REPORT NO.
UCB/SESM-84/05

STRUCTURAL ENGINEERING AND
STRUCTURAL MECHANICS

SOLUTION OF FINITE ELEMENT PROBLEMS BY PRECONDITIONED CONJUGATE GRADIENT AND LANCZOS METHODS

by

ROBERT L. TAYLOR

and

BAHRAM NOUR-OMID

MAY 1984

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

SOLUTION OF FINITE ELEMENT PROBLEMS BY PRECONDITIONED CONJUGATE GRADIENT AND LANCZOS METHODS

by

R. L. Taylor and B. Nour-Omid
Department of Civil Engineering
University of California, Berkeley

ABSTRACT

The solution of nonlinear, transient finite element problems may be achieved using step-by-step integration of the equations of motion combined with a Newton solution of the resulting nonlinear algebraic equations. The use of Newton type methods leads to a set of linear simultaneous algebraic equations whose solution gives the next iterate. For very large problems the solution of the large set of linearized equations may be a formidable task - often consuming more than half of the computing effort when performed by a direct method based upon Gauss elimination. Accordingly, it is of considerable importance to investigate alternative methods to solve the problem. The present study presents results obtained by using a Preconditioned Conjugate Gradient Method (PCG) described in [7] and a Preconditioned Lanczos Method (PLM) described in [6] to solve a variety of numerical examples. Based upon results obtained it is evident that a significant reduction in overall effort, compared to direct solutions, may be achieved using the preconditioned methods.

The work reported here was supported by the Naval Civil Engineering Laboratory, Port Hueneme, CA, under purchase order N62583/83 M T265.

1. Introduction

The finite element method of discretization is used to reduce many complex continuum problems to discrete systems. Although this reduction is the most important step in the overall analysis of a structure, solving the resulting discrete problem is often far from trivial. In general, the reduced system is nonlinear and an iterative method must be employed to arrive at the solution. Most solution methods are based on some form of Newton's method in which the nonlinear problem is linearized, using an initial approximation, to arrive at a linear set of simultaneous algebraic equations. The solution of the set of linear equations leads to a correction of the initial approximation. When solving the linear equations, one should not loose sight of the primary objective: solving the nonlinear problem.

Iterative methods, such as the conjugate gradient or Lanczos method, are among the many methods that may be used to solve systems of linear equations. The advantage of these methods, when used as the inner loop of the Newton iteration, is twofold.

(i) The linear equation may be solved to any desired level of accuracy as governed by the Newton iteration.

(ii) A considerable reduction in storage can be achieved when no triangular factorization need be performed.

In [6] a method was developed, based on the preconditioned Lanczos method, to realize some of the advantages of iterative methods. In this previous study, the triangular factors of the initial tangent matrix were used to form a preconditioning matrix for the subsequent solution steps. In the present study we have eliminated factorizations by employing other preconditioners and further, have reduced the storage needs of the method.

2. A Preconditioned Conjugate Gradient Method

An essential step in nonlinear analysis of structures using Newton's method (or a variant such as modified Newton or quasi-Newton methods) is solving a linear system of algebraic equations. The preconditioned conjugate gradient method (hereafter called PCG) is one of the many procedures for solving

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{0} \tag{2.1}$$

where $\mathbf{A}$ is an $n \times n$ symmetric positive definite matrix (which for finite element calculations is sparsely populated) and $\mathbf{b}$ is the right-hand side vector. In the case of static analysis, $\mathbf{A}$ is the current tangent matrix and in the case of dynamic analysis, $\mathbf{A}$ depends on the mass, damping and tangent stiffness matrices, as well as the time increment.

The initial popularity of the conjugate gradient method was due to a number of factors. In exact arithmetic the method required a maximum of n iterations to solve (2.1) which made the method superior to other iterative methods. In fact conjugate gradient is in the class of semi-iterative methods which also includes the Lanczos algorithm [10]. The disadvantage of direct methods is their large storage demands for keeping the factors of $\mathbf{A}$. The only interface between the conjugate gradient method and $\mathbf{A}$ is through the product $\mathbf{A}\mathbf{v}$ for a given vector $\mathbf{v}$. This is an elegant way of taking advantage of sparsity of $\mathbf{A}$ which has the added advantage that $\mathbf{A}$ need not be known explicitly but only a means of computing the matrix-vector product is required.

The popularity of the conjugate gradient method vanished once it was found that under certain conditions the method required as many as $5n$ or $6n$ steps to reduce the residual to the desired level. This degradation is due to the strong influence of round-off error.

The addition of preconditioning eliminated this difficulty. Instead of solving (2.1) we solve

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b} \tag{2.2}$$

for some appropriate choice of $\mathbf{P}$. The object then is to choose $\mathbf{P}$ such that the coefficient matrix of (2.2) is well conditioned.

Theoretical considerations suggest that at the end of each iteration of CG the residual norm is reduced by a factor $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ when solving (2.1) where κ is the condition number of $\mathbf{A}$, defined by $\kappa = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|$. See [1] for more details. Note that when $\kappa = 1$, one iteration is sufficient to solve the equation. This provides us with a guideline for choosing $\mathbf{P}$. For a well chosen $\mathbf{P}$ only a few iterations reduce the residual norm to the desired level. Here we give an outline for the preconditioned conjugate algorithm:

Given an initial guess $\mathbf{x}_0$, a positive definite preconditioning matrix $\mathbf{P}$, the matrix $\mathbf{A}$ and the right hand side $\mathbf{b}$:

```
(1)  Set $\mathbf{p}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$

(2)  Solve $\mathbf{P}\mathbf{d}_0 = \mathbf{r}_0$, for $\mathbf{d}_0$

(3)  for $k = 0, 1, 2, \cdots$ until convergence do

     (a)  $\alpha_k = (\mathbf{r}_k,\mathbf{d}_k)/(\mathbf{p}_k,\mathbf{A}\mathbf{p}_k)$

     (b)  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

     (c)  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$

     (d)  Solve $\mathbf{P}\mathbf{d}_{k+1} = \mathbf{r}_{k+1}$

     (e)  $\beta_k = (\mathbf{r}_{k+1},\mathbf{d}_{k+1})/(\mathbf{r}_k,\mathbf{d}_k)$

     (f)  $\mathbf{p}_{k+1} = \mathbf{d}_{k+1} + \beta_k \mathbf{p}_k$
```

The operation $(\mathbf{v},\mathbf{u})$ denotes the inner product $\mathbf{v}^T\mathbf{u}$. This algorithm generates a sequence of approximations to the solution $\mathbf{x}$ with a corresponding residual vector $\mathbf{r}_k$. The termination criterion can be chosen based on these quantities. In addition to storage demands for $\mathbf{A}$ and $\mathbf{P}$ the algorithm requires storage for 4 vectors. The total number of operation per iteration is $NZA + NZP + 5N$, where NZA and NZM are the number of operations for forming $\mathbf{A}\mathbf{u}$ and $\mathbf{P}^{-1}\mathbf{v}$ for a given $\mathbf{u}$ and $\mathbf{v}$.

3. Splitting Methods

Next we turn to a topic which at first sight may seem unrelated to the solution of non-linear algebraic equations. Consider the system of first order differential equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \tag{3.1}$$

where $\mathbf{x}$ is an n-dimensional vector, the superposed dot, ($\dot{}$), denotes differentiation with respect to time and $\mathbf{f}$ is a function of the unknown vector $\mathbf{x}$ and t.

