^e = \Delta\epsilon$$

$$I = 0 \; (I = \textit{iteration count})$$

Step 3: Adjust the strain increment for the particular path chosen, compute the increment in stress, and update the stress.

$$\Delta\sigma = D : \Delta\epsilon^e \tag{24}$$

$$\sigma_I = \sigma_{I-1} + \Delta\sigma$$

$$I = I + 1$$

Step 4: Compute the damage function and check the damage condition.

(a) If $|\Psi| < \bar{\epsilon}$ and I $= 1$ (i.e. the first step), then the solution is elastic and the stress σ is correct. Go to step 7.

(b) If $|\Psi| < \bar{\epsilon}$ and I > 1, where $\bar{\epsilon}$ is a specified tolerance, then a damage solution has been obtained and the stress σ is correct. Go to step 7.

(c) Else, go to step 5

Step 5: For I $= 1$, prescribe an initially small increment of Δk.

For I > 1, use the Secant Method:

$$(\Delta k)_I = -(\Delta k)_{I-1} \frac{\Psi_I}{\Psi_I - \Psi_{I-1}} \tag{25}$$

Step 6: Calculate the added flexibility, increments of elastic and inelastic damage strains and update all variables.

$$(\Delta C^c)_I = (\Delta k)_I \, R \tag{26}$$

where C^c is the added flexibility tensor and R is the fourth order response tensor determining the direction of incurring damage given by Equation (20). Then

$$(C^c)_I = (C^c)_{I-1} + (\Delta C^c)_I \tag{27}$$

$$C = C_{I-1} + (C^c)_I \tag{28}$$

where C is the current compliance tensor.

$$D = C^{-1} \tag{29}$$

$$(\Delta\epsilon^D)_I = (\Delta\epsilon^D)_{I-1} + (\Delta k)_I \left(\frac{\partial\Psi}{\partial\sigma}\right)_I \tag{30}$$

$$(\Delta\epsilon^e)_I = (\Delta\epsilon^e)_{I-1} - (\Delta k)_I \left(\frac{\partial\Psi}{\partial\sigma}\right)_I \tag{31}$$

$$k = k_0 + (\Delta k)_I \tag{32}$$

Go to step 3.

Step 7: Update elastic, damage and total strain.

$$\epsilon^e = \epsilon^e{}_0 + \Delta\epsilon^e \tag{33a}$$

$$\epsilon^{*D} = \epsilon^{*D}{}_0 + \Delta\epsilon^D \tag{33b}$$

$$\epsilon = \epsilon^e + \epsilon^{*D} \tag{33c}$$

Step 8: Return to the main program for another load increment.

The plain stress condition of shear wall problems is handled during each loading increment in the damage subroutine. As was explained in step 3 before, the strain increment vector is adjusted for a particular path. For the plain stress path, the increment in the third normal strain component is:

$$\Delta\epsilon^e{}_{33} = -[D_{3311}\Delta\epsilon^e{}_{11} + D_{3322}\Delta\epsilon^e{}_{22}] / D_{3333} \tag{34}$$

where D_{3311}, D_{3322}, and D_{3333} represent the components of the updated stiffness tensor, respectively. The components of the elastic incremental strain vector are given by $\Delta\epsilon^e{}_{11}$, $\Delta\epsilon^e{}_{22}$, and $\Delta\epsilon^e{}_{33}$.

IV. Problem and results

The finite element program developed was used to obtain the theoretical response features of the model shear wall used by Endebrock et al. (1985). Figures 5 and 6 show different mesh sizes, boundary conditions, loading functions, and the resulting load-deformation curves for the convergence study. It was determined that with 24 elements, convergence is sufficiently achieved and that this mesh arrangement would be used for the other examples. The material parameters for Figure 6 are given in Table 1. It should be mentioned that for a monotonic convergence to the damage surface in the constitutive equation subroutine, the strain increment of the order E-06 to E-05 should be used. This can be achieved by specifying small increments in the loading function. Loading increments of 5E-04 kips to 5E-03 kips were used and the resulting load-deformation curves were practically identical.

For the initial cracking and its effect on the material stiffness the following modification of Equation 18 is considered. Let E and ν denote the Young's modulus and Poisson's ratio, respectively, of a material obtained from standard cylinder tests. For the first load increment when $tr(\Delta\epsilon) \geq 0$, define two other parameters α_1 and α_2 such that $\beta_1 = \alpha_1 \frac{1}{E}$ and $\beta_2 = -\alpha_2 \frac{\nu}{E}$. For a special condition where α_1 and α_2 are both one, the conventional isotropic compliance tensor corresponding to the standard cylinder parameters are obtained.

Figure 7 shows the load-deformation curve and the nonlinear character that is associated with progressive damage. The linear elastic response also obtained from the program and the experimentally obtained values for the initial stiffness by Endebrock, et al. (1985) is also plotted. The material parameters for this example are tabulated in Table 2. It should be mentioned that a significant reduction in the material stiffness and its subsequent effect on the nonlinear load-deformation curve becomes only noticeable when material elements are strained beyond the limit state into the softening regime. A real structure can not sustain such a degree of distress and the role of reinforcing steel becomes important. Steel reinforcement was not modeled in this analysis. Figure 7 also illustrates the evolution of the damage and its effect on the response.

The variation of the initial stiffness due to varying values of α_1 is plotted in Figure 8 where the material parameters of Table 3 are used. It was found that the response is more sensitive to the variation of α_1 when the mesh arrangement is more coarse. The reason is that the value of α_1 is implemented only under the

condition when the trace of the total strain increment is positive. It was also found that the variation of the second parameter, α_2, keeping α_1 constant, does not introduce any noticeable change in the load-deformation curve.

The changes of the fundamental frequencies of the structure due to progressive damage can be obtained by solving the characteristic equation $det[K - M\,\omega^2] = 0$ where global stiffness and mass matrices are given by K and M, respectively, and ω denotes a natural frequency of the system. The eigenvalues were obtained at each load level using an eigenvalue solver program available at the University of New Mexico. The fundamental frequency is plotted with respect to the applied loading in Figure 9 for the material parameters of Table 2, and the plot shows that the eigenvalue decreases with progressive degradation of the structural stiffness. This finding is consistent with the experimental work of Endebrock, et al. (1985) where the fundamental frequencies of several shear walls are found to decrease with progressive damage.

V. Conclusion

A finite element program was developed. to incorporate the nonlinear material behavior due to progressive damage. The program requires a large amount of memory since the stiffness and flexibility matrices, the stress vector, the critical stress, and the cumulative damage parameter, among others, are stored for each Gauss point of every element. This is, however, a problem with most nonlinear codes and is not exclusive to the damage mechanics model.

A simple approach to the loss of stiffness due to shrinkage cracking is also outlined and its effect on the structural response is shown. With the lack of experimental investigations on the pattern of shrinkage cracking, an isotropic degradation of the initial stiffness is suggested. The use of initial damage, however, can explain the discrepancy in observed initial elastic response of two different sizes of models for shear walls.

The last part of the study was concerned with the changes in the fundamental frequencies with progressive damage. The reduction in natural frequency with the load is consistent with experimental observations.

This report shows that continuum damage mechanics is a potentially useful framework for reflecting the effects of initial shrinkage cracks and the reduction in natural frequencies of loaded concrete members. Although the development of such models is relatively new, there is a significant potential for accurately reflecting the behavior of concrete structures that display damage. However, for a direct correlation between theoretical and experimental data, the effects of reinforcing steel should be incorporated in the numerical simulation of the response of the shear walls.

Acknowledgment

This investigation was supported financially by the Los Alamos National Laboratory.

VI. References

- Andenaes, E., K. Gerstle and H.Y. Ko (1977), "Response of Mortar and Concrete to Biaxial Compression," J. Engng. Mech. Div., ASCE, Vol. 103, p.515.

- Endbrock, E.G., R.C. Dove, and W.E. Dunwoody (1985), "Analysis and Tests on Small Shear Walls," Report no. LA-10433-MS

- Ervin, S. P. and N. Jundi (1969), "Pullout Bond Stress Distribution Under Static and Dynamic Repeated Loadings," ACI, Vol. 66, No. 28, pp. 377.

- Hanssen, T.C. and A.H. Mattock (1966), "The Influence of Size and Shape of Member on the Shrinkage and Creep of Concrete," ACI Journal, Vol. 63, p.26.

- Liu, T.C.Y., A.H. Nilson and F.O. Slate (1972), "Stress-Strain Response and Fracture of Concrete in Uniaxial and Biaxial Compression", Proc. ACI, Vol. 69, No. 5, p. 291

- Mckeen R. G. and W. B. Ledbetter (1970), "Shrinkage-Cracking Characteristics of Structural Light Weight Concrete," ACI, Vol. 69, No. 44, pp. 769.

- Picket, G. (1946), "Shrinkage Stresses in Concrete," ACI Journal, Vol. 42, no. 3, p.165.

- Smith, G.M. and L.E. Young (1955), "Ultimate Theory in Flexure by Exponential Function", Proc. ACI, Vol. 52, No. 3, p. 349

- Troxell, G. D., G. E. Raphael, and H. E. Davis (1958), "Long-Time Creep and Shrinkage Tests of Plain and Reinforced Concrete," ASTM Proceedings, Vol. 58, pp. 1101-1120.

- Yazdani, S. and H.L. Schreyer (1988), "An Anisotropic Damage Model for Concrete with Dilatation," Mechanics of Materials, to appear.

Parameters	Values
E	4000 ksi (Young's modulus)
ν	0.20 (Poisson's ratio)
f_c	4 ksi (Uniaxial compressive strength)
f_t	.5 ksi (Uniaxial tensile strength)
α	0.00112
w	0.0067
μ	1
α_1	1.0
α_2	1.0

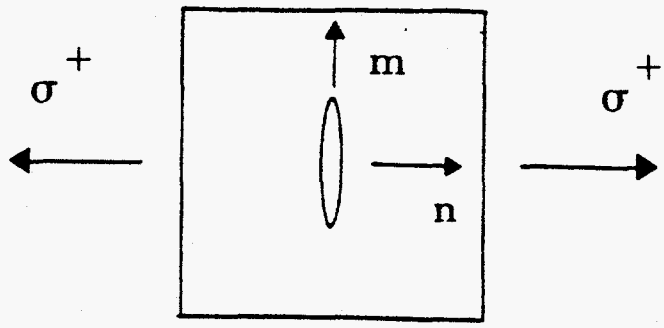
Table 1. Material parameters for the results shown in Figure 6.

Parameters	Values
E	4000 ksi (Young's modulus)
ν	0.20 (Poisson's ratio)
f_c	4 ksi (Uniaxial compressive strength)
f_t	.5 ksi (Uniaxial tensile strength)
α	0.00112
w	0.0067
μ	1
α_1	0.80
α_2	1.0

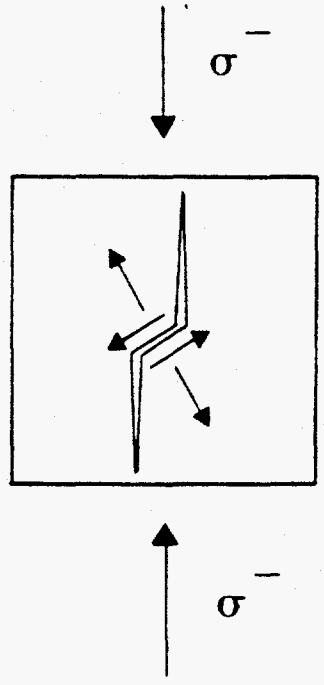
Table 2. Material parameters for the results shown in Figures 7 and 9.

Parameters	Values
E	4000 ksi (Young's modulus)
ν	0.20 (Poisson's ratio)
f_c	4 ksi (Uniaxial compressive strength)
f_t	.5 ksi (Uniaxial tensile strength)
α	0.00112
w	0.0067
μ	1
α_1	Variable
α_2	1.0

Table 3. Material parameters for the results shown in Figure 8.



(a) Mode I



(b) Mode II

Figure 1. Schematic representation of crack opening in modes I and II.

