Structural Engineering and Structural Mechanics Division
Report No. UCB/SESM-80/07

SOLUTION OF NONLINEAR FINITE ELEMENT EQUATIONS BY QUASI-NEWTON METHODS

by

Robert L. Taylor
Professor of Civil Engineering

and

Bahram Nour-Omid
Research Assistant

Department of Civil Engineering
University of California
Berkeley, California 94720

Final Report to

Naval Civil Engineering Laboratory
Naval Construction Battalion Center
Port Hueneme, California

Contract No. N62583/80 M R479

October 1980

1. INTRODUCTION

During the last few years, considerable effort has been directed towards the solution of nonlinear initial-boundary value problems in structural mechanics. The finite element method is one of the more popular approaches employed to reduce continuum problems to nonlinear algebraic, discrete problems. Finite element methodology is well documented in the literature (e.g., see [1]) and attention here is devoted to procedures which may be employed to solve the resulting nonlinear algebraic problem. While the general class of problems of interest include both material and geometric nonlinearities, attention in this report is restricted to problems with material nonlinearity only. In particular, we consider problems whose material behavior is described by elastic/viscoplastic models (e.g., see Perzyna [2]).

In Section 2 we briefly review the application of the finite element to problems in nonlinear continuum mechanics. The result is a large set of nonlinear algebraic equations which must be solved for the state variables (e.g., displacements and stresses). In Section 3 we indicate how Newton's method is normally employed to solve nonlinear equations. We include some discussion on operation count estimates, use of line searches to enhance solution, convergence characteristics, and the advantages and disadvantages associated with Newton's method. In Section 4 we briefly consider modified Newton's method. In general, Newton's method possesses ideal characteristics of convergence and stability but is too expensive to employ in solving large finite element problems. On the other hand, modified Newton's method has desirable operation count estimates but also is too expensive to employ because of its low convergence rate characteristics. Recently, Matthies and Strang [3] have suggested that quasi-Newton methods be used to solve nonlinear finite element problems. In Section 5 we summarize some of the quasi-Newton methods which have been used in optimization methods. We include only the Broyden method and the Broyden-Fletcher-Goldfarb-Shano (BFGS) method in our discussion. A comprehensive review of quasi-Newton methods is given in [4].

In Section 6 we present a summary of our results to date, which were obtained by implementing the program given in [3] in the finite element computer program FEAP described in Chapter 24 of [1]. Preliminary results are very promising, and in Section 7, we conclude with recommendations for work which may be considered for future studies. In particular, we focus attention on methodologies which can be explored to improve cost effectiveness of the quasi-Newton methods. These are primarily associated with update costs and the costs encountered in solution and resolution of large sets of linearized algebraic equations.

2. THE FINITE ELEMENT METHOD FOR NONLINEAR PROBLEMS

In this report we consider problems in structural mechanics which are associated with nonlinear constitutive relations. In particular, we will consider problems modeled by elastic/viscoplastic constitutive equations [2]. In these models the transition between elastic and viscoplastic behavior leads to strong nonlinearities in the algebraic equations. We believe that this problem is a severe test on the applicability of quasi-Newton methods to solve the resulting nonlinear algebraic equations.

We begin with a brief summary of the application of the finite element method to elastic/viscoplastic problems. Consider first the momentum balance equation given by

$$\sigma_{ji,j} + b_i = \rho \ddot{u}_i \tag{1}$$

where σ_{ij} are the components of stress ($\sigma_{ij} = \sigma_{ji}$), u_i are components of displacement, b_i are body forces, ρ is density, $(\ )_{,j}$ denotes partial differentiation with respect to coordinate x, and a superposed dot, $(\dot{\ })$, denotes differentiation with respect to time. A weak form of the momentum balance equations, equivalent to virtual work, may be constructed by multiplying (1) by an arbitrary function W_i, integrating over the domain of interest Ω, using integration by parts on the stress term and setting the result to zero. Accordingly, we obtain

$$\int_{\Omega} (W_i \rho \ddot{u}_i + W_{i,j} \sigma_{ij} - W_i b_i) d\Omega - \int_{\Gamma_2} W_i \bar{t}_i \, d\Gamma = 0 \tag{2}$$

where, in addition to previously defined quantities, $\bar{t}_i$ is a specified traction and Γ_2 is the part of the boundary where traction is specified. For (2) to be valid, W_i must vanish on Γ_1 that part of the boundary where displacements are specified (i.e., $u_i = \bar{u}_i$ on Γ_1). The traction is related to stress through

$$t_i = n_j \sigma_{ij} \tag{3}$$

where n_j are direction cosines of the outward normal to Γ.