We consider a special form of $\mathbf{f}$ which can be written as a sum of its subcomponents $\mathbf{f}_i$.

$$\mathbf{f} = \sum_{i=1}^{s} \mathbf{f}_i \tag{3.2}$$

Under these conditions the original problem can be thought as a sum of s subproblems

$$\dot{\mathbf{x}} = \mathbf{f}_i(\mathbf{x}, t) \qquad i = 1,...,s \tag{3.3}$$

In the case of finite element discretization of the spatial domain the sum in (3.2) ranges over the elements or a set of elements. In other cases the splitting may be formed by other means, one of which is demonstrated in the following section.

A consistent algorithm for the solution of (3.1), based on the notion of a splitting technique [2], can now be constructed as a product of algorithms for the sub-problems. In other words, write the algorithm for (3.3) as

$$\mathbf{x}_{m+1} = S_i^{(h)}[\mathbf{x}_m] \tag{3.4}$$

where $S_i^{(h)}$ is an operator denoting the algorithm and the index m ranges over the increment in time, h. Then the algorithm for (3.1) can be written as

$$\mathbf{x}_{m+1} = S^{(h)}[\mathbf{x}_m] \tag{3.5}$$

where

$$S^{(h)} = \prod_{i=1}^{s} S_i^{(h)} \tag{3.6}$$

One of the disadvantages of the splitting method is its low accuracy. The best that these methods can achieve is second order accuracy. That is the truncation error is of the order of h^3 at best. In the sequel we will use the above procedure to construct a preconditioning matrix for the conjugate gradient algorithm described in section 2. The inherent inaccuracy of the splitting method poses no problem since the algorithm is used only as a preconditioner and therefore one can obtain very high accuracies through the conjugate gradient iteration.

4. Solution of Static Problems

Consider the system of linear first order differential equations

$$\tau \dot{\mathbf{x}} + \mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.1}$$

where τ is a given parameter. Formally the solution to equation (4.1) is

$$\mathbf{x}(t) = e^{-\mathbf{A}t/\tau}(\mathbf{x}_0 - \mathbf{A}^{-1}\mathbf{b}) + \mathbf{A}^{-1}\mathbf{b} \tag{4.2}$$

where $\mathbf{x}_0 = \mathbf{x}(0)$, is an initial condition. We observe from (4.2) thet as t ends to infinity $\mathbf{x}(t)$ converges to the solution of (2.1) for $\tau > 0$. Consiquently (4.1) may be utilized to solve the linear equations (2.1). Indeed this approach has been suggested previously (e.g., see [9]). In general the exponential of a large matrix cannot be easily computed and a numerical solution of (4.1) must be used. In order to achieve a soluion of (2.1) the numerical solution to (4.1) must be assymptotically correct for infinite h, or a very large number of time steps must be used to compute the solution at infinite time. Here we are not concerned with constructing an accurate solution to (4.1), rather we consider the method as a means of constructing a suitable preconditioning matrix for the conjugate gradient algorithm described above.

Splitting methods may be applied to any problem of the form

$$\dot{\mathbf{x}} = \mathbf{B}\mathbf{x} \tag{4.3}$$

where $\mathbf{B}$ is an additive operator defined by

$$\mathbf{B} = \sum_{i=1}^{s} \mathbf{B}_i \tag{4.4}$$

such that the equations

$$\dot{\mathbf{x}} = \mathbf{B}_i\mathbf{x} \qquad i = 1,...,s \tag{4.5}$$

are significantly easier to solve than the original equations. The time stepping algorithm for the global problem is then the product of all the time stepping algorithms for the subproblems with a fractional time step h/s [2].

The coefficient matrix $\mathbf{A}$ in (2.1) may be written as the sum of its diagonal matrix, $\mathbf{D}$, a strictly lower triangular matrix, $\mathbf{L}$, and a srictly upper triangular matrix such that

$$\mathbf{A} = (\tfrac{1}{2}\mathbf{D} + \mathbf{L}) + (\tfrac{1}{2}\mathbf{D} + \mathbf{L})^T \tag{4.6}$$

The associated subproblems, $\dot{\mathbf{x}} = -(\tfrac{1}{2}\mathbf{D} + \mathbf{L})\mathbf{x}$ and $\dot{\mathbf{x}} = -(\tfrac{1}{2}\mathbf{D} + \mathbf{L}^T)\mathbf{x}$ can be solved easily.

Applying a backward difference method with a time step $h/2$ to each of the subproblems, we arrive at

$$\mathbf{x}_{m+1} = \left[\mathbf{I} + \frac{h}{4\tau}(\mathbf{D} + 2\mathbf{L}\right]^{-1}\left[\mathbf{I} + \frac{h}{4\tau}(\mathbf{D} + 2\mathbf{L}^T)\right]^{-1}\left(\frac{h}{2\tau}\mathbf{b} + \mathbf{x}_m\right] \tag{4.7}$$

where $\mathbf{x}_m$ is an approximation to $\mathbf{x}(mh)$. For an initial condition $\mathbf{x}_0 = 0$ we get an approximation to $\mathbf{x}(h)$

$$\mathbf{x}_1 = \left\{\frac{h}{2\tau}\left[\mathbf{I} + \frac{h}{4\tau}(\mathbf{D} + 2\mathbf{L})\right]^{-1}\left[\mathbf{I} + \frac{h}{4\tau}(\mathbf{D} + 2\mathbf{L}^T)\right]^{-1}\right\}\mathbf{b} \tag{4.8}$$

which is compared to the exact solution

$$\mathbf{x}(h) = \left(\mathbf{I} - e^{-\mathbf{A}t/\tau}\right]\mathbf{A}^{-1}\,\mathbf{b} \tag{4.9}$$

Comparing equations (4.8) and (4.9) suggests that the coefficient matrix in (4.8) may be a good approximation to $\mathbf{A}^{-1}$ for large h, and may therefore be an effective preconditioning matrix. The scalar factor $\frac{h}{2\tau}$ may be ignored for preconditioning purposes.

When using (4.8) in conjunction with the conjugate gradient algorithm of section 2 the preconditioning matrix becomes

$$\mathbf{P} = (\mathbf{I} + \omega/2\mathbf{D} + \omega\mathbf{L})(\mathbf{I} + \omega/2\mathbf{D} + \omega\mathbf{L}^T) \tag{4.10}$$

where $\omega = h/2\tau$ is now a free parameter.

To simplify the choice of ω we scale the stiffness matrix $\mathbf{A}$ such that diagonal of $\mathbf{A}$ is unity. The resulting matrix is $\overline{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The system of equations (2.1) now becomes

$$\overline{\mathbf{A}}\,\overline{\mathbf{x}} = \overline{\mathbf{b}} \tag{4.11}$$

where $\overline{\mathbf{x}} = \mathbf{D}^{1/2}\,\mathbf{x}$ and $\overline{\mathbf{b}} = \mathbf{D}^{-1/2}\,\mathbf{b}$.

The preconditioned matrix must now be applied to (4.11) resulting in

$$\overline{\mathbf{P}} = (\mathbf{I} + \omega\overline{\mathbf{L}})(\mathbf{I} + \omega\overline{\mathbf{L}}^T) \tag{4.12}$$

where $\overline{\mathbf{A}} = \overline{\mathbf{L}} + \overline{\mathbf{L}}^T$. It is easy to show that preconditioning (4.11) using $\overline{\mathbf{P}}$ is equivalent to preconditioning (2.1) with

$$\mathbf{P} = (\mathbf{D} + \omega\mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \omega\mathbf{L}^T) \tag{4.13}$$

This can be identified as a member of the class of incomplete Choleski preconditioners [3].