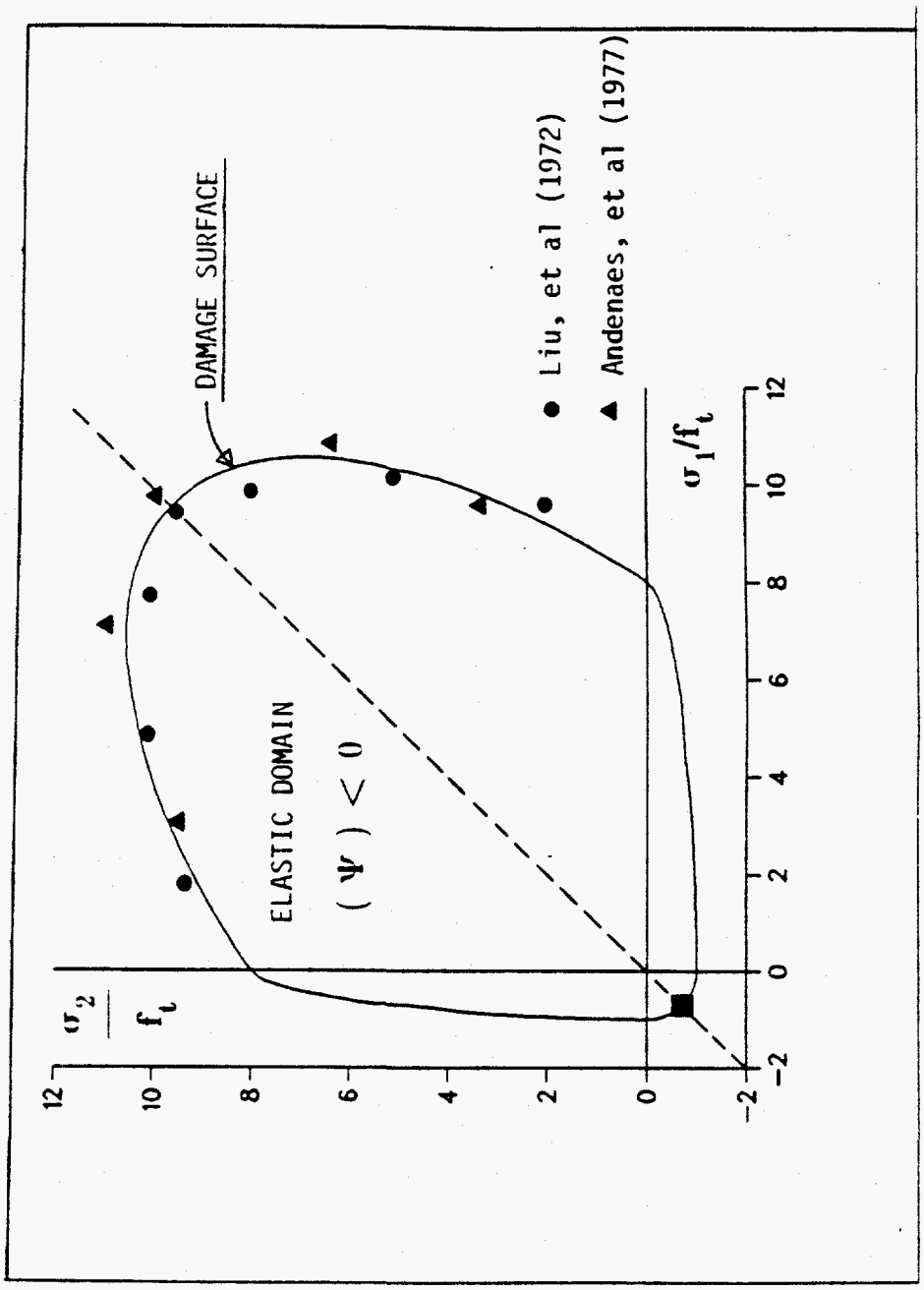
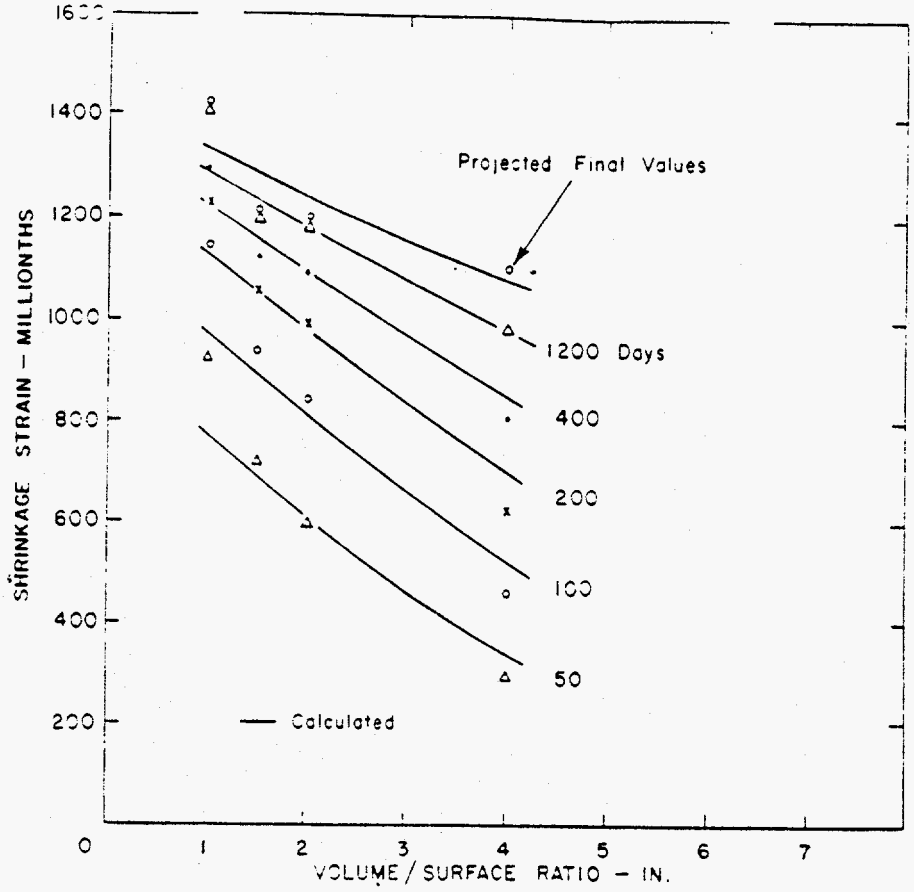
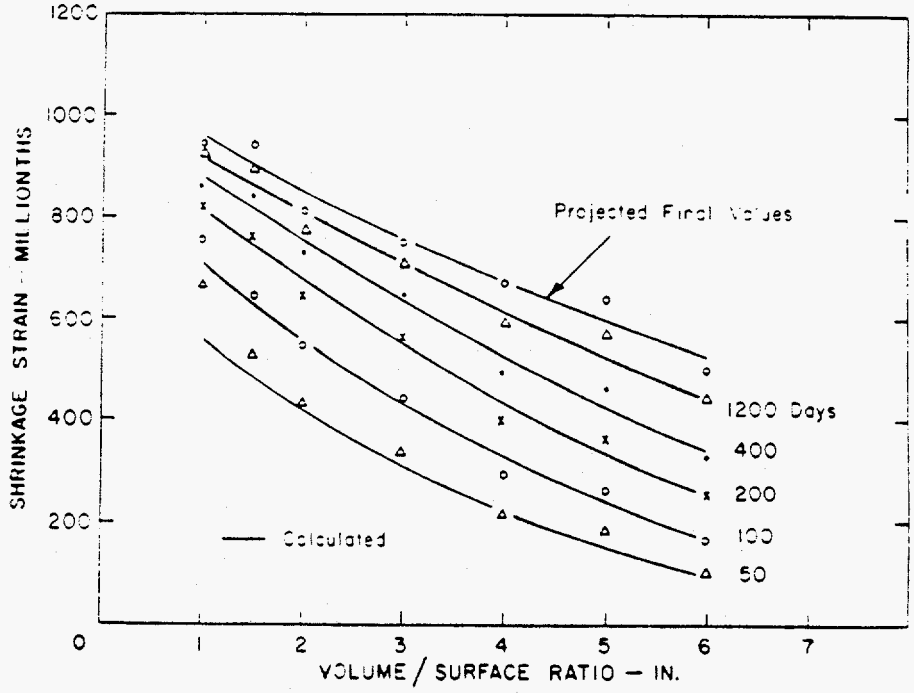


Figure 2. A biaxial strength envelope for concrete based on the damage model.



(a) Sand-stone aggregate concrete



(b) Elgin gravel aggregate concrete

Figure 3. Variation of shrinkage with volume/surface ratio, at different ages (Hanssen and Mattock, 1966).

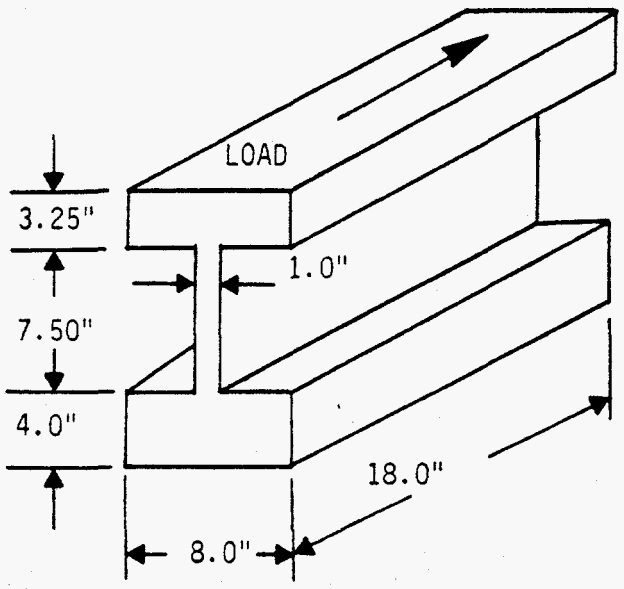
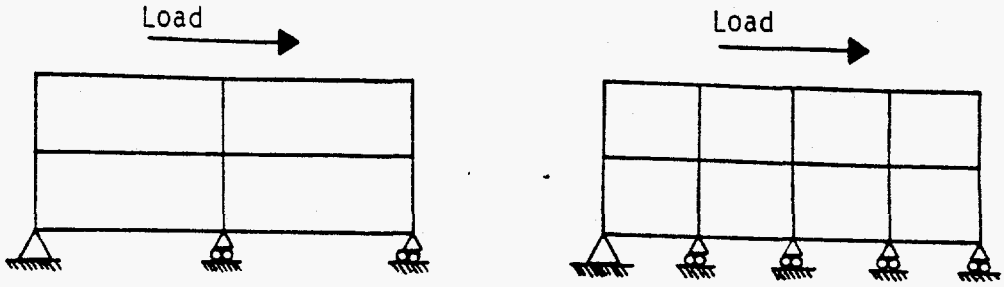
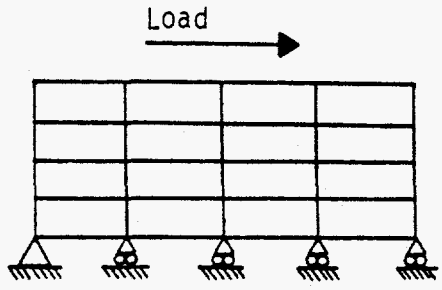


Figure 4. A schematic representation of the one-story model shear wall used by Endebrock, et al. (1985).

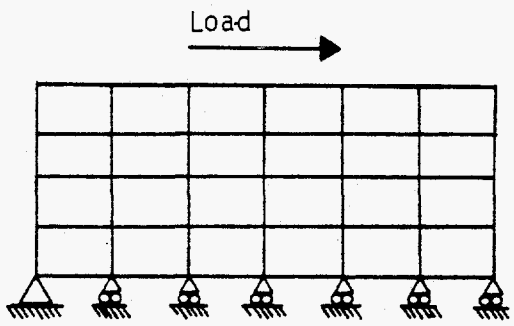


(a) 4-Element Mesh

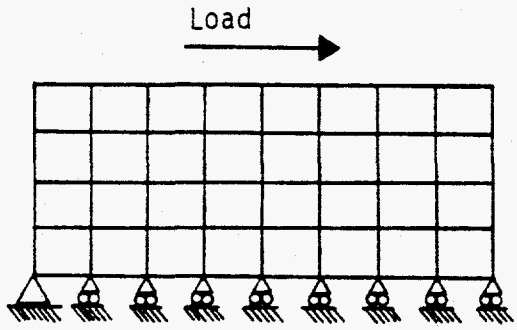
(b) 8-Element Mesh



(c) 16-Element Mesh



(d) 24-Element Mesh



(e) 32-Element Mesh

Figure 5. A schematic representation of mesh arrangement, loading, and boundary conditions.

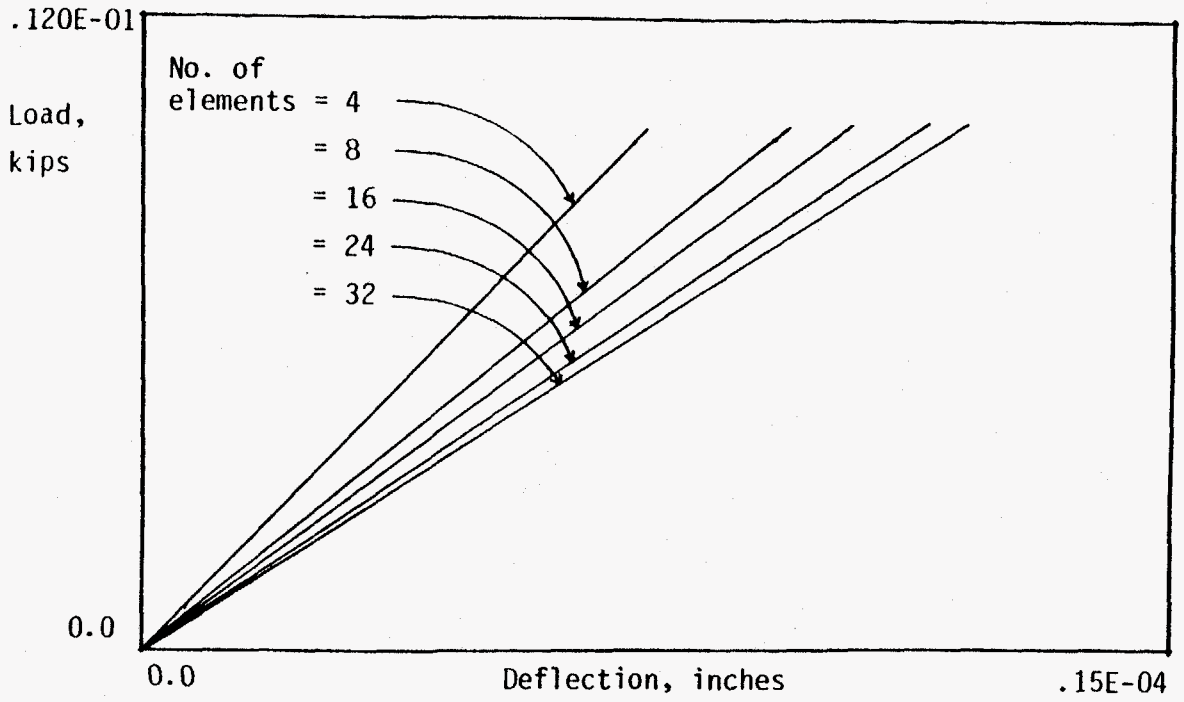


Figure 6. Load-deflection curves for a convergence study based on the mesh arrangements of Figure 5.