Note that when $\omega = 0$, $\mathbf{P}$ becomes the diagonal matrix $\mathbf{D}$, resulting in the simplest form of preconditioning; diagonal scaling. When $\omega = 1$ then $\mathbf{P} = \mathbf{A} + \mathbf{L}\mathbf{D}^{-1}\mathbf{L}^T$ where we note that the error matrix $\mathbf{L}\mathbf{D}^{-1}\mathbf{L}^T$ is rank deficient since $\mathbf{L}$ has zero diagonals. If the norm of $\mathbf{D}$ is larger then the norm of $\mathbf{L}$ then the norm of the error matrix will be small compared to the norm of $\mathbf{A}$. consequently, for most problems it is expected that the optimum ω will be close to unity.

5. Solution of Dynamic Problems

We next construct a preconditioning matrix for the linear system of equations arising in a step-by-step algorithm for dynamic analysis of linear and nonlinear structures. In particular, we consider the Newmark algorithm and the preconditioning matrix follows from the splitting method of section 3, in much the same way as for the static problem.

Consider the linearized equations of motion

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f} \tag{5.1}$$

where $\mathbf{M}$ is the diagonal mass matrix, $\mathbf{K}$ is the stiffness matrix, $\mathbf{f}$ is the external load vector and $\mathbf{u}$ is the response of the structure. For simplicity, we ignore damping effects; however, all of the following results may be extended easily to the damped case. Accordingly, the linearized system of equations arising at every time step of the Newmark method is

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{5.2}$$

where

$$\mathbf{A} = \mathbf{K} + \frac{1}{\beta\Delta t^2}\mathbf{M} \tag{5.3}$$

and

$$\mathbf{b} = \mathbf{f}_{t+\Delta t} + \frac{1}{\beta\Delta t^2}\mathbf{M}[\mathbf{u}_t + \Delta t\mathbf{v}_t + (½ - \beta)\Delta t^2\mathbf{a}_t\,] \tag{5.4}$$

Here $\mathbf{v}$ and $\mathbf{a}$ are velocity and acceleration vectors, respectively, Δt is the specified time increment, t is the time and $\mathbf{x}$ is now the increment of displacement response. The Newmark parameters are chosen such that $\beta \geqslant (½ + \gamma)^2/4$ with $\gamma \geqslant ½$ which ensures unconditional local stability. The discretizations in time are

$$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta t\mathbf{v}_t + +½\,\Delta t^2\,[(1-2\beta)\mathbf{a}_t + 2\beta\mathbf{a}_{t+\Delta t})\,] \tag{5.5}$$

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t(1-\gamma)\mathbf{a}_t + \gamma\Delta t\mathbf{a}_{t+\Delta t} \tag{5.6}$$

The object is to solve (5.2) without forming the factors of $\mathbf{A}$.

A splitting method similar to the one used for equation (4.1) can now be applied to equation (5.1). The matrix resulting from the splitting algorithm can then be used as a preconditioner for (5.2). Consider

$$\mathbf{P} = (\mathbf{L} + \frac{1}{\beta\Delta t^2}\mathbf{M})\mathbf{M}^{-1}(\mathbf{L}^T + \frac{1}{\beta\Delta t^2}\mathbf{M}) \tag{5.7}$$

where $\mathbf{K} = \mathbf{L} + \mathbf{L}^T$. Multiplying out the terms in (5.7), we obtain

$$\begin{aligned} \mathbf{P} &= \frac{1}{\beta\Delta t^2}\,[\beta\Delta t^2\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T + \mathbf{L} + \mathbf{L}^T + \frac{1}{\beta\Delta t^2}\,\mathbf{M}] \\ &= \frac{1}{\beta\Delta t^2}\,[\beta\Delta t^2\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T + \mathbf{A}] \\ &= \frac{1}{\beta\Delta t^2}\,[\mathbf{E}(\Delta t^2) + \mathbf{A}] \end{aligned} \tag{5.8}$$

where $\mathbf{E}(\Delta t^2) = \beta\Delta t^2\mathbf{L}\mathbf{M}^{-1}\mathbf{L}^T$.

The preconditioned conjugate gradient algorithm of section 2 is invariant under the scaling of the preconditioning matrix; therefore, (5.8) shows that **P** will tend quadratically to the dynamic stiffness matrix **A** as the time step diminishes. In other words, **E** tends to the zero matrix quadratically in Δt. We see later that this characteristic results in an effective preconditioning and the solution of equation (5.2) is obtained in very few iterations of the preconditioned conjugate gradient algorithm.

6. Lanczos Algorithm for Solution of the Linearized Problem

The discretization of nonlinear structural mechanics problems and linearization of the resulting nonlinear algebraic equations for application of a Newton type methods normally will lead to a symmetric system of linear algebraic equations, (2.1).

In Section 2 we described the use of the conjugate gradient method to solve the linear system of equations. In this section an alternative solution technique, a simple Lanczos method, is presented.

Iterative methods often have been used in numerical analysis for the solution of large systems of equations. The Conjugate Gradient method is one such technique introduced in 1952 by Hestenes and Stiefel [4]. In the same year Lanczos published his method of minimized iteration which was initially introduced for computing the eigen pairs of a large symmetric matrix. Lanczos and Householder [5] pointed out the intimate connection between the two approaches. These methods have several attractive features in common. There is no need for $\mathbf{A}$ to have further special properties, such as banded form, no acceleration parameters have to be estimated, and the storage requirements are only a few n-vectors in addition to the storage needs of $\mathbf{A}$.

6.1. The Lanczos Algorithm

For certain applications of the finite element method, especially in nonlinear problems, it is usual to have on hand an initial approximation $\mathbf{x}^a$ to the true solution of 2.1. The problem is now to find a correction $\mathbf{x}^c$ to be added to $\mathbf{x}^a$. Then

$$\mathbf{A}\mathbf{x}^c = \mathbf{r}_0 \tag{6.1}$$

where

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}^a \tag{6.2}$$

The Lanczos algorithm may be described very simply as a process of constructing the weak form of equation 2.1 from a very special subspace. The subspace under consideration is generated from the set of j vectors ($\mathbf{r}_0$, $\mathbf{A}\mathbf{r}_0$, $\cdots$, $\mathbf{A}^{j-1}\mathbf{r}_0$), known as the Krylov subspace [9].

To construct the weak form it would be simpler if an orthonormal set of vectors, say $(\mathbf{q}_1,\mathbf{q}_2, \cdots ,\mathbf{q}_j)$, were available. This can be achieved by applying Gram-Schmidt orthogonalization to the Krylov vectors. Initially, this appears to be an expensive way of obtaining an orthonormal base vectors, however this process can be simplified when the following two facts are used [9]:

(i) The use of $\mathbf{A}\mathbf{q}_j$ and $\mathbf{A}^j\mathbf{r}_0$, for orthogonalization against the previous $\mathbf{q}$ vectors and normalization of the resulting vector, leads to the same vector $\mathbf{q}_{j+1}$.

(ii) The vector $\mathbf{A}\mathbf{q}_j$ is orthogonal to $\mathbf{q}_1,\mathbf{q}_2, \cdots ,\mathbf{q}_{j-2}$.

Consequently it is sufficient to orthogonalize $\mathbf{A}\mathbf{q}_j$ against $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$ to obtain the next orthogonal vector. Accordingly,

$$\mathbf{r}_j \equiv \beta_{j+1}\mathbf{q}_{j+1} = \mathbf{A}\mathbf{q}_j - \alpha_j\mathbf{q}_j - \beta_j\mathbf{q}_{j-1} \tag{6.3}$$

where $\alpha_j = \mathbf{q}_j^T\mathbf{A}\mathbf{q}_j$ and $\beta_j = \mathbf{q}_{j-1}^T\mathbf{A}\mathbf{q}_j$. It is important to note that the vectors $(\mathbf{q}_1,\mathbf{q}_2, \cdots ,\mathbf{q}_{j-2})$ are not needed in equation 6.3 . This defines one step of the simple Lanczos algorithm. The normalization of $\mathbf{r}_j$ results in $\mathbf{q}_{j+1}$. It is easy to show, by looking at $\mathbf{q}_{j+1}^T\mathbf{r}_j$, that $\beta_{j+1} = \|\mathbf{r}_j\|$.