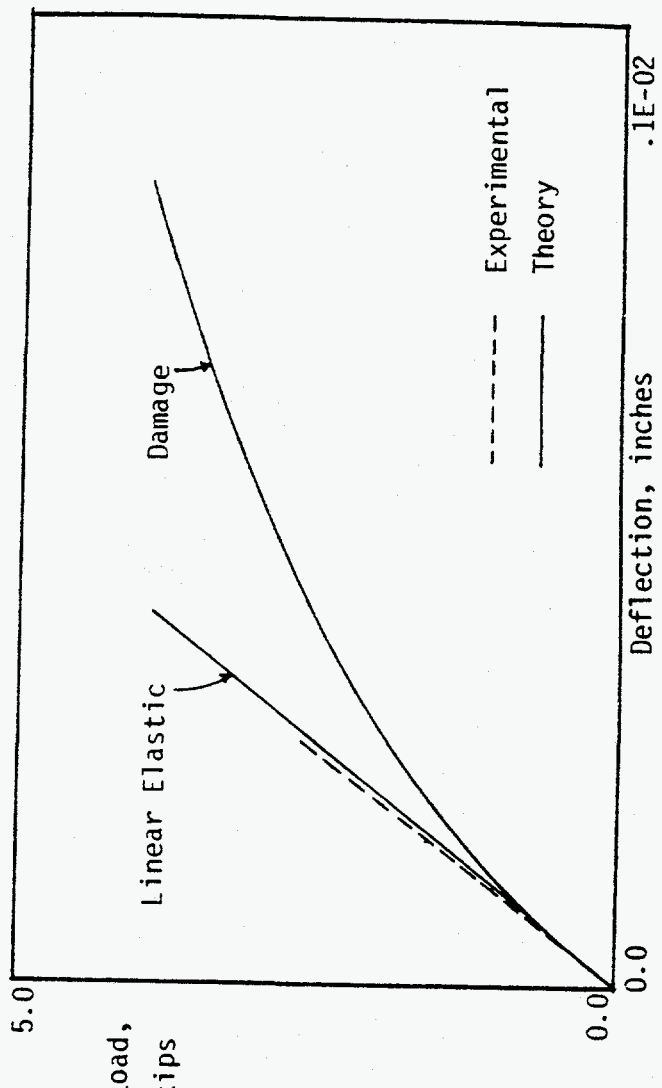


Figure 7. Theoretical load-deflection curve for a plain concrete shear wall with elastic-fracturing behavior.

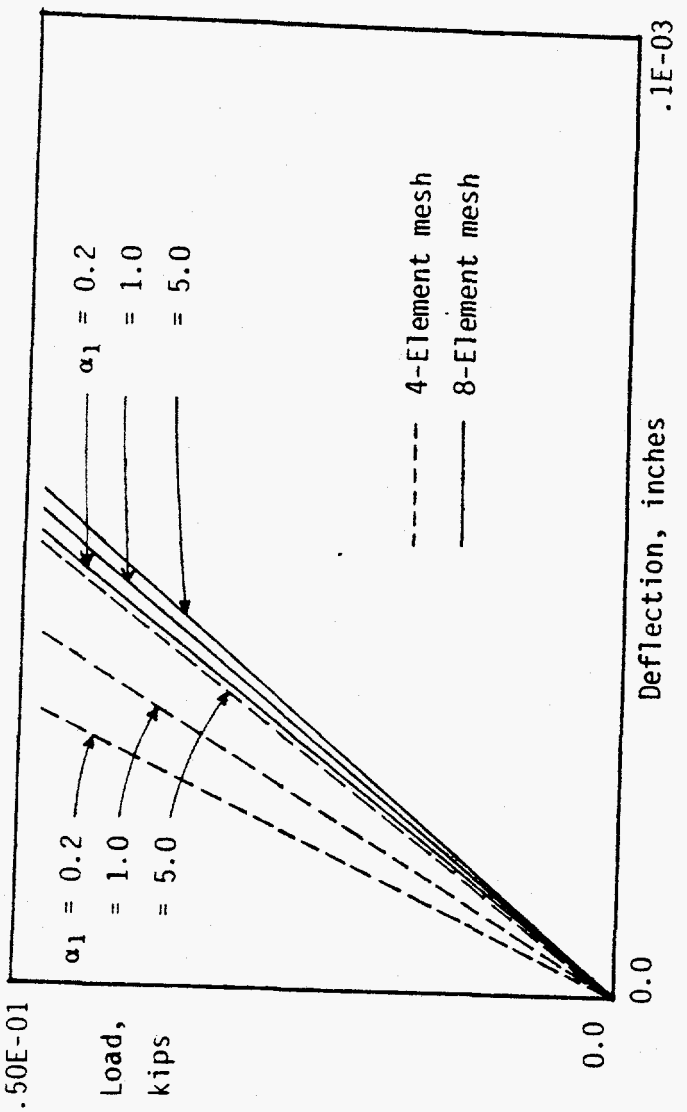


Figure 8. Load-deflection curves for different values of α_1.

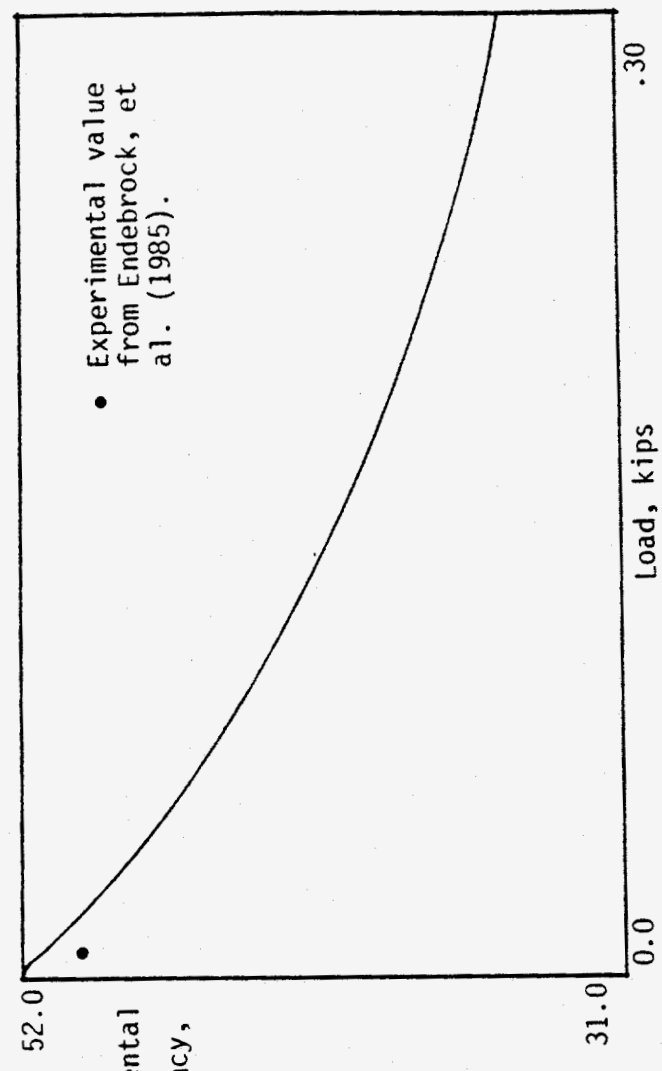


Figure 9. The degradation of fundamental frequency of the structure due to damage.

VII. Appendix

The listing of the finite element program together with a simple flow chart representation and a sample of an input data is given in this section. The program consists of a driver and twenty one subroutines. Some of the subroutines are taken from a graduate level finite element class taught by Drs. Roy Johnson and Walter Gerstle in the department of Civil Engineering at the University of New Mexico. These include subroutines input, stif, stifQ4, elast, n4, nn4, disout, recovr, jacob, mult, matadd, aeqnum, band, assmbl, and solver of which the first eight were modified to be compatible with the nonlinear character of the problem. The remaining subroutines were developed by the authors and include programs eldam, damage, matinv, principal, and principal1. The latter two use an eigenvalue solver subroutine, dsyev, available at the University of New Mexico.

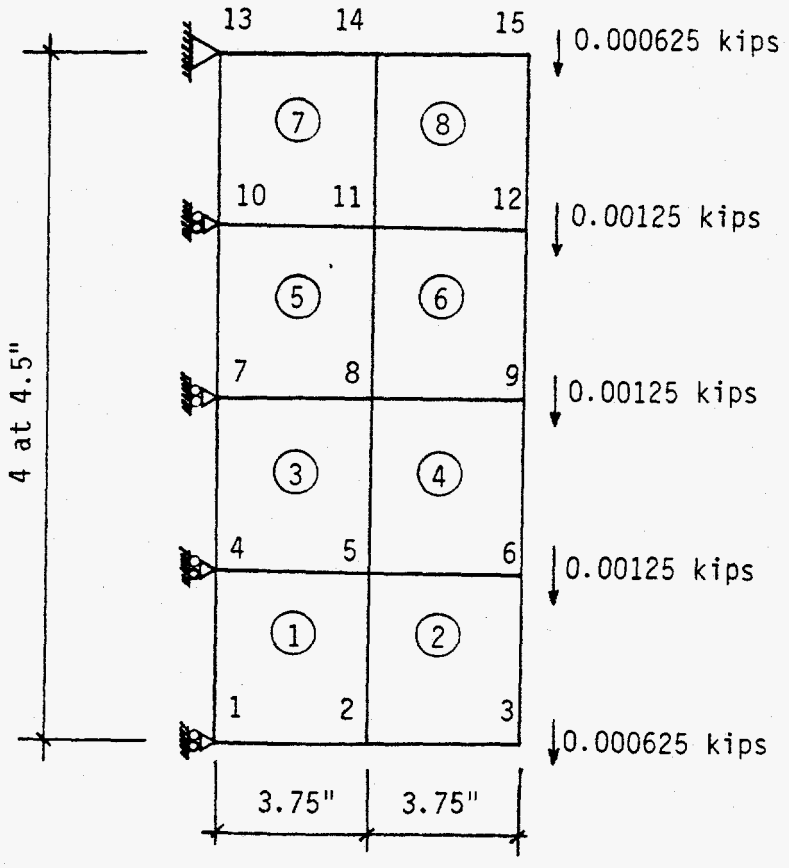
Flow chart of the program:

Program main	General control.
Call input	Reads in all data.
Call aeqnum	Assigns equation no. to the degrees of freedom.
Call band	Determines the band width of the stiffness matrix.
-[A]- Start the main do-loop:	
Call stif	Develops updated stiffness matrix.
Call stifQ4	Calculates the stiffness of a 4-noded element.
Call elast	Sets up the initial material stiffness.
Call nn4	Specifies local coordinates.
Call n4	Computes shape functions and derivatives.
Call jacob	Forms the Jacobian and its inverse.
Call stdisp	Sets up strain-displacement relationships.
Call mult	An algorithm for multiplication of matrices.
Call matadd	An algorithm for addition of matrices.
Call assmbl	Assembles the stiffness matrix.
Call principal1	Finds frequencies (normally first).
Call solver	Solves the system of equations.
Call disout	Obtains displacement increments.
Call recovr	Obtains strains from displacements.
Call nn4	
Call n4	
Call jacob	
Call stdisp	
Call mult	Increment of strains are avialable at this pt.
Call eldam	Sets up the data for the damage relations.
Call damage	Determines damage parameter and damage strain.
Call principal	Finds eigenvect. for transf. to global coord.
Call matinv	Obtains stiffness from flexibility.
Go to -[A]-	Repeat for new load increment.

A sample output for the shear wall shown on page 32:	Explanation
'Compression Member'	
15,8,1,2,20,.005	- No. of elements, nodes, analysis code, Guass integration points, no. of load increments, and the magnitude of the load increment.
1,1,2,5,4,0,0,0,0,4,100 2,2,3,6,5,0,0,0,0,4,100 3,4,5,8,7,0,0,0,0,4,100 4,5,6,9,8,0,0,0,0,4,100 5,7,8,11,10,0,0,0,0,4,100 6,8,9,12,11,0,0,0,0,4,100 7,10,11,14,13,0,0,0,0,4,100 8,11,12,15,14,0,0,0,0,4,100	- Element connectivity input: element no., nodes in that element specified in a counter clockwise direction (max. of 8 nodes per element. Here 4-noded quads are used. For higher order elements, zeros must be replaced by node nos.), material type, and material property number.
1,0.0,0.0,20 2,3.75,0.0,80 3,7.5,0.0,70 4,0.0,4.5,20 5,3.75,4.5,80 6,7.5,4.5,90 7,0.0,9.0,20 8,3.75,9.0,80 9,7.5,9.0,90 10,0.0,13.5,20 11,3.75,13.5,80 12,7.5,13.5,90 13,0.0,18,30 14,3.75,18,80 15,7.5,18,70	- Nodes and the corresponding coordinates: the node number, the X-coordinate, the Y-coordinate, and the label for the node based on the boundary condition of the node.
20,0.0,0.0,1,0 30,0.0,0.0,1,1 70,0.0,-.000625,0,0 80,0.0,0.0,0,0 90,0.0,-.00125,0,0	- Node label identification: label number, displacement or force in the X-direct., displacement or force in the Y-direct., 1 if X-displ. is specified 0 if X-force is specified, 1 if Y-displ. is specified 0 if Y-force is specified.
100,4000.0,0.20,1.0,1.0,1.0,1.0	- Properties for a particular element: element property number, Young's modulus, Poisson's ratio, and thicknesses at nodes.
0.9,1.0,15,1,.000303313	- model, mode2, the particular node whose deflection is computed, a switch to solve eigenvalue problems, total mass of the structure. Model and mode2 are the initial cracking parameters.
0.10e-05,.50,.0066964,0.00112,1.0	- initial arbitrary and small damage used to get started in the damage subroutine, uniaxial tensile strength, w, alpha, mu. The last three are referenced in the text.
4.,0,.0001,0.0,0.0	- Uniaxial compression strength, switch for hydrostatic compression, epsilon which is used as a measure of tolerance, reference stress and pressure.

2,1,1,1 - ipss.itest.iwl.ipl. See damage and main for the explanation.