The special choice for the base vectors of the subspace has an additional advantage. The projection of $\mathbf{A}$ onto this subspace is a tridiagonal matrix, $\mathbf{T}_j$.

$$\mathbf{T}_j = \mathbf{Q}_j^T\mathbf{A}\mathbf{Q}_j = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \cdot & & \\ & & \cdot & & \\ & & & \alpha_{j-1} & \beta_j \\ & & & \beta_j & \alpha_j \end{bmatrix} \tag{6.4}$$

where the $\mathbf{q}$ vectors form the columns of the matrix $\mathbf{Q}_j$, $\mathbf{Q}_j \equiv (\mathbf{q}_1,\mathbf{q}_2, \cdots ,\mathbf{q}_j)$. This fact was realized soon after Lanczos introduced his method and the algorithm was put to use as a process for the orthogonal transformation of a matrix to tridiagonal form. Despite its additional attractions, the Lanczos process gave way to Givens' method in 1954 and later to Householder's method in 1958.

The relationships that define the simple Lanczos algorithm can now be summarized in the following three equations.

$$\mathbf{Q}_j^T\mathbf{Q}_j = \mathbf{I}_j \tag{6.5.a}$$

$$\mathbf{A}\mathbf{Q}_j - \mathbf{Q}_j\mathbf{T}_j = \mathbf{r}_j\mathbf{e}_j^T \tag{6.5.b}$$

$$\mathbf{Q}_j^T\mathbf{r}_j = 0 \tag{6.5.c}$$

where $\mathbf{e}_j$ is the j-th column of the $j \times j$ identity matrix $\mathbf{I}_j$. Setting $\mathbf{q}_0 = 0$ and using $\mathbf{r}_0$ as the starting vector, the Lanczos algorithm may then be described as

```
Given $\mathbf{r}_0$, set $\beta_1 = \|\mathbf{r}_0\|$, for $j = 1,2, \cdots$ repeat
(1) $\mathbf{q}_j \leftarrow \frac{\mathbf{r}_{j-1}}{\beta_j}$
(2) $\mathbf{u}_j \leftarrow \mathbf{A}\mathbf{q}_j$
(3) $\mathbf{r}_j \leftarrow \mathbf{u}_j - \beta_j\mathbf{q}_{j-1}$
(4) $\alpha_j \leftarrow \mathbf{q}_j^T\mathbf{r}_j$
(5) $\mathbf{r}_j \leftarrow \mathbf{r}_j - \alpha_j\mathbf{q}_j$
(6) $\beta_{j+1} \leftarrow \|\mathbf{r}_j\|$
```

While a direct use of the simple Lanczos algorithm usually leads to numerical difficulties, thus requiring some reorthogonalization of the vectors, with care in selecting the preconditioning matrix these difficulties are avoided. In our test of the algorithm to date no difficulties have been encountered in using the simple preconditioned Lanczos algorithm (see Numerical Examples).

7. Modification to the FEAP Program.

The finite element computer program FEAP is described in Chapter 24 of Zienkiewicz [13] and forms the basis for all computations performed as part of the current effort. Several basic modifications have occurred since the original program was published. These include: (a) replacement of the original subprograms ACTCOL and UACTCL for the direct solution of linear algebraic equations for symmetric and unsymmetric problems, respectively, by the single solver DASOL which is probably the most efficient implementation of a variable band, active column equation solver available today for virtual memory machines; (b) inclusion of the Newmark method to integrate the equations of motion for linear and nonlinear problems in structural mechanics; (c) the ability to construct a tangent stiffness matrix and a residual force simultaineously (instead of using TANG followed by FORM, e.g., see below); (d) a convergence criteria based upon the current increment in energy (e.g., $\Delta \mathbf{x}^T \mathbf{r}$); and (e) inclusion of a general data storage structure for nonlinear materials which require additional information to the displacements in order to evaluate the current stress state at each integration point in an element.

In the work reported here the program FEAP has been extended further to include the capability of solving problems using the Preconditioned Conjugate Gradient Method (PCG) or the Preconditioned Lanczos Method (PLM) described above. In addition, a line search algorithm has been incorporated for use with any of the solution methods - i.e., direct, PCG, or PLM. As noted above, the preconditioning matrices require a knowledge of the nonzero terms in the global tangent stiffness matrix (as well as the mass matrix for dynamics problems). Since it is not necessary to factor the preconditioning matrix, which would cause fill in the nonzero structure, we have developed a direct means to construct the array containing the nonzero terms.

The algorithm to construct the locations of the nonzero terms in the compressed tangent stiffness matrix may be summarized by the following steps:

1.) Make a list of the elements which are attached to each node. In FEAP this step is accomplished by constructing dynamically dimensioned arrays which will contain only the terms associated with the actual nonzero structure; so that no storage will be wasted. Accordingly, the first step is divided into two parts - the first to determine dimensioning and the second to obtain the actual elements connected to each node.

2.) For each nodal degree-of-freedom use the list of elements to find all the other nodal degree-of-freedoms which are connected. Since we are currently considering symmetric equations only the terms above the diagonal entry are constructed.

The above two steps have been incorporated into the set of subprograms which are listed in Appendix A. In FEAP the array of nonzero terms may be constructed by using the macro command CTAN (compressed tangent stiffness matrix) instead of the usual TANG (note that there is no equivalent compressed array for UTAN since neither of the preconditioned solvers is programmed to handle unsymmetric equations). Thus a typical Newton iteration using the PCG algorithm is given by the set of macro statements:

```
LOOP,NEWT,10
CTAN
FORM
PCG ,LINE,1.,100.
NEXT
```

where the LOOP for the NEWTon step is to be executed for a maximum of 10 iterations (the iteration will terminate earlier if the prescribed tolerance on energy is met), CTAN indicates a compressed tangent is to be constructed as described above, and PCG indicates that the Preconditioned Conjugate Gradient algorithm is to be used with LINE search (omit LINE if no line search is desired - the current search requires several evaluations of the residual, consequently, for large problems may be time consuming), the 1. indicates that the value of ω is unity, and 100. is the maximum number of PCG iterations to be used to solve the equations. Alternatively, the commands:

```
LOOP,NEWT,10
CTAN,----,1.
LANC,LINE,1.,100.
NEXT
```

may be used. The nonzero value on the CTAN instruction indicates that a residual is to be computed as well as the compressed tangent and is thus equivalent to CTAN and FORM. The use of LANC instead of PCG indicates that the Lanczos algorithm is to be employed.

8. Numerical Examples

The preconditioned conjugate gradient algorithm (as well as the Lanczos method) has been tested on a series of test problems. In order to assess the overall efficiency of problem solution we report both the computing time (for a VAX 11/780 computer operating under UNIX 4.2 BSD) and storage requirements for the coefficient array. In our test problems we include two and three dimensional solids subjected to both static and dynamic loading states.

Example 1. Two dimensional cantilever structure.