A model shear wall corresponding to the input sample of page 31



```
c
      program main
c
c -----------------------------------------------------------------
c
c This Finite Element Program incorporates  non-linear damage constitutive
c  relations.
c  05/26/1988
c -----------------------------------------------------------------
c
c     implicit none
c
c
      double precision   coord(2,50)
c
c coord(1,n) = X_coordinate of node n
c coord(2,n) = Y_coordinate of node n
c
c
      double precision   bc(2,20)
c
c bc(1,s)   = X_force or displace. of set s.
c bc(2,s)   = Y_force or displace. of set s.
c
c
      integer   bccode(3,20)
c
c bccode(1,s) = 0  if X_force is specified.
c bccode(2,s) = 0  if Y_force is specified.
c bccode(1,s) = 1  if X_displacement is specified.
c bccode(2,s) = 1  if Y_sidplacement is specified.
c bccode(3,s) is the label for this set.
c
c
      integer   bcset(50)
c
c bcset(i) is the boundary condition set of this node
c
c
      integer           elconn(10,40)
c
c elconn(i,n) is the element connectivity array where:
c (9,n)  = element_type of element n
c (10,n) = mat_prop of element n
c (m,n)  = mth node in element n, m = 1,...,8
c
c
      integer     numnod
c
c numnod  =  number of nodes in structure
c
c
      double precision      displ(50,2)
c
```

```
c displ(50,2) is the displacement at nodes.
c
c
      integer   numelm
c
c numelm is the number of elements in structure.
c
c
      integer   numeq
c
c numeq is the number of equations to be solved.
c
c
      integer   code
c
c code --------   1 = plain stress
c                 2 = plane strain
c
c
      double precision      matpro(7,10)
c
c matpro(1,s) =  label for set
c matpro(2,s) =  Young's modulus
c matpro(3,s) =  Poisson's ratio
c matpro(4,s) =  thickness of material at node 1
c matpro(5,s) =  thichness of material at node 2
c matpro(6,s) =  thickness of material at node 3
c matpro(7,s) =  thickness of material at node 4
c
      double precision      d(6,6,40,4)
c
c d(6,6,40,4) is the updated elasticity matrix in the
c  principal directions.
c
      integer   eqnum(2,50)
c
c eqnum(d,n) is the equation number of node n, direction d
c  (d = 0,1 if node is fixed in this direction)
c
c
      double precision      A(200,200)
      double precision      B(200)
c
c A(200,200) is the [Kff] in semibanded form.
c B(200)      is the {Ff}-[Kfs]{ds} forcing vector.
c
      double precision delk,ft,w,alpha,mu,fc,epsl,sgl,pl
c
      double precision      displt(50,2)
      double precision      eigen1(100)
      double precision      load,xx
      double precision      deflection
c
c displt(n,i) is the total displacement of node n
```

```
c  in the direction of 1 = x-direction.
c                      2 = y-direction.
c load is the total force the structure is subjected after
c  n increment.
c xx is the total incremental load.
c Deflection is the deflection at the node of interest.
c
      integer   width
c
c width is the semibandwidth of [Kff].
c
c
      integer   Gauss
c
c Gauss is the order of integration.
c
c
      integer   nbcset
      integer   nmaset
      integer   nn,iw
      integer   ij,ii,y
      integer   ipss,itest,iwl,ipl,jp
      double precision mode1,mode2,mass
c
c nbcset is the num_bc_set.
c nmaset is the num_mat_sets.
c ij is the counter.
c ii is a particular node where the calculation
c  and plotting of the deflection is desired.
c y is an integer switch if y = 1 then solve eigenvalue problem,
c                      if y not equal to one, do not sove the
c                          eigenvalue problem.
c nn is the total number if increment that is specified.
c iwl determines at what interval of the loading increment
c  the eigenvalue problem should be solved.
c ipl is the controlling mechanism for plotting. Not used here.
c ipss,itest,jp are explained in the damage subroutine.
c
c
c open input and output files.
c
      open(unit=1,file='input.dat',status='old')
      open(unit=6,file='output.dat',status='old')
      open(unit=5,file='pluni',status='old')
      open(unit=7,file='pleig',status='old')
c
c input all data.
c
      call input(coord, bc, bccode, bcset, elconn, numnod,
     !           numelm, code, matpro, Gauss,
     !           nbcset, nmaset,
     !           nn,delk,ft,w,alpha,mu,fc,jp,epsl,sgl,
     !           pl,ipss,itest,iwl,ipl,xx,
     !           mode1,mode2,ii,y,mass)
```

```
c
c assigen equation numbers to free degrees of freedom.
c
      call aeqnum(bccode, bcset, numnod, eqnum, numeq,
     !            nbcset)
c
c calculate semibandwidth of [Kff].
c
      call band(elconn, eqnum, numelm, width)
c
      do 49 i=1,50
      do 48 j=1,2
      displt(i,j) = 0.0d0
  48  continue
  49  continue
      load = 0.0d0
      write(5,*) displt(4,1),load
      iw = 0
c
      do 100 ij = 1,nn
c
c nn is the number of load steps taken.
c
c
c calculate element stiffness matrices and write them to stiff.dat.
c
      call stif(coord, elconn, numelm, matpro, nmaset, code, d, ij)
c
c
c assemble the structure stiffness matrix [Kff] and the
c  corresponding forcing vector {{Ff} - [Kfs]{ds}}
c
      call assmbl(elconn, bccode, bcset, bc, eqnum, numeq,
     !            numelm, numnod, width, A, B,
     !            nbcset)
c
      if(y.eq.1) then
       iw = iw + 1
       if(iw .ne. iwl) go to 21
        call principal1(A,eigen1,numeq,mass,numelm)
  21   continue
      else
       continue
      endif
c
c Call semibanded equation solver:
c
       call solver(numeq,width,A,B)
c
c
c call subroutine to output displacements
c
      call disout(coord, bc, bcset, bccode, elconn,
     !            numnod, B, eqnum, output, numeq,
```

```
     !          nbcset, displ, displt,nn,ij)
c
c Call subroutine to compute and output strains and stresses.
c
      call recovr(elconn, bccode, bcset, bc, eqnum, numeq,
     !          numelm, numnod, coord, output,
     !          code, matpro, displ, nmaset, d,
     !          delk,ft,w,alpha,mu,fc,jp,epsl,sgl,
     !          pl,ipss,itest,iwl,ipl,ij,nn,
     !          mode1,mode2)
c
c
c
      load = load + xx
      deflection = dabs(displt(ii,2))
      write(5,*) deflection,load
      if(y.eq.1) then
       if(iw .ne. iwl) go to 22
       iw = 0
       write(7,*) load,dsqrt(dabs(eigen1(numeq)))
   22  continue
      else
      continue
      endif
  100 continue
c
       close(1)
       close(6)
       close(5)
       close(7)
c
      stop
      end
c
c
c ---------------------------------------------------------------------
c
      subroutine input(coord, bc, bccode, bcset, elconn, numnod,
     !          numelm, code, matpro, Gauss,
     !          nbcset, nmaset,
     !          nn,delk,ft,w,alpha,mu,fc,jp,epsl,sgl,
     !          pl,ipss,itest,iwl,ipl,xx,
     !          mode1,mode2,ii,y,mass)
c
c ---------------------------------------------------------------------
c
c This Subroutine reads the input data to the finite element
c  program from a file called 'input.dat'. The information
c  read is echoed to an output file called 'output.dat'.
c
c ---------------------------------------------------------------------
c
c     implicit none
c
```

```
c Variables not described here are explained in the main program.
c
      double precision   coord(2,50)
c
      double precision   bc(2,20)
c
      double precision   matpro(7,10)
c
      double precision   delk,ft,w,alpha,mu,fc,epsl,sgl,pl,xx
      double precision   mode1,mode2,mass
c
      integer   bccode(3,20)
c
      integer   bcset(50)
c
      integer   elconn(10,40)
c
      integer   numnod
c
      integer   numelm
c
      integer   code
c
      integer   Gauss
c
      integer   nn,ipss,itest,iwl,ipl,jp
c
      character*72  title
      integer   i,ii,y
      integer   j
      integer   nbcset
      integer   nmaset
      integer   node
      integer   elem
      integer   flag
c
c node is the node number.
c elem is the element number.
c
c -------------------------------------------------------
c
c read and write the title of the problem
c
      read(1,*) title
      write(6,1011)
      write(6,1000)
      write(6,*) title
c
c read and write some problem parameters.
c
      read(1,*) numnod, numelm, code, Gauss, nn, xx
      write(6,1011)
      write(6,1020) numnod, numelm, code, Gauss
c
```

```
c read and write element connectivity, type, and material number.
c
      do 10 i = 1, numelm
       read(1,*) elem,(elconn(j,elem), j = 1, 10)
   10 continue
      write(6,1011)
      write(6,1025)
      write(6,1010)
c
      do 15 i = 1, numelm
       write(6,1030) i, (elconn(j,i), j = 1, 10)
   15 continue
c
c determine how many material property sets there are:
c
      nmaset = 0
      do 25 elem = 1, numelm
      flag = 0
       do 20 i = 1, elem - 1
        if(elconn(10,elem) .eq. elconn(10,i)) then
         flag = 1
        else
         continue
        endif
   20  continue
      if(flag .eq. 0) nmaset = nmaset + 1
   25 continue
c
c read and write nodal coordinates and boundary condition sets.
c
      do 30 i = 1, numnod
       read(1,*) node, (coord(j,node), j = 1,2), bcset(node)
   30 continue
c
      write(6,1011)
      write(6,1029)
      write(6,1010)
      do 35 i = 1, numnod
       write(6,1040) i, (coord(j,i), j = 1,2), bcset(i)
   35 continue
c
c determine how many boundary condition sets there are:
c
      nbcset = 0
      do 45 node = 1, numnod
       flag = 0
        do 40 i = 1, node - 1
         if(bcset(node) .eq. bcset(i)) then
          flag = 1
         else
          continue
         endif
   40   continue
       if(flag .eq. 0) nbcset = nbcset + 1
```

```
   45 continue
c
c read and write none zero applied force or displacement
c  boundary conditions.
c
      do 50 i = 1, nbcset
       read(1,*) bccode(3,i),(bc(j,i),j=1,2),(bccode(j,i),
     !          j=1,2)
   50 continue
c
      write(6,1011)
      write(6,1045)
      write(6,1010)
      write(6,1036)
      write(6,1038)
      write(6,1039)
      write(6,1010)
      do 55 i = 1, nbcset
       write(6,1050)bccode(3,I),(bc(j,i),j=1,2),(bccode(j,i),
     !            j = 1,2)
   55 continue
c
c read and write material properties for each material type.
c
      do 60 i = 1, nmaset
       read(1,*)(matpro(j,i), j = 1,7)
       write(6,*)matpro(1,1),matpro(2,1),matpro(3,1),matpro(4,1)
   60 continue
c
      write(6,1011)
      write(6,1053)
      write(6,1010)
      write(6,1055)
      write(6,1010)
      do 65 i = 1,nmaset
       write(6,1060) idint(matpro(1,i)),(matpro(j,i),j=2,4)
   65 continue
c
      write(6,1011)
      write(6,1070)
      write(6,1010)
c
      read(1,*) mode1,mode2.ii,y,mass
      read(1,*) delk,ft,w,alpha.mu
      read(1,*) fc,jp,epsl,sgl,pl
      read(1,*) ipss,itest,iwl,ipl
      return
c
 1000 format('         OUTPUT OF PROGRAM MAIN              '/)
 1010 format(/)
 1011 format(//)
 1020 format('NUMBER OH NODAL POINTS        = ',I5/
     !       'NUMBER OF ELEMENTS            = ',I5/
     !       'ANALYSIS CODE                 = ',I5/
```

```
     !      'INTEGRATION ORDER              = ',I5)
 1025 format(' ELEMENT          NODAL CONNECTIVITY
     !TYPE   MATERIAL')
 1029 format(' NODES        COORDINATES              BC SETS')
 1030 format(I5,'      ',8I5,'     ',2I5)
 1036 format('                                  | 0 = SPECIFIED
     !FORCE |')
 1038 format(' SET      (FORCE OR DISPLACEMENT)      | 1 = SPECIFIED
     !DISPLACEMENT')
 1039 format('          X       Y                  | X-CODE |
     ! | Y_CODE |')
 1040 format(I5,2f12.4,'     ',I15)
 1045 format('    NONE ZERO PRESCRIBED NODAL BOUNDARY VALUE SETS')
 1050 format(I5,2f12.5,'     ',2I10)
 1053 format('          Material Property Set')
 1055 format('LABEL     YOUNG MODULUS          POISSONS RATIO,
     !THICKNESS')
 1060 format(I5,'     ',d17.2,'     ',d7.2,'     ',d25.4)
 1070 format('---------- END OF INPUT PHASE ----------')
c
      end
c
c -----------------------------------------------------------------
c
      subroutine stif(coord,elconn,numelm, matpro, nmaset, code, d, ij)
c
c This subroutine calculates each element stiffness matrix and
c  stores them in stiff.dat.
c
c     implicit none
c
      double precision      d(6.6,40,4)
      double precision      coord(2,50)
      double precision      matpro(7,10)
      double precision      stiff(8,8)
      double precision      lcoord(4,2)
      double precision      thick(4)