The first example considered is a cantilever structure with two holes to induce added stress gradient effects. The model consists of 225 nodes with 184 4-node plane elements, see Figure 1. The material is linear elastic and utilizes ELMT01 described in Chapter 24 of [13]. For this problem (as well as all subsequent analyses) we perform a solution using the direct solution of the equations as well as the PCG and Lanczos algorithms. The timing and performance of the PCG and Lanczos methods utilized are nearly identical, accordingly, we shall report only the results for the PCG algorithm. The essential results for the cantilever structure are summarized below.

```
Model: Cantilever type structure
       (225 nodes 184 elements)
       profile  9990          Non-zero terms:  3162

Static
        Direct
             total time:  16.77
        PCG
             total time:  28.78    <39 iterations>

Dynamic
(5 time  Direct
 steps)       total time:  32.42
        PCG
             total time:  77.52    <24 to 27 iterations>
```

These results are much as expected - indicating that the iterative PCG algorithm requires more solution effort (measured in CPU time) than a direct solution. The only redeeming feature for this example is the reduced storage requirements for the stiffness matrix (3162 words instead of

9990 words). Accordingly, if one had access to a very small computer it is conceivable that the PCG algorithm would be more effective since it could greatly reduce the number of calls to backing storage. On the otherhand, with access to a virtual memory machine the direct solution is to be preferred.

Example 2. Cylindrical Structures

As a second example we consider a cylindrical structure subjected to end loadings. Two different meshes are considered to illustrate the performance of the PLM algorithm under mesh refinements. The material is again linearly elastic and both static and dynamic loadings are considered. The first mesh consists of 231 nodes and 200 4-node isoparametric elements (type ELMT01), while the refined mesh consists of 496 nodes and 450 elements. The meshes are shown in Figures 2. and 3. Results for the analyses are summarized below.

```
Model: Small Cylinder Structure
      (231 nodes 200 elements)
      profile 17485          Non-zero terms: 3345

Static
        Direct
              total time:  22.50
        PCG
              total time:  30.63    <39 iterations>

Dynamic
(15 time   Direct
 steps)          total time:  83.32
        PCG
              total time: 232.75   <25 to 27 iterations>
```

```
Model: Large Cylinder structure
      (496 nodes 450 elements)
      profile 57280            Non-zero terms: 7570

Static
        Direct
              total time:  71.82
        PCG
              total time: 149.48    <140 iterations>

Dynamic
(5 time   Direct
 steps)         total time: 119.73
        PCG
              total time: 230.37    <31 to 39 iterations>
```

Once again the direct solution is more efficient in CPU, however the storage requirements for the PLM (or PCG) method are significantly less than the direct method. Note that the number of terms is almost directly proportional to the number of nodes (indeed for a regular mesh of 4-node quadrilateral elements the number of nonzero terms in any column is 10 or less), whereas for the direct method the number of terms within the nonzero profile of the matrix is almost proportional to the number of nodes squared! The other significant fact in this example is the number of iterations required to solve the dynamic problem is significantly less than that required for the static loading. Furthermore, for the dynamic loading the number of iterations required to solve the problem increases very little with increased problem size.

Example 3. Three Dimensional Structures

In order to assess the performance of the PCG algorithm on three dimensional problems we have considered the loading on a compact block of 8-node brick isoparametric elements. Two different meshes with linear elastic material properties have been considered. The first mesh consists of 64 elements which are arranged in a regular cube with 4 elements on a side. The mesh has 125 nodes with 21795 words required to store the nonzero profile for a direct solution and only 7455 words required for the PCG method. For the dynamic loading case, this problem produces the first PCG results which are more efficient than a direct solution. The results are summarized below:

```
Model: 4 X 4 X 4 solid structure
      (125 nodes 64 elements)
      profile 21795            Non-zero terms: 7455

Static
        Direct
              total time:  46.65
        PCG
              total time: 201.49    <187 iterations>

Dynamic
(4 time   Direct
 steps)         total time:  93.18
        PCG
              total time:  79.13    <25 to 27 iterations>
```

In order to assess the improvement in performance we constructed a larger problem by subdividing the mesh to form a cube with 8 elements on each edge. Accordingly, the mesh now contains 512 8-node brick elements with a total of 729 nodes. The nonzero profile increases dramatically to 469,071 words whereas, as before, the number of nonzero terms in the compressed profile only increases proportionally to the number of nodes to 60,903 words. The ratio of solution times for the dynamic loading case increases even more for this case, as summarized below; moreover, even the static loading case now requires less CPU for the PCG method than that of the direct solution.

```
Model: 8 X 8 X 8 solid structure
      (729 nodes 512 elements)
      profile 469071            Non-zero terms: 60903

Static
        Direct
              total time:1585.23
        PCG
              total time:1145.29    <130 iterations>

Dynamic
(4 time   Direct
 steps)         total time:2070.32
        PCG
              total time: 320.83    <5 and 6 iterations>
```

This example illustrates the type of improvements which should be attained for all large three dimensional applications. The size of problem we have considered is quite small (indeed even the largest mesh we could consider is only marginally acceptable for simulating very simple geometries) and is limited primarily by the fact that we utilized a VAX 11/780 computer. In double precision arithmetic we required over 4 megabytes of dimensioned memory to solve the problem. We fully anticipate that applications to larger problems on faster ane larger computers can achieve the same level of improvement we have indicated here.

Example 4. Nonlinear Material Response - Two Dimensional Application.

In order to test the performance of the PLM algorithm in a nonlinear application, we considered the elastic-plastic static response of a plane strain strip with a hole. The mesh is shown in Figure 4. and the spread of plastic zone at different load steps in Figure 5. The problem was solved using both direct solution and the PLM method and utilized the consistent tangent formulation developed in [12]. This formulation ensures a quadratic asymptotic rate of convergence when used with a full Newton method. The overall solution time for the PLM method was greater than the direct solution, in accordance with results obtained for Examples 1 and 2. The PLM algorithm performed well, however, and showed no loss in performance with increased plastic deformations. Accordingly, we fully expect that the solution of nonlinear three dimensional problems will be more efficient with the PLM method than a solution achieved using a direct solution of the algebraic equations.

9. Comparison of Global and Element by Element Preconditioned Methods

The global preconditioning of the conjugate gradient or Lanczos method was shown previously to lead to more efficient solution of typical problems [7] than use of element-by-element preconditioning. In order to indicate the number of operations in an element-by-element (EXE) method compared to the global preconditioned form with compressed storage of the array we have constructed a table to indicate the number of operations in a single iteration of each method. For the element-by-element method we assume a second order accurate double pass method (e.g., see [8]). The results are summarized in the table for Examples 1, 2, and 3 cited above.

Example	Mesh	Operations per Iteration	
	Elmt	PCG/PLM	EXE
1	184	12,648	23,552
2	200	13,380	25,600
2	450	30,280	57,600
3	64	29,820	73,728
3	512	243,612	589,824

The difference in the number of operations is due to the fact that each degree-of-freedom in a mesh is associated with more than one element. Indeed, on the average, the above table indicates that there is a savings in number of operations by a ratio of about 1.8 to 2.5 for the PCG/PLM algorithms in comparison with an element-by-element algorithm. Thus, an element-by-element method must converge in about half as many steps in order to be as efficient as the PCG/PLM methods. Our previous experience indicated that element preconditioning never converged in fewer steps that the global preconditioning method; consequently, we believe that the current implementation offers considerable savings over element-by-element methods. The final proof of this assertion must, however, await considerable numerical testing of various implementations for iterative methods.

10. References

[1] Concus, P., G. H. Golub and D. P. O'Leary, "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," *Sparse Matrix Computations,* ed. by J. R. Bunch and D. J. Rose, Academic Press, New York, 1976.

[2] Gourlay, A. R., "Splitting Methods for Time Dependent Partial Differential Equations," *The State of the Art in Numerical Analysis,* ed. by D. Jacobs, Academic Press, New York, 1977.

[3] Kershaw, D. S., "The Incomplete Cholesky-Conjugate Gradient Method for Solution of System of Linear Equations," *Journal of Computational Physics,* vol. 26, pp. 43-65, 1978.

[4] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Bur. Standards,* v. 49, pp. 409-436, 1952.

[5] A. S. Householder, *The Theory of Matrices in Numerical Analysis,* Blaisdell, 1964.