c     double precision      x(100)
c     double precision      y(100)
      double precision      ym
      double precision      nu
c
c
c stiff(8,8) is the element stiffness matrix
c lcoord(n,i) are the global coordinates of nodes in the
c          element under consideration.
c thick(4) are thicknesses at nodes.
c x(100) are the x values.
c y(100) are the y values.
c ym is the Young's modulus.
c nu is the Poisson's ratio.
c
      integer   elconn(10,40)
      integer   numelm
```

```
      integer   nmaset
      integer   code
      integer   elem
      integer   i
      integer   node
      integer   ij
c
c elem is the element number.
c i is the counter.
c node is the node number.
c
c
c Open a file for stiff.dat
c
      open(unit=2,file='stiff.dat',status='old')
c
c Compute the element stiffness for each element:
c
c start the main do loop
c
      do 30 elem = 1, numelm
c
c Find the material properties for the elements:
c
      do 10 i = 1, nmaset
       if(elconn(10,elem) .eq. matpro(1,i)) then
        ym = matpro(2,i)
        nu= matpro(3,i)
        thick(1) = matpro(4,i)
        thick(2) = matpro(5,i)
        thick(3) = matpro(6,i)
        thick(4) = matpro(7,i)
       else
        continue
       endif
   10 continue
c
c transfer element coordinate to global
c
      do 20 node = 1, elconn(9,elem)
       lcoord(node,1) = coord(1,elconn(node,elem))
       lcoord(node,2) = coord(2,elconn(node,elem))
   20 continue
c
c Call and write stifQ4 to stiff.dat
c
      call stifQ4(lcoord, thick, ym, nu, code, stiff, elem, d, ij)
c
      do 21 i=1,8
      write(2,*) (stiff(i,j),j=1,8)
   21 continue
c
   30 continue
c
```

```
      close(2)
c
      return
      end
c
c -----------------------------------------------------------------
c
      subroutine stifQ4(lcoord, thick, ym, nu, code, stiff, elem, d, ij)
c
c This subroutine calculates the stiffness matrix of a four noded
c  element.
c
c     implicit none
c
      double precision    d(6,6,40,4)
      double precision    lcoord(4,2)
      double precision    stiff(8,8)
      double precision    BT(8,3)
      double precision    thick(4)
      double precision    ym
      double precision    nu
      double precision    csi,eta
      integer             i,j
      integer             elem,kk
      integer             code
      integer             ij
c
c The expanations for some of the variables are as follow:
c
c BT is the transpose of the Strain-displacement matrix BSD.
c csi and eta are values for a given point of Gauss point.
c i,j,k are do loops counters
c
      double precision    ds(8,8)
      double precision    C(3,3)
      double precision    CB(3,8)
      double precision    BSD(3,8)
      double precision    jinv(2,2)
      double precision    sf(4)
      double precision    ncsi(4)
      double precision    neta(4)
c     double precision    xcoord(4)
c     double precision    ycoord(4)
      double precision    detj
      double precision    t
c
c The explanations for some of the variables are as follow:
c
c ds(8,8) is the product [BT][C][B]
c C is the elasticity matrix
c BSD is the strain displacement matrix
c q(8,8) is the product in the multiplication matrix
c d(8,8) is the sum of 2 #'s in matadd
c jinv is the inverse of the Jacobian matrix
```

```
c sf(1) are the shape functions at nodes
c ncsi(4) are the partial of n wrt csi
c neta(4) are the partial of n wrt eta
c xcoord are the x coordinates of nodes
c ycoord are the y coordinates of nodes
c detj is determinant of Jacobian
c t is the thickness times sf.
c
c
c Initialize stiff(i,j)
c
      do 20 i = 1, 8
      do 10 j = 1, 8
       stiff(i,j) = 0.0
   10 continue
   20 continue
c
c
c Call the elsticity matrix C
c
c
      do 90 kk=1,4
c
      call elast(ym, nu, code, C, ij, d, elem, kk)
c
c Use 2 Gaussian points
c Evaluate everything at 4 points
c For n=2, csi and eta are both .57735
c Construct a do loop for each point
c
       call nn4(csi,eta,kk)
       call n4(sf,ncsi,neta,csi,eta)
       call jacob(lcoord,ncsi,neta,jinv,detj)
c
c Calculate the thickness
c
       t = 0.0
       do 40 i = 1,4
        t = t + sf(i)*thick(i)
   40  continue
c
       call stdisp(BSD,ncsi,neta,jinv,sf,code)
c
c Calcluate B transpose
c
       do 60 i = 1,8
       do 50 j = 1,3
        BT(i,j) = BSD(j,i)
   50  continue
   60  continue
c
c Multiply BT times C times B
c
       call mult(C,BSD,3,3,8,CB)
```

```
      call mult(BT,CB,8,3,8,ds)
c
c Multiply ds times thickness times w.f
c  t(1) = [sf(1)  sf(2)  sf(3)  sf(4)]*{thick(1)}
c                                      {thick(2)}
c                                      {thick(3)}
c                                      {thick(4)}
c
c
c For the case of 2 Gauss points w.f.(weithing function) is 1.0
c Multiply ds(i,j) times t times w.f.(1)
c
      do 80 i = 1, 8
      do 70 j = 1, 8
       ds(i,j) = t*ds(i,j)*detj
   70 continue
   80 continue
c
c Add ds(i,j) into stiff(i,j)
c
      call matadd(ds,stiff,8,8,stiff)
c
   90 continue
c
      return
      end
c ---------------------------------------------------------------------
c
      subroutine elast(ym,nu,code,C, ij, d, elem, kk)
c
c This subroutine sets up the initial stiffness matrix.
c
      double precision    d(6,6,40,4)
      double precision    C(3,3)
      double precision    ym
      double precision    nu
      double precision    con
      double precision    cons
c
      integer     code
      integer     i,j,elem,kk,ij
c
c Some of the variables are explained as follows:
c
c C(4,4) is the elasticity matrix.
c con,cons, and const are some common values that are
c          identified in the elasticity matrices.
c i,j,k are do loop counters.
c
      if(ij. gt. 1) then
       go to 100
      else
       continue
      endif
```

```
c
c Initialize Elasticity Matrix:
c
      do 20 i=1,3
      do 10 j=1,3
       C(i,j) = 0.0d0
   10 continue
   20 continue
c
c
c Define the elasticity matrix for the plain strain problems:
c
      if(code .eq. 2) then
       con = ym/(1.0d0-nu**2.0d0)
       C(1,1) = con
       C(1,2) = con * nu
       C(2,1) = C(1,2)
       C(2,2) = con
       C(3,3) = con * (1.0d0-nu)/2.0d0
      else
       continue
      endif
c
c Define the elasticity matrix for the plain stress problem:
c
      if(code .eq. 1) then
       cons = (ym*(1.0d0-nu))/((1.0d0+nu)*(1.0d0-2.0d0*nu))
       C(1,1) = cons
       C(1,2) = cons*nu/(1.0d0-nu)
       C(2,1) = C(1,2)
       C(2,2) = cons
       C(3,3) = cons*(1.0d0-2.0d0*nu)/(2.0d0*(1.0d0-nu))
      else
       continue
      endif
      go to 200
c
  100 continue
c
      C(1,1) = d(1,1,elem,kk)
      C(1,2) = d(1,2,elem,kk)
      C(2,1) = C(1,2)
      C(2,2) = d(2,2,elem,kk)
c
  200 continue
c
      return
      end
c
c -----------------------------------------------------------------
c
      subroutine n4(sf,ncsi,neta,csi,eta)
c
c This subroutine calculates the shape functions and the derivatives.
```

```
c
      double precision    sf(4)
      double precision    ncsi(4)
      double precision    neta(4)
      double precision    csi
      double precision    eta
      double precision    a,b,c,d
c
c sf(4) designate values of the shape function at each point.
c ncsi(4) are the partial derivative of shape functions W.R.T. csi.
c neta(4) are the partial derivative of shape functions W.R.T. eta.
c csi and eta are the local coordinates of the elements.
c a, b, c, and d are for the ease of calculations.
c
c Define a, b, c, and d:
c
      a = (1.0d0 + csi)
      b = (1.0d0 - csi)
      c = (1.0d0 + eta)
      d = (1.0d0 - eta)
c
      sf(1) = .25d0 * b * d
      sf(2) = .25d0 * a * d
      sf(3) = .25d0 * a * c
      sf(4) = .25d0 * b * c
c
c Calculate the Partials of Shape Functions W.R.T. csi:
c
      ncsi(1) = -.25d0 * d
      ncsi(2) = +.25d0 * d
      ncsi(3) = +.25d0 * c
      ncsi(4) = -.25d0 * c
c
c Calculate the Partials of the Shape Functions W.R.T. eta:
c
      neta(1) = -.25d0 * b
      neta(2) = -.25d0 * a
      neta(3) = +.25d0 * a
      neta(4) = +.25d0 * b
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
c
      subroutine nn4(csi,eta,kk)
c
c This subroutine specifies local coordinates.
c
      double precision csi
      double precision eta
      integer kk
```

```
c
      if(kk.eq.1) then
       csi = -.57735
       eta = csi
      else
       continue
      endif
      if(kk.eq.2) then
       csi = .57735
       eta = -.57735
      else
       continue
      endif
c
      if(kk.eq.3) then
       csi = .57735
       eta = csi
      else
       continue
      endif
c
      if(kk.eq.4) then
       csi = -.57735
       eta = +.57735
      else
       continue
      endif
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
      subroutine jacob(lcoord,ncsi,neta,jinv,detj)
c
c This subroutine forms the Jacobian of transformation and
c  its inverse.
c
      double precision    lcoord(4,2)
      double precision    j(2,2)
      double precision    jinv(2,2)
      double precision    ncsi(4)
      double precision    neta(4)
      double precision    detj
c
      integer     i,k
c
c j(2,2) is the Jocobian matrix.
c jinv(2,2) is the inverse of the Jacobian.
c detj is the determinant of the Jacobian matrix.
c n is the number of nodes.
c
c Initialize the Jacobian and its inverse Matrices:
```

```
c
      do 20 i=1,2
      do 10 k=1,2
       j(i,k) = 0.0d0
       jinv(i,k) = 0.0d0
   10 continue
   20 continue
c
c Multiply the matrix of shape function, [sf], by the coordinates
c   of the nodes.
c
      do 30 i = 1,4
       j(1,1) = j(1,1) + ncsi(i) * lcoord(i,1)
       j(1,2) = j(1,2) + ncsi(i) * lcoord(i,2)
       j(2,1) = j(2,1) + neta(i) * lcoord(i,1)
       j(2,2) = j(2,2) + neta(i) * lcoord(i,2)
   30 continue
c
c Calculate the determinant of the Jacobian:
c
      detj = j(1,1)*j(2,2) - j(2,1)*j(1,2)
c
c Calculate the inverse of j(2,2):
c
      jinv(1,1) = j(2,2)/detj
      jinv(1,2) = -j(1,2)/detj
      jinv(2,1) = -j(2,1)/detj
      jinv(2,2) = j(1,1)/detj
c
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
      subroutine stdisp(BSD,ncsi,neta,jinv,sf,code)
c
c Strain-displacement relations are formed in this subroutine.
c
      double precision    BSD(3,8)
      double precision    jinv(2,2)
      double precision    ncsi(4)
      double precision    neta(4)
      double precision    sf(4)
      double precision    q(4)
      double precision    p(4)
c
      integer     code
      integer     i,l
c
c BSD(3,8) is the Strain-Displacement Matrix.
c q(4) is the partial of [sf] w.r.t. x.
c p(4) is the partial of [sf] w.r.t. y
```

```
c
c
c Initialize the B matrix:
c
      do 20 i=1,3
      do 10 l=1,8
       BSD(i,l) = 0.0d0
   10 continue
   20 continue
c
      do 25 i=1,4
       q(i)   = 0.0d0
       p(i)   = 0.0d0
   25 continue
c
c Generate the B matrix:
c
      do 30 i=1,4
       q(i) = ncsi(i)*jinv(1,1) + neta(i)*jinv(1,2)
       p(i) = ncsi(i)*jinv(2,1) + neta(i)*jinv(2,2)
   30 continue
c
      BSD(1,1) = q(1)
      BSD(1,3) = q(2)
      BSD(1,5) = q(3)
      BSD(1,7) = q(4)
      BSD(2,2) = p(1)
      BSD(2,4) = p(2)
      BSD(2,6) = p(3)
      BSD(2,8) = p(4)
      BSD(3,1) = p(1)
      BSD(3,2) = q(1)
      BSD(3,3) = p(2)
      BSD(3,4) = q(2)
      BSD(3,5) = p(3)
      BSD(3,6) = q(3)
      BSD(3,7) = p(4)
      BSD(3,8) = q(4)
c
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
      subroutine mult(A,B,NRA,NCA,NCB,CB)
c
c This is a multiplication algorithm for matrices.
c
      integer    NRA
      integer    NCA
      integer    NCB
      integer    i,j,k
```

```
c
      double precision    A(NRA,NCA)
      double precision    B(NCA,NCB)
      double precision    CB(NRA,NCB)
c
c
c CB(NRA,NCB) is the product of [A] and [B].
c NRA is the number of rows is A.
c NCA is the number of Columns in A.
c NCB is the number of columns in B
c i,j,k are the do loop variables.
c
c Initialize the [CB] matrix:
c
      do 20 i=1,NRA
      do 10 j=1,NCB
       CB(i,j) = 0.0d0
   10 continue
   20 continue
c
c Multiply [A] times [B]:
c
      do 50 i=1,NRA
      do 40 j=1,NCB
      do 30 k=1,NCA
       CB(i,j) = CB(i,j) + A(i,k)*B(k,j)
   30 continue
   40 continue
   50 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
      subroutine matadd(A,B,NRA,NCA,D)
c
c This is an algorithm for addition of matrices.
      integer     NRA
      integer     NCA
      integer     i,j
c
      double precision    A(NRA,NCA)
      double precision    B(NRA,NCA)
      double precision    D(NRA,NCA)
c
c NRA is the number of rows in the matrix A.
c NCA id the number of cloumns in the matrix A.
c i,j are the variables for do-loops.
c
c Add A to B:
c
      do 20 i = 1, NRA
```

```
      do 10 j = 1, NCA
        D(i,j) = A(i,j) + B(i,j)
   10 continue
   20 continue
c
      return
      end
c
c -----------------------------------------------------------------
c
c234567
      subroutine aeqnum(bccode,bcset,numnod,eqnum,numeq,nbcset)
c
c This subroutine assigns a number for every equation at every
c  degree of freedom.
c
      integer    bccode(3,20)
      integer    eqnum(2,50)
      integer    bcset(50)
      integer    numnod
      integer    numeq
      integer    nbcset
      integer    nodebc
      integer    label
      integer    idir
      integer    inode
      integer    i
c
c nodebc is the number of 1 node for that boundary condition.
c label is used to compare numbers.
c idir is the ith direction.
c inode is the ith node.
c i is the counter.
c
c Initialize numeq:
c
      numeq = 1
c
c Search through each node and assign a value to label:
c
      do 30 inode = 1, numnod
       label = bcset(inode)
c
c Now find the B.C. set that is equal to label:
c
        do 10 i = 1,nbcset
        if(bccode(3,i) .eq. label) then
         nodebc = i
        else
         continue
        endif
   10   continue
c
c Search both the X and Y directions:
```

```
c
      do 20 idir = 1,2
       if(bccode(idir,nodebc) .eq. 0) then
        eqnum(idir,inode) = numeq
        numeq = numeq + 1
       else
        eqnum(idir,inode) = 0
       endif
  20   continue
  30 continue
c234567
     numeq = numeq - 1
c
     return
     end
c
c ---------------------------------------------------------------------
c
c234567
     subroutine band(elconn, eqnum, numelm, width)
c
c This subroutine finds the semi-bandwith of [Kff].
c
     integer    elconn(10,40)
     integer    eqnum(2,50)
     integer    numelm
     integer    width
     integer    eqelma
     integer    eqelmi
     integer    elmdif
     integer    maeldi
     integer    nelnod
     integer    elem
     integer    idir
     integer    inode
c
c eqelma is the element's maximum equation number.
c eqelmi is the elemnet's minimum equation number.
c elmdif is the difference between 2 nodes.
c maeldi is the maximum differnece of two equations in an element.
c nelnod is the number of element nodes.
c
c Initialize variables to zero:
c
     elmdif = 0
     maeldi = 0
     eqelma = 0
c
c Make eqelmi (EQ_ELEM_MIN) larger than your greatest number of
c  equations(2000).
c
c Search each element:
c
     do 30 elem = 1, numelm
```

```
      nelnod = elconn(9,elem)
      eqelmi = 3000
c
c Search for each node:
c
      do 20 inode = 1, nelnod
c
c Search in the X and Y directions and find the maximum and
c  minimum element equations:
c
      do 10 idir = 1 , 2
       eqelma = max(eqnum(idir,elconn(inode,elem)),eqelma)
       eqelmi = min(eqnum(idir,elconn(inode,elem)),eqelmi)
   10 continue
c
   20 continue
c
c Find the maximum difference:
c
      elmdif = eqelma - eqelmi
c
c Find the maximum element difference:
c
      maeldi = max(maeldi,elmdif)
c
   30 continue
c
c Find the semi_bandwidth:
c
      width = maeldi + 1
c
      return
      end
c
c -----------------------------------------------------------------------
c
c234567
      subroutine assmbl(elconn,bccode,bcset,bc,eqnum,numeq,
     *                  numelm,numnod,width,A,B,nbcset)
c
c This subroutine assembles the stiffness matrix.
c
c
      integer     elconn(10,40)
      integer     bccode(3,20)
      integer     eqnum(2,50)
      integer     bcset(50)
      integer     numnod
      integer     numelm
      integer     numeq
      integer     width
      integer     node,elem,n1,n2
      integer     node1,node2,dir,dr
      integer     row,col,i
```

```
      integer    bcflag
      integer    bcnum
      integer    nbcset
c
      double precision    A(200,200)
      double precision    B(200)
      double precision    stiff(8,8)
      double precision    bc(2,20)
      double precision    spdisp
      double precision    spforc
c
c A is the stiffness matrix Kff.
c B is the forcing vector.
c Stiff is the stiffness matrix.
c spdisp is the specified displacement.
c spforc is the specified force.
c
c Open file:
c
      open(unit=2,file='stiff.dat',status='old')
c
c Initialize the matrix A:
c
      do 1 i=1,numeq
       do 1 j=1,numeq
        A(i,j) = 0.0d00
        B(i)   = 0.0d00
    1 continue
c
      row = 0
      col = 0
c diag
      do 60 elem = 1 , numelm
       do 5 i=1,8
        read(2,*)(stiff(i,j),j=1,8)
    5  continue
       do 50 n1=1,elconn(9,elem)
        node1=elconn(n1,elem)
c      write(6,*)'we are in subroutine assemble line #1224'
c      write(6,*)'ij=',ij,'kk=',kk
        do 40 dir = 1,2
         row = eqnum(dir,node1)
         do 30 n2 = 1,elconn(9,elem)
          node2 = elconn(n2,elem)
          do 20 dr = 1,2
           col = eqnum(dr,node2)
           if(row.gt.0) then
            if(col.gt.0) then
             if((col-row+1).gt.0) then
              A(row,(col-row+1)) = A(row,(col-row+1)) + stiff(((n1-1)
     *                          *2 + dir),((n2-1)*2 + dr))
             else
              continue
             endif
```

```
          else
           bcflag = bcset(node2)
           do 10 i = 1,nbcset
            if(bccode(3,i) .eq. bcflag) then
             bcnum = i
            else
             continue
            endif
   10       continue
c
c Calculate {Ff}-{Ffs}*{delts}:
c
           spdisp = bc(dr,bcnum)
           B(row) = B(row) - stiff((n1-1)*2 + dir,(n2-1)*2 +
     *                              dr)*spdisp
          endif
         else
          continue
         endif
   20    continue
   30   continue
   40  continue
   50 continue
   60 continue
c
c
      do 90 node = 1,numnod
       do 80 dir = 1,2
        if(eqnum(dir,node).gt.0) then
         bcflag = bcset(node)
         do 70 i = 1, nbcset
          if(bccode(3,i).eq.bcflag)then
           bcnum = i
          else
           continue
          endif
   70    continue
         spforc = bc(dir,bcnum)
         B(eqnum(dir,node)) = B(eqnum(dir,node))+spforc
        else
         continue
        endif
   80  continue
   90 continue
c
      close(2)
      return
      end
c
c ---------------------------------------------------------------
c
c234567
      subroutine disout(coord,bc,bcset,bccode,elconn,numnod,
     *                  B,eqnum,output,numeq,nbcset,displ,displt,nn,ij)
```

```
c
c This subroutine calculates the incremental displacements.
c
      integer     elconn(10,40)
      integer     eqnum(2,50)
      integer     bccode(3,20)
      integer     bcset(50)
      integer     numeq
      integer     numnod
      integer     output
      integer     idir,inode,i
      integer     label,check
      integer     nbcset
      integer     nn,ij
c
      double precision     coord(2,50)
      double precision     bc(2,20)
      double precision     displ(50,2)
      double precision     displt(50,2)
      double precision     B(numeq)
      double precision     D(2)
c
c displ is the nodal displacement matrix.
c B is the displacement vector.
c D is a local variable for writing the displacements.
c output is the output number.
c label and check are programming flags.
c
c
      if(ij.eq.nn)then
      write(6,177)
  177 format(' G_NODE',2x,'X DISP',6x,'Y DISP')
      else
      continue
      endif
c
      do 30 inode = 1,numnod
       do 20 idir = 1,2
        if(eqnum(idir,inode).eq.0) then
         label = bcset(inode)
         do 10 i=1,nbcset
          if(label.eq.bccode(3,i))then
           check = i
           D(idir) = bc(idir,check)
          else
           continue
          endif
   10    continue
        else
         D(idir) = B(eqnum(idir,inode))
        endif
       displ(inode,idir) = D(idir)
       displt(inode,idir) = displt(inode,idir)+displ(inode,idir)
   20  continue
```

```
      if(ij.eq.nn)write(6,100) inode,(D(idir),idir=1,2)
   30 continue
c
  100 format(i5,2e12.4)
c
      return
      end
c
c -----------------------------------------------------------------
c
c234567
      subroutine recovr(elconn,bccode,bcset,bc,eqnum,numeq,
     !                  numelm,numnod,coord,output,
     !                  code,matpro,displ,nmaset,d,
     !              delk,ft,w,alpha,mu,fc,jp,epsl,sgl,
     !              pl,ipss,itest,iwl,ipl,ij,nn,
     !              mode1,mode2)
c
c This subroutine computes strains from displacements.
c
      integer     elconn(10,40)
      integer     bccode(3,20)
      integer     bc(2,20)
      integer     eqnum(2,50)
      integer     bcset(50)
      integer     numeq
      integer     numelm
      integer     numnod
      integer     code
      integer     elem
      integer     node
      integer     nmaset
      integer     gnode
      integer     i,kk
      integer     ipss,itest,iwl,ipl,ij,nn
c
      double precision     d(6,6,40,4)
      double precision     nstrain(40,50,3)
      double precision     gstrain(40,4,3)
      double precision     coord(2,50)
      double precision     matpro(7,10)
      double precision     displ(50,2)
      double precision     lcoord(4,2)
      double precision     BSD(3,8)
      double precision     strain(3,1)
      double precision     jinv(2,2)
      double precision     delta(8)
      double precision     sf(4)
      double precision     ncsi(4)
      double precision     neta(4)
      double precision     z(12)
      double precision     csi
      double precision     eta
      double precision     detj
```

```
      double precision      ym
      double precision      nu
      double precision      delk,ft,w,alpha,mu,fc,jp,epsl,sgl,pl
      double precision      mode1,mode2
c
c
c
      if(ij.eq.1)then
      write(6,123)
  123 format(//)
      write(6,124)
  124 format('  ELEM',3x,'G_NODE',3x,'L_NODE',3x,'STRAIN X',
     *       5x,'STRAIN Y ',4x,'STRAIN XY',4x,'STRAIN 0')
      else
      continue
      endif
c
c234567
c
      do 50 elem = 1,numelm
       delta(1) = displ(elconn(1,elem),1)
       delta(2) = displ(elconn(1,elem),2)
       delta(3) = displ(elconn(2,elem),1)
       delta(4) = displ(elconn(2,elem),2)
       delta(5) = displ(elconn(3,elem),1)
       delta(6) = displ(elconn(3,elem),2)
       delta(7) = displ(elconn(4,elem),1)
       delta(8) = displ(elconn(4,elem),2)
c
      do 10 node = 1, elconn(9,elem)
        lcoord(node,1) = coord(1,elconn(node,elem))
        lcoord(node,2) = coord(2,elconn(node,elem))
   10 continue
c
      do 20 i = 1, nmaset
       if(elconn(10,elem) .eq. matpro(1,i)) then
        ym = matpro(2,i)
        nu= matpro(3,i)
       else
        continue
       endif
   20 continue
c
      do 40 node = 1, 4
      if(node.eq.1) then
       csi = -1.0d0
       eta = -1.0d0
      else
       continue
      endif
      if(node.eq.2) then
       csi = 1.0d0
       eta = -1.0d0
      else
```

```
      continue
     endif
     if(node.eq.3) then
      csi = 1.0d0
      eta = 1.0d0
     else
      continue
     endif
     if(node.eq.4) then
      csi = -1.0
      eta = 1.0
     else
      continue
     endif
c234567
     call n4(sf,ncsi,neta,csi,eta)
     call jacob(lcoord,ncsi,neta,jinv,detj)
     call stdisp(BSD,ncsi,neta,jinv,sf,code)
c
c Multiply B times delta to get strain:
c
c diag
     call mult(BSD,delta,3,8,1,strain)
c
     gnode = elconn(node,elem)
c
     do 25 i=1,3
     nstrain(elem,gnode,i) = strain(i,1)
  25 continue
c
c Strain(3,1) is the strain matrix Ex,Ey,Exy,E0.
c
     if(ij.eq.nn) then
     write(6,155) elem,node,elconn(node,elem),(strain(i,1),i=1,3)
     else
      continue
     endif
c
  155 format(i5,i7,i9,5x,4e12.4)
c
   40 continue
c
     do 41 node = 1,4
     gnode = elconn(node,elem)
     do 41 i=1,3
     if(node.eq.1) z(i)   = nstrain(elem,gnode,i)
     if(node.eq.2) z(i+3) = nstrain(elem,gnode,i)
     if(node.eq.3) z(i+6) = nstrain(elem,gnode,i)
     if(node.eq.4) z(i+9) = nstrain(elem,gnode,i)
   41 continue
c
     do 43 kk=1,4
     call nn4(csi,eta,kk)
     call n4(sf,ncsi,neta,csi,eta)
```

```
      do 42 i=1,3
       gstrain(elem,kk,i) = sf(1)*z(i)+sf(2)*z(i+3)+sf(3)*z(i+6)
     !                    +sf(4)*z(i+9)