[6] B. Nour-Omid, B. N. Parlett and R. L. Taylor, "A Newton- Lanczos Method for Solution of Nonlinear Finite Element Equations," *Computers and Structures,* vol. 16, No. 1-4, pp. 241-252, 1982.

[7] B. Nour-Omid, C. Rodrigues, and R. L. Taylor, "Solution Techniques in Finite Element Analysis," *Report No. UCB/SESM-83/02,* Structural Engineering and Structural Mechanics, University of California, Berkeley, January 1983.

[8] M. Ortiz, P.M. Pinsky, and R.L. Taylor, "Unconditionally Stable Element-by-Element Algorithms for Dynamic Problems," *Report No. UCB/SESM-82/01,* Structural Engineering and Structural Mechanics, University of California, Berkeley, January 1982.

[9] K. C. Park and J. M. Housner, "Semi-Implicit Transient Analysis Procedures for Structural Dynamic Analysis," *International Journal for Numerical Methods in Engineering,* vol. 18, pp. 609-622, 1982.

[10] B. N. Parlett, *The Symmetric Eigenvalue Problem,* Prentice-Hall, Englewood Cliffs (1980).

[11] B. N. Parlett, "A New Look at the Lanczos Algorithm for Solving Symmetric Systems of Linear Equations," *Linear Algebraic Applications,* vol. 29, pp. 323-346, 1980.

[12] J. C. Simo and R. L. Taylor, "Consistent Tangent Operators for Rate Independent Elasto-Plasticity," *Report No. UCB/SESM-84/03,* Structural Engineering and Structural Mechanics, University of California, Berkeley, February 1984.

[13] O. C. Zienkiewicz, *The Finite Element Method,* Third Edition , Mc-Graw Hill, Inc., London, 1977.

Appendix A. Listing of Compact Stiffness Construction.

The full description of the algorithm to construct the compact storage of the stiffness is described in: "An Algorithm for Assembly of Stiffness Matrices into a Compacted Data Structure," by B. Nour-Omid and R. L. Taylor, Report No. UCB/SESM 84/06, Structural Engineering and Structural Mechanics, University of California, Berkeley, May 1984. The listing follows:

```
      SUBROUTINE ELCNT(NUMNP,NUMEL,NEN,NEN1,IX,IC)
      DIMENSION IX(NEN1,1),IC(1)
C
C.... INPUT
C     NUMNP     TOTAL NO. OF NODES IN THE MESH
C     NUMEL     TOTAL NO. OF ELEMENTS IN THE MESH
C     NEN       MAX. NO. OF NODES PER ELEMENT
C     NEN1      DIMENSION OF IX ARRAY
C     IX        ELEMENT CONNECTIVITY ARRAY
C
C.... OUTPUT
C     IC        ARRAY OF LENGTH NUMNP. IT FIRST HOLDS THE ELEMENT DEGREE
C               OF EACH NODE, THEN BECOMES A POINTER FOR AN ARRAY THAT
C               CONTAINS THE SET OF ELEMENTS CONNECTED TO EACH NODE.
C
C.... COUNT THE NUMBER OF ELEMENTS EACH NODE BELONGS TO
C
      CALL IZERO(IC,NUMNP)
      DO 110 N = 1,NUMEL
         DO 100 J = 1,NEN
            I = IX(J,N)
            IF(I.GT.0) IC(I) = IC(I) + 1
100      CONTINUE
110   CONTINUE
C
C.... SET UP POINTERS
C
      DO 120 I = 2,NUMNP
         IC(I) = IC(I) + IC(I-1)
120   CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE CASSEM(D,A,B,S,P,JCOLE,IROW,LD,ID,NST,NEL,AFL,BFL)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      LOGICAL AFL,BFL
      DIMENSION D(1),A(1),B(1),S(NST,1),P(1),JCOLE(1),IROW(1),LD(1)
     1          ,ID(1)
C
C.... COMPACT ASSEMBLY OF PROFILE MATRIX
C
      DO 200 J = 1,NEL
         N = LD(J)
         IF ( AFL .AND. N .GT. 1 ) THEN
            DO 150 I = 1,NEL
               K = LD(I)
               IF ( K .GT. 0 .AND. K .LT. N ) THEN
                  INZ = INZA(JCOLE(N-1)+1,JCOLE(N),IROW,K)
                  A(INZ) = A(INZ) + S(I,J)
               END IF
150         CONTINUE
         END IF
C....    ASSEMBLE THE DIAGONAL
         IF ( N .GE. 1 ) THEN
            IF ( AFL ) D(N) = D(N) + S(J,J)
C....       ASSEMBLE THE LOAD IF NECESSARY
            IF ( BFL ) B(N) = B(N) + P(J)
         END IF
200   CONTINUE
      RETURN
      END
      SUBROUTINE COMPRO(NUMNP,NUMEL,NEN,NEN1,NDF,IX,ID,IC,IROW,IELC,
     1                  JCOLE,KP)
      DIMENSION IX(NEN1,1),ID(NDF,1),IC(1),IROW(1),IELC(1),JCOLE(1)
C
C     FOR (NUMNP,NUMEL,NEN,NEN1,IX,IC) SEE SUBROUTINE ELCNT
C.... INPUT
C     NDF     NUMBER OF UNKNOWNS AT EACH NODE
C     ID      ACTIVE UNKNOWNS AT EACH NODE
C.... OUTPUT
C     IELC    HOLDS THE SET OF ELEMENTS CONNECTED TO EACH NODE
C     IROW    ROW NUMBER OF EACH NONZERO IN THE STIFFNESS MATRIX
C     JCOLE   END OF ENTRIES IN IROW FORM A GIVEN COLUMN
C
C.... FIND ELEMENTS CONNECTED TO NODES
C
      CALL IZERO (IELC,IC(NUMNP))
      DO 230 N = 1,NUMEL
         DO 220 J = 1,NEN
            I = IX(J,N)
            IF ( I .GT. 0 ) THEN
               KP = IC(I)
200            IF ( IELC(KP) .EQ. 0 ) GO TO 210
                  KP = KP - 1
               GO TO 200
210            IELC(KP) = N
            END IF
220      CONTINUE
230   CONTINUE
C
C.... SET UP COMPRESSED PROFILE POINTERS
C
      KP = 0
      NEP = 1
      DO 350 I = 1,NUMNP
         NE = IC(I)
         DO 340 II = 1,NDF
            NEQ = ID(II,I)
            IF ( NEQ .GT. 0 ) THEN
               JCOLE(NEQ) = KP
               KPO = KP + 1
               IF ( NEP .LE. NE ) THEN
                  DO 330 N = NEP,NE
                     NN = IELC(N)
                     DO 320 J = 1,NEN
                        K = IX(J,NN)
                        DO 310 JJ = 1,NDF
                           NEQJ = ID(JJ,K)
                           IF (NEQJ .GE. NEQ .OR. NEQJ .LT. 0) GO TO 310
C
C....                      CHECK TO SEE IF NODE ALREADY IN LIST
C
                           IF ( KPO .LE. KP ) THEN
                              DO 300 KK = KPO,KP
                                 IF( IROW(KK) .EQ. NEQJ ) GO TO 310
300                           CONTINUE
                           END IF
                           KP = KP + 1
                           IROW(KP) = NEQJ
310                     CONTINUE
320                  CONTINUE
330               CONTINUE
                  JCOLE(NEQ) = KP
               END IF
            END IF
340      CONTINUE
         NEP = NE + 1
350   CONTINUE
      RETURN
      END
      INTEGER FUNCTION INZA(N1,N2,IROW,K)
      DIMENSION IROW(1)
C
C.... FIND THE TERM FOR THE ASSEMBLY
C
      DO 100 N = N1,N2
         IF ( IROW(N) .EQ. K ) THEN
            INZA = N
            RETURN
         END IF
100   CONTINUE
C.... ERROR IF LOOP EXITS
      STOP
      END



      SUBROUTINE IZERO(IA,NN)
      DIMENSION IA(NN)
      DO 100 N = 1,NN
         IA(N) = 0
100   CONTINUE
      RETURN
      END
```

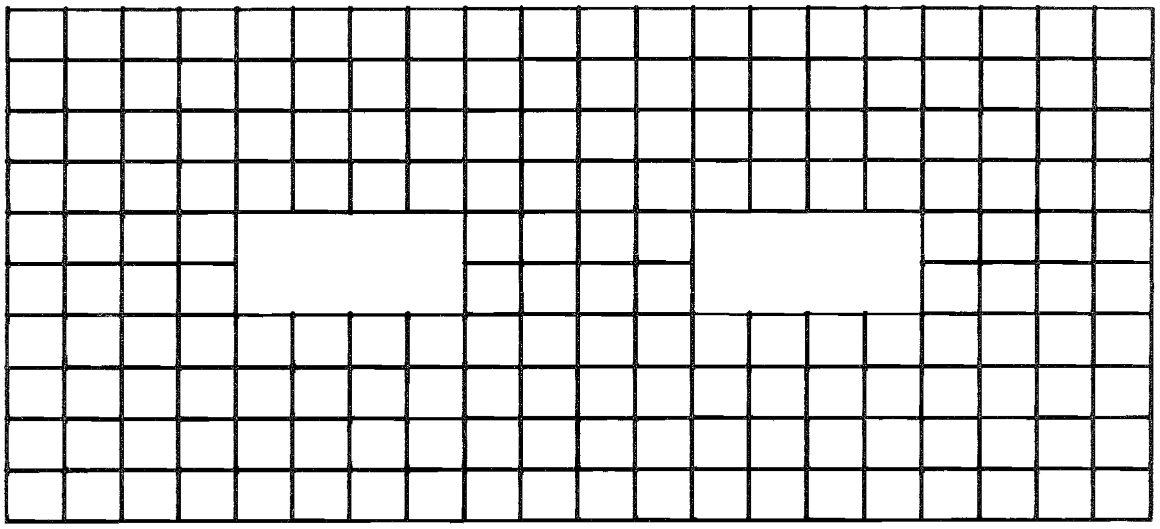

Figure 1. Cantilever Type Structure

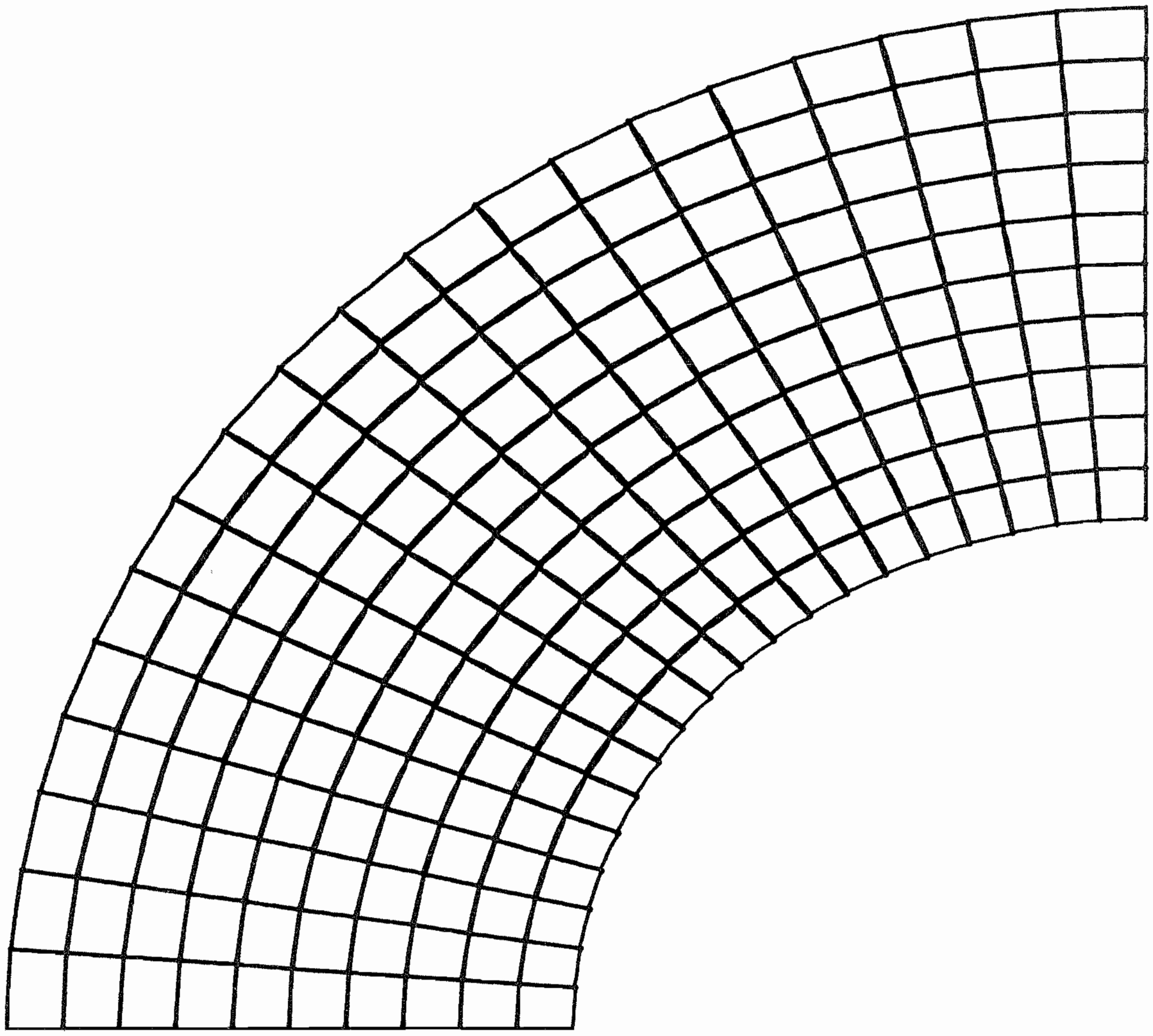

Figure 2. Small Cylinder Structure

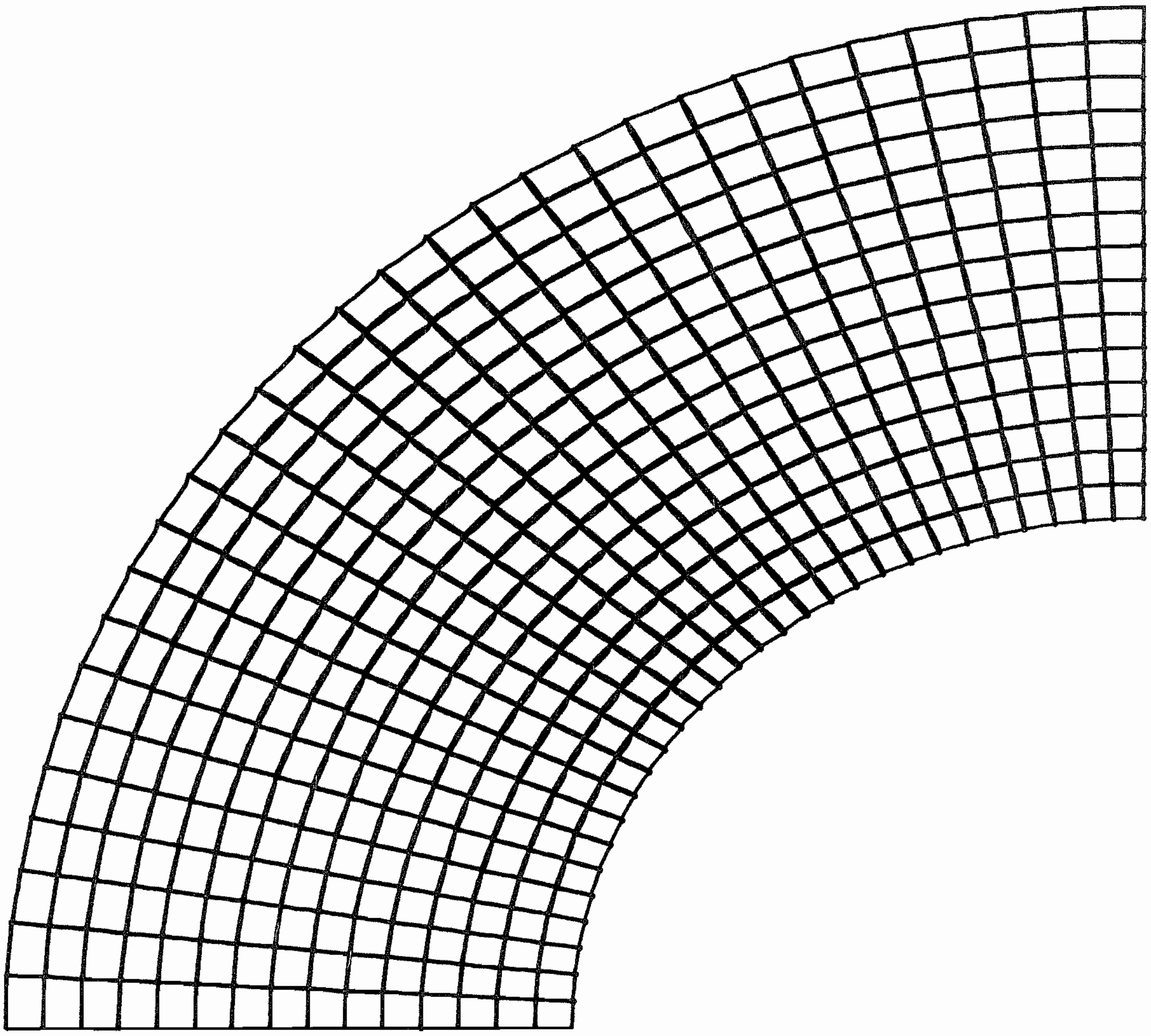