   42 continue
c
c Call subroutine for damage.
c
      call eldam(gstrain,ym,nu,delk,ft,w,alpha,mu,fc,jp,epsl,
     !           sgl,pl,ipss,itest,iwl,ipl,ij,elem,kk,d,nn,
     !           mode1,mode2)
c
   43 continue
c
   50 continue
c
  150 format(i5,i7,i9,5x4e12.4)
c  60 continue
c  70 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
c234567
      subroutine solver(numeq,mband,A,B)
c
c subroutine solver solves a symmetric system of banded
c simulataneous linear equations using the method of
c Banachiewicz (Cholesky decomposition for symmetric
c matrices). The coefficient matrix [A] is a distorted
c array with the diagonal elements of the global
c stiffness matrix [K] stored in column 1 of [A].
c
      integer     numeq
      integer     mband
      double precision    A(200,200)
      double precision    B(200)
      double precision    diag
      double precision    air
      double precision    sum
c
      integer     i
      integer     ii
      integer     j
      integer     jj
c     integer     k
      integer     l
      integer     m
      integer     n
      integer     n1
      integer     nhw
      integer     itrig
      integer     nred
```

```
      integer     lim
c
c
c     do 1 i=1,21
c      write(6,*)'B(',i,')=',B(i)
c   1 continue
      n = numeq
      nhw = mband - 1
      itrig = 0
       nred = 0
       lim = mband
   10 if(nred + 1 -n) 20,100,100
   20 nred = nred + 1
      diag =  A(nred,1)
c
c Test for a Zero or negative diagonal element
c
c     if(diag.gt.1.0d0E-10) then
      if((diag-1.0d-30).gt.0.0d0) then
       diag = sqrt(diag)
      else
       go to 90
      endif
c
c Drive row by square root of Diagonal element
c
      do 40 j=1,lim
   40 A(nred,j)=A(nred,j)/diag
c
c     reduce remaining block of numbers
c
      do 80 i = 1,nhw
       l=nred + i
        if(l-n) 50,50,80
   50 air = A(nred,i+1)
c
c Skip this row if the multiplier air is zero.
c
      if(air) 60,80,60
   60 do 70 j=i,nhw
       m=1+j-i
   70   A(l,m)=A(l,m)-air*A(nred,j+1)
   80  continue
      go to 10
c
   90 itrig = nred
c 100 continue
  100 if(itrig) 110,120,110
c
c Matrix is singular or not positive definite
c
  110 write(6,200) itrig
c     stop
      go to 190
```

```
  120 continue
c
c
c Reduce the right hand side:
c
c
      nred = 0
  130 if(nred+1-n) 140,170,170
  140 nred = nred + 1
c
c divide row by square root of diagonal element
c reduce the remaining block of numbers
c
      B(nred) = B(nred)/A(nred,1)
      do 160 i = 1, nhw
       l = nred + i
        if(l-n) 150,150,160
  150    B(l) = B(l) - A(nred,i+1)*B(nred)
  160     continue
          go to 130
  170 B(n) = B(n)/A(n,1)
      n1 = n - 1
      do 190 ii = 1,n1
      i = n - ii
      sum = 0.0d0
      do 180 jj=1,nhw
      m = jj + i
      if(n-m) 190,180,180
  180  sum = sum + A(i,jj+1)*B(m)
      B(i) = (B(i) - sum)/A(i,1)
  190  continue
c
      return
c
  200 format('singular matrix at euation number',i4/
     1       '  posibble causes of the singularity are:'//
     2       '1. sturcture is underconstrained permitting'/
     3       '  rigid body motion.'/
     4       '2. Material properties are improperly definrd.'/
     5       '3. The Jacobian matrix is not posotive'/
     6       '  definite. Check the order inwhich the '/
     7       '  element nodes are specified.'/
     8       '4. Last but not leat check your input data'/
     9       '  to make sure that all input quatities'/
     1       '  are specified properly')
c
c
      end
c
c ---------------------------------------------------------------------
      subroutine eldam(gstrain,ym,nu,delk,ft,w,alpha,mu,fc,jp,epsl,
     !                 sgl,pl,ipss,itest,iwl,ipl,ij,
     !                 elem,kk,d,nn,
     !                 mode1,mode2)
```

```
c
      double precision F(6,6,40,4),cc(6,6,40,4),cc1(6,6,40,4),
     !                 cc2(6,6,40,4),d(6,6,40,4)
      double precision sg11(40,4),sg22(40,4),
     !                sg33(40,4),sg12(40,4),p,sgl,pl
      double precision gstrain(40,4,3),stt11,stt22,stt33,stt12,dstt11,
     !                 dstt22,dstt33,dstt12,ste11,ste22,ste33,ste12,
     !                 sttvol
      double precision k(40,4),t(40,4),ft,delk,epsl
      double precision std11,std22,std33,std12
      double precision ci(40,4),fc,w,alpha,mu,ym,nu
      double precision e1,e2,tg,pr1,pr2,pr3,pr4
      double precision mode1,mode2,mean
      double precision vector(3,3),eigen(3)
      integer          iswich,jp,iwl,ipl
      integer          ipss,itest,iter,ij,elem,kk
      integer          i,j,nn
      open(unit=3,file='output',status='old')
c -------------------------------------------------------------
c
c This subroutine sets the data for the damage relations.
c The part from here to 10 is used to initialize variables and
c  elastic matrices. Normally one goes directly to 10.
c
      if(ij.gt.1) go to 10
      iter  = 0
      iswich= 0
      stt11 = 0.d0
      stt22 = 0.d0
      stt33 = 0.d0
      stt12 = 0.d0
      sttvol= 0.d0
      std11 = 0.d0
      std22 = 0.d0
      std33 = 0.d0
      std12 = 0.d0
      ste11 = 0.d0
      ste22 = 0.d0
      ste33 = 0.d0
      ste12 = 0.d0
      p     = 0.d0
      pl    = 0.d0
      t(elem,kk) = 0.d0
      k(elem,kk) = 0.d0
      ci(elem,kk)= 0.d0
      sg11(elem,kk)  = 0.d0
      sg22(elem,kk)  = 0.d0
      sg33(elem,kk)  = 0.d0
      sg12(elem,kk)  = 0.d0
      do 150 i=1,6
      do 140 j=1,6
      F(i,j,elem,kk) = 0.
      cc(i,j,elem,kk)= 0.
      d(i,j,elem,kk) = 0.
```

```
 110 continue
 150 continue
c iter is the iteration count.
c stt11, stt22, stt33 are the total strains in 1-1, 2-2, and 3-3 directions.
c std11, std22, std33 are the total damage strains in the three directions.
c ste11, ste22, ste33 are the total elastic strain in the three directions.
c p is the pressure, pl is a limiting value of pressure, t is the critical
c   stress, k and ci are the damage parameter and surface, respectively.
c sg11, sg22, and sg33 are the stresses in the three directions.
c F is the current flexibility tensor, cc is the added flexibility, and
c d denotes the current stiffness tensor.
c mean is the trace of the incremental strain tensor.
c
      mean' = (dstt11+dstt22+dstt33)/3.0d0
c
      pr1 = 1. - nu
      pr2 = nu/pr1
      pr3 = 1. + nu
      pr4 = 1. / (pr3 * (1.0 - 2.0 * nu))
      e1  = ym * pr1 * pr4
      e2  = ym * nu * pr4
      tg  = ym / pr3
      t(elem,kk) = 0.0
      if(mean.gt.0.0) then
       e1 = model * e1
       e2 = model * e2
       tg = model * tg
      else
       continue
      endif
c
c Form elastic matrices for each element "elem", and Gauss pt. "k".
c
      d(1,1,elem,kk) = e1
      d(1,2,elem,kk) = e2
      d(1,3,elem,kk) = e2
      d(2,1,elem,kk) = e2
      d(2,2,elem,kk) = e1
      d(2,3,elem,kk) = e2
      d(3,1,elem,kk) = e2
      d(3,2,elem,kk) = e2
      d(3,3,elem,kk) = e1
      d(4,4,elem,kk) = tg
      d(5,5,elem,kk) = tg
      d(6,6,elem,kk) = tg
c
      F(1,1,elem,kk) = 1./ym
      if(mean.gt.0.0) then
       F(1,1,elem,kk) = F(1,1,elem,kk)/model
      else
       continue
      endif
      F(1,2,elem,kk) = -nu * F(1,1,elem,kk)
      F(1,3,elem,kk) = F(1,2,elem,kk)
```

```
      F(2,1,elem,kk) = F(1,2,elem,kk)
      F(2,2,elem,kk) = F(1,1,elem,kk)
      F(2,3,elem,kk) = F(1,2,elem,kk)
      F(3,1,elem,kk) = F(1,2,elem,kk)
      F(3,2,elem,kk) = F(1,2,elem,kk)
      F(3,3,elem,kk) = F(1,1,elem,kk)
      F(4,4,elem,kk) = 1./tg
      if(mean.gt.0.0) then
       F(4,4,elem,kk) = F(4,4,elem,kk)/model
      else
       continue
      endif
      F(5,5,elem,kk) = F(4,4,elem,kk)
      F(6,6,elem,kk) = F(4,4,elem,kk)
c
  10  continue
c
c  This is the entry point after initialization. Total strain increments
c    assumed given in gstrain.
c
      call principal(gstrain,nn,ij,elem,kk,eigen,vector)
c
      dstt11 = eigen(1)
      dstt22 = eigen(2)
c
c
c       call constitutive law subroutine damage. ----------
      call damage(F,cc,cc1,cc2,d,sg11,sg22,sg33,sg12,p,stt11,
     !            stt22,stt33,stt12,dstt11,dstt22,dstt33,dstt12,
     !            ste11,ste22,ste33,ste12,sttvol,k,ft,t,delk,iswich,
     !            jp,std11,std22,std33,std12,fc,w,ci,alpha,nu,ym,nu,
     !            epsl,ipss,iter,ij,elem,kk,nn,vector)
c
      go to (30,40,50,100) itest
  30  if(sg11(elem,kk).le.sgl) go to 100
      go to 200
  40  if(sg11(elem,kk).ge.sgl) go to 100
      go to 200
  50  if(p.ge.pl) go to 100
c
  100 continue
c
c Stop the program, one way is to set ij = nn
c
      ij = nn
c
  200 continue
c
c itest is a switch as what we want to be done, if itest = 1 then
c  the limiting value of the run is the stress, if itest = 2 then
c  the limiting value is also the stress but with different implications,
c  if itest = 3 the the mean pressure is the limiting factor for the
c  run.
c
```

```
      return
      end
c
c-----------------------------------------------------------------
c
      subroutine  damage(F,cc,cc1,cc2,d,sg11,sg22,sg33,sg12,p,stt11,
     !           stt22,stt33,stt12,dstt11,dstt22,dstt33,dstt12,
     !           ste11,ste22,ste33,ste12,sttvol,k,ft,t,delk,iswich,
     !           jp,std11,std22,std33,std12,fc,w,ci,alpha,mu,ym,nu,
     !           epsl,ipss,iel,iter,ij,elem,kk,nn,vector)
c
c
      double precision F(6,6,40,4),cc(6,6,40,4),cc1(6,6,40,4),
     !           cc2(6,6,40,4),d(6,6,40,4)
      double precision sg11(40,4),sg22(40,4),sg33(40,4),
     !          sg12(40,4),p
      double precision stt11,stt22,stt33,stt12,dstt11,
     !          dstt22,dstt33,dstt12,ste11,ste22,ste33,ste12,
     !          sttvol
      double precision k(40,4),t(40,4),ft,delk,epsl
      double precision std11,std22,std33,std12
      double precision ci(40,4),fc,w,alpha,mu,ym,nu
      double precision sgz11,sgz22,sgz33,sgz12
      double precision sgnp11,sgnp22,sgnp33,sgnn11,sgnn22,sgnn33
      double precision dedp11,dedp22,dedp33,dedn11,dedn22,dedn33
      double precision dstd11,dstd22,dstd33,dstd12
      double precision dste11,dste22,dste33,dste12
      double precision dn,d21,d31,ac21,ac31
      double precision tz,dsg11,dsg22,dsg33,dsg12
      double precision trace5,trace1,trace2,trace3
      double precision sgp11,sgp22,sgp33,sgn11,sgn22,sgn33
      double precision minsg,sge11,sge22,sge33
      double precision a11,a22,a33,a12,a13,a23
      double precision ciz,second,third,fifth,dkz,dk
      double precision aa(3,3),ainv(3,3)
      double precision sup11,sup22,sup33,sun11,sun22,sun33
      double precision vector(3,3),vectort(3,3)
      double precision TR1(3,3),TR2(3,3)
      integer          iswich,jp
      integer          ipss,iel,iter,ij,elem,kk
      integer          i,j,iterm,ji
      integer          iterz,nn
c
c     ************************************************************
      data iterm/10/,pib2/1.570796/
c
c     initialize parameters for this step. --------------------
      iter  = 0
      dk    = 0.
      iterz = iter
      sgz11 = sg11(elem,kk)
      sgz22 = sg22(elem,kk)
      sgz33 = sg33(elem,kk)
      sgz12 = sg12(elem,kk)
```

```
c
      sgnp11 = 0.d0
      sgnp22 = 0.d0
      sgnp33 = 0.d0
      sgnn11 = 0.d0
      sgnn22 = 0.d0
      sgnn33 = 0.d0
c
      do 10 i=1,6
      do 5 j=1,6
      cc1(i,j,elem,kk) = 0.
      cc2(i,j,elem,kk) = 0.
    5 continue
   10 continue
c
c
      do 12 i=1,3
      do 11 j=1,3
      vectort(j,i) = vector(i,j)
   11 continue
   12 continue
c
c     Assume step is elastic. ----------
      dedp11 = 0.d0
      dedp22 = 0.d0
      dedp33 = 0.d0
      dedn11 = 0.d0
      dedn22 = 0.d0
      dedn33 = 0.d0
c
      dstd11 = 0.d0
      dstd22 = 0.d0
      dstd33 = 0.d0
      dstd12 = 0.d0
c
      dste11 = dstt11
      dste22 = dstt22
      dste33 = dstt33
      dste12 = dstt12
c
c ipss is a path prescriber
c      if ipss = 1  -----  uniaxial stress,
c      if ipss = 2  -----  biaxial  stress,
c      if ipss = 3  -----  general stress path,
c      if ipss = 4  -----  user prescribed path.
      if(ipss.ne.4) go to 110
        dn = 1.0 - nu * (ac21 + ac31)
        d21= (ac21 - nu * (1.0 + ac31))/dn
        d31= (ac21 - nu * (1.0 + ac21))/dn
        dste22 = d21 * dste11
        dste33 = d31 * dste11
      go to 150
c
  110 if(ipss.eq.2) go to 130
```

```
      if(ipss.eq.3) go to 150
        dste22 = F(2,1,elem,kk) * dste11 / F(1,1,elem,kk)