Figure 3. Large Cylinder Structure

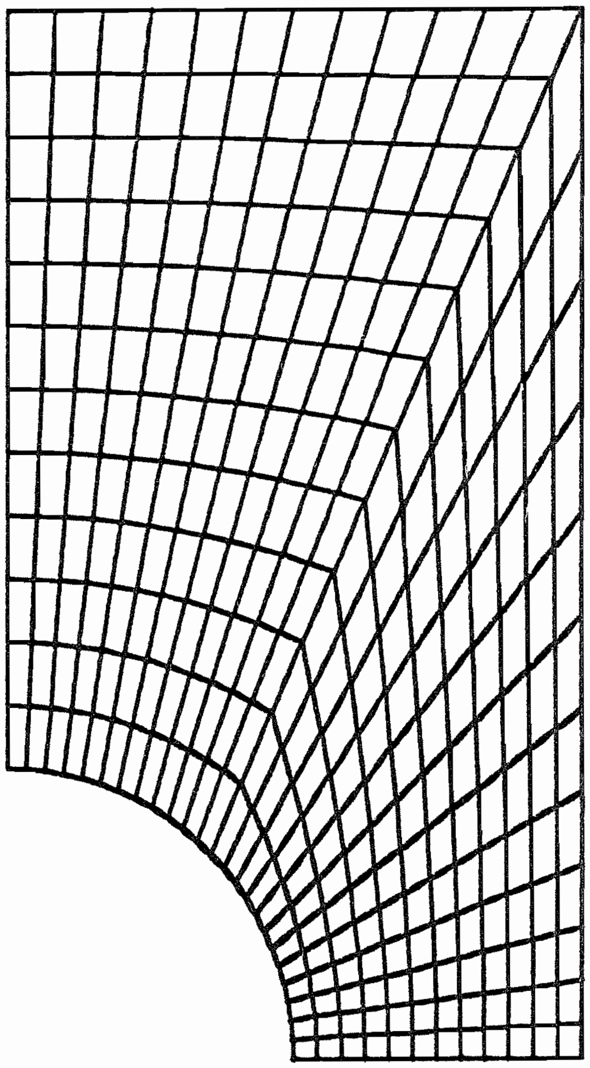

Figure 4. Perforated strip. Finite element mesh.

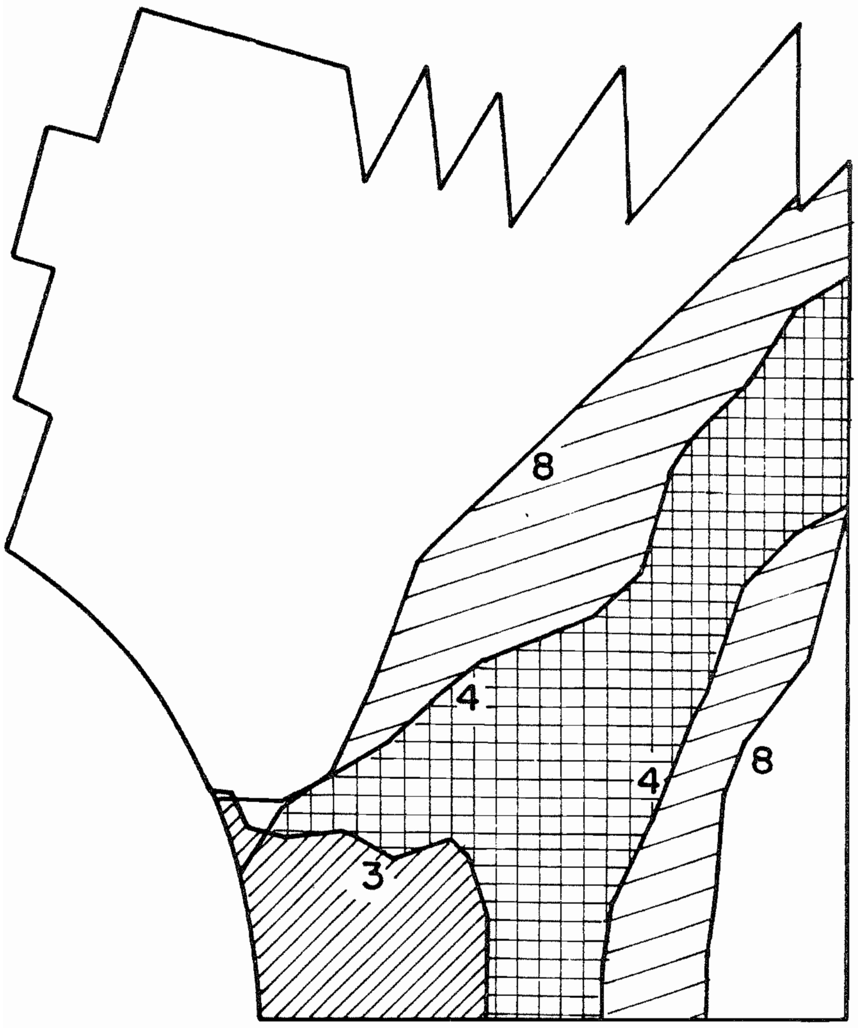

Figure 5. Perforated strip. Elastic-plastic interface.