        dste33 = F(2,1,elem,kk) * dste11 / F(1,1,elem,kk)
      go to 150
  130 dste33 = -(d(3,1,elem,kk)*dste11 + d(3,2,elem,kk)*dste22)/
     !         d(3,3,elem,kk)
  150 continue
      tz = t(elem,kk)
c
c     Update stresses. ------------------
      dsg11 = d(1,1,elem,kk)*dste11 + d(1,2,elem,kk)*dste22 +
     !        d(1,3,elem,kk)*dste33
      dsg22 = d(2,1,elem,kk)*dste11 + d(2,2,elem,kk)*dste22 +
     !        d(2,3,elem,kk)*dste33
      dsg33 = d(3,1,elem,kk)*dste11 + d(3,2,elem,kk)*dste22 +
     !        d(3,3,elem,kk)*dste33
      dsg12 = d(4,4,elem,kk)*dsg11
      if(ipss.eq.1) then
       dsg22=0.d0
       dsg33=0.d0
      else
      endif
      if(ipss.eq.2) dsg33 = 0.0d0
c
      sg11(elem,kk) = sgz11 + dsg11
      sg22(elem,kk) = sgz22 + dsg22
      sg33(elem,kk) = sgz33 + dsg33
      sg12(elem,kk) = sgz12 + dsg12
      p     = -(sg11(elem,kk) + sg22(elem,kk) + sg33(elem,kk))/3.
      trace5 = sg11(elem,kk)*sg11(elem,kk) +
     !         sg22(elem,kk)*sg22(elem,kk) +
     !         sg33(elem,kk)*sg33(elem,kk)
      if(jp.eq.1) go to 300
c jp is a switch that if jp = 1 we are in a hydrostatic stress path
c  which in this case damage does not occur and no need to
c  go through the damage subroutine.
c
c     form P+(sg) (Positive cone of stress tensor)--------------------
      sgp11 = sg11(elem,kk)
      sgp22 = sg22(elem,kk)
      sgp33 = sg33(elem,kk)
      if(sgp11.lt.0.) sgp11 = 0.
      if(sgp22.lt.0.) sgp22 = 0.
      if(sgp33.lt.0.) sgp33 = 0.
c
c     get (sgij+ : sgij+)
      trace1 = sgp11**2.d0 + sgp22**2.d0 + sgp33**2.d0
c
c     form P-(sg) (Negative cone of stress tensor) --------------------
      sgn11 = sg11(elem,kk)
      sgn22 = sg22(elem,kk)
      sgn33 = sg33(elem,kk)
      if(sgn11.gt.0.) sgn11 = 0.
      if(sgn22.gt.0.) sgn22 = 0.
```

```
      if(sgn33.gt.0.) sgn33 = 0.
c
c     determine the min. eigenvalue of the sgn tensor.----------
      minsg = dabs(sgn11)
      if(dabs(sgn22).lt.dabs(sgn11)) then
        minsg = dabs(sgn22)
        if(dabs(sgn33).lt.dabs(sgn22))minsg = dabs(sgn33)
        continue
      else
      if(dabs(sgn33).lt.dabs(sgn11)) minsg= dabs(sgn33)
      endif
c
c     determine sge where
c     sge is a shifted stress tensor.
c
      sge11 = sgn11 - minsg
      sge22 = sgn22 - minsg
      sge33 = sgn33 - minsg
      trace2 = sge11**2.d0 + sge22**2.d0 ++ sge33**2.d0
c
c     form the fourth order damage tensor R(sigma minus)
      a11 = sge11*sge11
      a22 = sge22*sge22
      a33 = sge33*sge33
      a12 = sge11*sge22
      a13 = sge11*sge33
      a23 = sge22*sge33
c
      trace3 = (sgn11*a11*sgn11)+(sgn22*a22*sgn22)+(sgn33*a33*sgn33)+
     *        2*(sgn11*a12*sgn22 + sgn11*a13*sgn33 + sgn22*a23*sgn33)
c
c
      iter = iter + 1
      ciz  = ci(elem,kk)
c
c     evaluate the damage surface. ----------
      if(trace2 .lt. .0000001) then
       second = 0.d0
       third  = 0.d0
       fifth  = 0.d0
      else
       second = w*(trace3)/trace2
       third  = w*(alpha)*9.d0*(p*p)
       fifth  = w*mu*alpha*trace5
      endif
c
      ci(elem,kk) = (trace1 + second + fifth - third -
     !              tz**2.d0)
      ci(elem,kk) = 0.5d0 * ci(elem,kk)
c     ci(elem,kk) = -2.0d0
c For linear elastic problems make ci a negative number as
c  suggested above.
c
c     check if the damage surface is reached. ----------
```

```
      if(ci(elem,kk).gt.epsl) go to 170
      if(iter .eq. 1) go to 250
      if(ci(elem,kk) .gt. -epsl) go to 250
  170 continue
      iswich = 1
c
c     compute dk using secant approximation. ------------------
      dkz = dk
      if(iter .eq. 1) dk = delk
      if(iter .eq. 1) go to 175
      dk = dkz * ci(elem,kk)/(ciz - ci(elem,kk))
  175 continue
c
c     compute the critical stress t. -------------------
      k(elem,kk) = k(elem,kk) + dk
      t(elem,kk) = ft *
     !      (dexp(1.d0))*dlog(1.d0+ym*k(elem,kk))/(1.d0+ym*k(elem,kk))
c
c     compute the added flexibility tensors in modes I and II. --------
c
      if(trace1 .le. .0001) go to 176
      cc1(1,1,elem,kk) = dk*(sgp11*sgp11)/trace1
      cc1(1,2,elem,kk) = dk*(sgp11*sgp22)/trace1
      cc1(1,3,elem,kk) = dk*(sgp11*sgp33)/trace1
      cc1(2,2,elem,kk) = dk*(sgp22*sgp22)/trace1
      cc1(2,3,elem,kk) = dk*(sgp22*sgp33)/trace1
      cc1(3,3,elem,kk) = dk*(sgp33*sgp33)/trace1
      cc1(2,1,elem,kk) = cc1(1,2,elem,kk)
      cc1(3,1,elem,kk) = cc1(1,3,elem,kk)
      cc1(3,2,elem,kk) = cc1(2,3,elem,kk)
      go to 179
  176 do 178 i = 1,6
      do 177 j = 1,6
  177 cc1(i,j,elem,kk) = 0.
  178 continue
  179 continue
c
      if(trace2 .le. 0.0001) go to 192
      cc2(1,1,elem,kk) = dk*(a11/trace2 + mu*alpha - alpha)
      cc2(1,2,elem,kk) = dk*(a12/trace2 - alpha)
      cc2(1,3,elem,kk) = dk*(a13/trace2 - alpha)
      cc2(2,2,elem,kk) = dk*(a22/trace2 + mu*alpha - alpha)
      cc2(2,3,elem,kk) = dk*(a23/trace2 - alpha)
      cc2(3,3,elem,kk) = dk*(a33/trace2 + mu*alpha - alpha)
      cc2(2,1,elem,kk) = cc2(1,2,elem,kk)
      cc2(3,1,elem,kk) = cc2(1,3,elem,kk)
      cc2(3,2,elem,kk) = cc2(2,3,elem,kk)
      go to 195
  192 do 194 i = 1.6
      do 193 j = 1,6
  193 cc2(i,j,elem,kk) = 0.0
  194 continue
  195 continue
c     compute the total added and the current flexibility tensors. --
```

```
      do 185 i=1,3
      do 180 j=1,3
      cc(i,j,elem,kk) = cc1(i,j,elem,kk) + cc2(i,j,elem,kk)
  180 continue
  185 continue
c
c cc(i,j,elem,kk) corresponds to the added flexibility in the
c  principal directions. Now with the transformation matrix
c  vector(3,3) we can rotate the corresponding matrices to the
c  global axes. The next few lines are to this effect.
c
c Initialize metrices:
c
      do 405 i=1,3
      do 405 j=1,3
      TR1(i,j) = 0.0d0
  405 TR2(i,j) = 0.0d0
c
      do 408 i=1,3
      do 407 j=1,3
      do 406 ji=1,3
       TR1(i,j) = TR1(i,j) + vector(i,ji) * cc(ji,j,elem,kk)
  406 continue
  407 continue
  408 continue
c
      do 412 i=1,3
      do 411 j=1,3
      do 410 ji=1,3
       TR2(i,j) = TR2(i,j) + TR1(i,ji) * vectort(ji,j)
  410 continue
  411 continue
  412 continue
c
      do 414 i=1,3
      do 413 j=1,3
      F(i,j,elem,kk)  = F(i,j,elem,kk) + TR2(i,j)
  413 continue
  414 continue
c
      do 287 i=1,3
      do 286 j=1,3
  286 aa(i,j) = F(i,j,elem,kk)
  287 continue
c
c     call the subroutine matinv to obtain the current stiffness
c          tensor. ------------------
      call matinv(aa,ainv)
      do 187 i=1,3
      do 186 j=1,3
      d(i,j,elem,kk) = ainv(i,j)
  186 continue
  187 continue
c
```

```
c     compute the incremental damage strain. ------------------
      sup11 = cc1(1,1,elem,kk)*sgp11 + cc1(1,2,elem,kk)*sgp22 +
     !        cc1(1,3,elem,kk)*sgp33
      sup22 = cc1(2,1,elem,kk)*sgp11 + cc1(2,2,elem,kk)*sgp22 +
     !        cc1(2,3,elem,kk)*sgp33
      sup33 = cc1(3,1,elem,kk)*sgp11 + cc1(3,2,elem,kk)*sgp22 +
     !        cc1(3,3,elem,kk)*sgp33
      sun11 = cc2(1,1,elem,kk)*sg11(elem,kk) + cc2(1,2,elem,kk)*
     !        sg22(elem,kk) + cc2(1,3,elem,kk)*sg33(elem,kk)
      sun22 = cc2(2,1,elem,kk)*sg11(elem,kk) + cc2(2,2,elem,kk)*
     !        sg22(elem,kk) + cc2(2,3,elem,kk)*sg33(elem,kk)
      sun33 = cc2(3,1,elem,kk)*sg11(elem,kk) + cc2(3,2,elem,kk)*
     !        sg22(elem,kk) + cc2(3,3,elem,kk)*sg33(elem,kk)
c
      dedp11 = dedp11 + sup11
      dedp22 = dedp22 + sup22
      dedp33 = dedp33 + sup33
      dedn11 = dedn11 + sun11
      dedn22 = dedn22 + sun22
      dedn33 = dedn33 + sun33
c
c     compute the total damage strain increments. ------------------
      dstd11 = dedp11 + dedn11
      dstd22 = dedp22 + dedn22
      dstd33 = dedp33 + dedn33
c
c     recompute the elastic strain increments. ------------------
      dste11 = dste11 - sup11 - sun11
      dste22 = dste22 - sup22 - sun22
      dste33 = dste33 - sup33 - sun33
      dste12 = dste12
c
c     adjust for the special conditions. ------------------
      if(ipss.eq.4) go to 240
      if(ipss.eq.3) go to 245
      if(ipss.eq.2) go to 230
      dste22 = F(2,1,elem,kk) * dste11/ F(1,1,elem,kk)
      dste33 = dste22
      go to 245
  230 dste33 = -(d(3,1,elem,kk)*dste11 + d(3,2,elem,kk)*dste22)/
     !         d(3,3,elem,kk)
      go to 245
  240 dste22 = d21 * dste11
      dste33 = d31 * dste11
  245 if(iter .le. iterm) go to 150
  250 continue
c
c
c     update strains. ------------------
c
c Note that for the particular problem of the shear wall
c  and due to the space limitations of the computer directory,
c  total strains were not stored. Becuase of this the next
c  few lines are commented out. If one desires to store total
```

```
c  strain also for reference, 2-dimensional arrays should be
c  set up similar to the stress tensor, e.g. std11(elem,kk),
c  stt11(elem,kk).
c
c     std11 = std11 + dstd11
c     std22 = std22 + dstd22
c     std33 = std33 + dstd33
c     std12 = std12 + dstd12
c     stdvol= std11 + std22 + std33
c
  300 continue
c
c     ste11 = ste11 + dste11
c     ste22 = ste22 + dste22
c     ste33 = ste33 + dste33
c     ste12 = ste12 + dste12
c
c     dstt11 = dste11 + dstd11
c     dstt22 = dste22 + dstd22
c     dstt33 = dste33 + dstd33
c     dstt12 = dste12 + dstd12
c
c     stt11 = stt11 + dstt11
c     stt22 = stt22 + dstt22
c     stt33 = stt33 + dstt33
c     stt12 = stt12 + dstt12
c     sttvol= stt11 + stt22 + stt33
c
      return
      end
c
c
c -------------------------------------------------------
c
      subroutine matinv(aa,ainv)
c
c This subroutine inverts a given matrix and stores it in ainv.
c
c
      double precision aa(3,3),ainv(3,3)
      double precision b(5,10)
      double precision temp
      integer n,i,j,j1,j2,k,kp1,l,km1,j2
      n = 3
c
      do 1 i=1,n
      do 1 j=1,n
    1 b(i,j) = aa(i,j)
c     load right half of matrix b with unit matrix. ----------
      j1 = n + 1
      j2 = 2 * n
      do 2 i=1,n
      do 2 j=j1,j2
    2 b(i,j) = 0.d0
```

```
      do 3 i=1,n
      j = i + n
    3 b(i,j) = 1.d0
c
c     start the pivotal condensation. ----------
c     k names the pivotal row. ----------
      do 610 k = 1,n
      kp1 = k + 1
      if(k .eq. n) go to 500
      l = k
      do 400 i = kp1,n
  400 if(abs(b(i,k)) .gt. abs(b(l,k))) l= i
      if(l.eq.k) go to 500
      do 410 j=k,j2
      temp = b(k,j)
      b(k,j) = b(l,j)
  410 b(l,j) = temp
  500 do 501 j = kp1,j2
  501 b(k,j) = b(k,j) / b(k,k)
      if(k .eq. 1) go to 600
      km1 = k -1
      do 510 i = 1,km1
      do 510 j = kp1,j2
  510 b(i,j) = b(i,j) - b(i,k) * b(k,j)
      if(k .eq.n) go to 700
  600 do 610 i = kp1,n
      do 610 j = kp1,j2
  610 b(i,j) = b(i,j) - b(i,k) * b(k,j)
  700 do 701 i = 1,n
      do 701 j = 1,n
      k = j + n
  701 ainv(i,j) = b(i,k)
c
      return
      end
c
c -------------------------------------------------------------
c
c234567
      s .broutine principal(gstrain,nn,ij,elem,kk,eigen,vector)
c
c This subroutine links up to an eigenvalue solver program
c  called dsyev available at the University of New Mexico.
c  Eigenvectors are obtained for the transformation matrix.
c
      double precision gstrain(40,4,3),A(3,3),work(50)
      double precision vector(3,3),eigen(3)
      integer          nn,ij,elem,kk
c
      A(1,1) = gstrain(elem,kk,1)
      A(1,2) = gstrain(elem,kk,3)
      A(2,1) = A(1,2)
      A(2,2) = gstrain(elem,kk,2)
      A(1,3) = 0.0d0
```

```
      A(2,3) = 0.0d0
      A(3,1) = A(1,3)
      A(3,2) = A(2,3)
      A(3,3) = -1.0
c
      call dsyev(A,3,3,eigen,vector,work,1,info)
      if(info.ne.0) write(6,*)'dsyev did not converge'
c
      return
      end
c
c -------------------------------------------------------------
c
c234567
      subroutine principal1(A,eigen1,numeq,mass,numelm)
c
c This subroutine links up with dsyev for the determination
c  of the minimum eigenvalue problem.
c
      double precision SS(200,200),A(200,200)
      double precision vector1(200,200),eigen1(200)
      double precision work(200),mass
      integer          i,j,numeq,numelm
      integer          l,ll,info
c
      do 11 i=1,numeq
       ll = i-1
       do 10 j=i,numeq
        l = j-ll
        SS(i,j) = A(i,l)
        SS(j,i) = SS(i,j)
        if(i.eq.j) then
         SS(i,j) = SS(i,j) * mass/numelm
        else
         continue
        endif
  10   continue
  11   continue
c
      call dsyev(SS,numeq,numeq,eigen1,vector1,work,0,info)
c
      return
      end
c
c -------------------------------------------------------------
```