The remainder of the problem consists of the strain displacement relations

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \tag{4}$$

in which ϵ_{ij} are the components of strain, a statement of initial conditions

$$\begin{aligned} u_i(x_k,0) &= d_o(x_k) \\ \text{and} \qquad & \\ \dot{u}_i(x_k,0) &= v_o(x_k) \end{aligned} \tag{5}$$

and establishing constitutive equations. In the sequel, we use an elastic/viscoplastic model of the form (e.g., see [2])

$$\dot{\epsilon}_{ij} = C_{ijkl}\,\dot{\sigma}_{kl} + \gamma\langle\phi(F(\sigma_{kl})\rangle \frac{\partial F}{\partial \sigma_{ij}} \tag{6}$$

where C_{ijkl} are elastic compliances, γ is a fluidity parameter, F is the loading and yield function (i.e., the model is associative) and

$$\langle\phi(F)\rangle = \begin{cases} \phi(F) & \text{for} \quad \phi > 0 \\ 0 & \text{for} \quad \phi \leq 0 \end{cases} \tag{7}$$

In our work we let

$$\phi(F) = (F/F_o)^n \tag{8}$$

with F_0 some reference value of F. Equation (6) is both nonlinear and rate dependent. In order to construct a finite element model, we will write a weak form of (6) in the form

$$\int_\Omega V_{ij}(\dot{\epsilon}_{ij} - C_{ijkl}\,\dot{\sigma}_{kl} - \gamma\langle\phi(F)\rangle \frac{\partial F}{\partial \sigma_{ij}})\,d\Omega = 0 \qquad (9)$$

In this form we will develop a mixed model representation in terms of displacements, u_i, and stresses, σ_{ij}. In a finite element model, we divide the domain Ω into elements Ω_e and use approximations for u_i and σ_{ij} which are C^o-continuous and piecewise continuous, respectively [1]. In each element we let

$$\begin{aligned} \underset{\sim}{u} &= \sum_I N_I(\underset{\sim}{x})\,\underset{\sim}{u}_I(t) \\ &\text{and} \\ \underset{\sim}{\sigma} &= \sum_J M_J(\underset{\sim}{x})\,\underset{\sim}{\sigma}_J(t) \end{aligned} \qquad (10)$$

where $N_I(\underset{\sim}{x})$ and $M_J(\underset{\sim}{x})$ are shape functions in Ω_e which satisfy the C^o and piecewise continuous requirements cited above. The arbitrary weighing functions also are approximated using these shape functions, as

$$\begin{aligned} \underset{\sim}{W} &= \sum_I N_I(\underset{\sim}{x})\,\underset{\sim}{W}_I \\ &\text{and} \\ \underset{\sim}{V} &= \sum_J M_J(\underset{\sim}{x})\,\underset{\sim}{V}_J \end{aligned} \qquad (11)$$

where now $\underset{\sim}{W}_I$ and $\underset{\sim}{V}_J$ are arbitrary parameters. We employ a time-stepping procedure (e.g., Newmark method [5]) to remove the time derivatives in (2) and (9).

In the applications discussed in Section 6, we consider the behavior of elastic/viscoplastic plane frames. Details for these finite element developments are contained in [6]. The result of the application of the finite element method and a time-stepping procedure is a large set of nonlinear algebraic equations for each discrete time, t_n. These equations are written as

$$\underset{\sim}{F}(\underset{\sim}{x}) = \underset{\sim}{0} \tag{12}$$

where $\underset{\sim}{F}$ is the composite of (2) and (9) and $\underset{\sim}{x}$ is the state vector at each time, t_n, which consists of all the nodal displacements, $\underset{\sim}{u}_I(t_n)$, and the element stress parameters, $\underset{\sim}{\sigma}_J(t_n)$.

3. NEWTON'S METHOD

Employing the finite element method and a time-stepping method to discretize the nonlinear structural mechanics problem leads to the large set of nonlinear algebraic equations given by (12). If we write a Taylor formula for (12) which is truncated with the linear term, we obtain

$$\underset{\sim}{F}(\underset{\sim}{x}_{k+1}) \approx \underset{\sim}{F}(\underset{\sim}{x}_k) + \underset{\sim}{F}'(\underset{\sim}{x}_k)\, \underset{\sim}{d}_k \tag{13}$$

where k is an iteration number, $\underset{\sim}{d}_k$ is a change in $\underset{\sim}{x}_k$ called the <u>step direction</u>, and $\underset{\sim}{F}$ is the Jacobian or tangent matrix of $\underset{\sim}{F}$ defined by

$$F'_{ij} = \frac{\partial F_i}{\partial x_j} \tag{14}$$

Newton's method consists of setting (13) to zero, solving for $\underset{\sim}{d}_k$ from

$$\underset{\sim}{F}'(\underset{\sim}{x}_k)\, \underset{\sim}{d}_k = -\underset{\sim}{F}(\underset{\sim}{x}_k) \tag{15}$$

and setting

$$\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + \underset{\sim}{d}_k \tag{16}$$

Newton's method requires the initial guess $\underset{\sim}{x}_o$ to be in a domain $\mathcal{D}$ (called the domain of attraction) such that convergence to a value $\underset{\sim}{x}^*$ in $\mathcal{D}$ where $\underset{\sim}{F}(\underset{\sim}{x}^*)$ is zero will occur. Furthermore, $\underset{\sim}{F}$ must be differentiable in $\mathcal{D}$ and $\underset{\sim}{F}'(\underset{\sim}{x}^*)$ must be nonsingular so that (15) holds. In practice it is often desirable to modify (16) to

$$\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + s_k\, \underset{\sim}{d}_k \tag{17}$$

where s_k is a scalar step size which is used to enhance stability of the algorithm. The value of s_k is determined from a line search as described below.

The algorithm for implementing Newton's method consists of choosing a good initial guess $\utilde{x}_o$ and carrying out the following steps for k = 1,2,... until convergence is achieved:

(i) given $\utilde{x}_k$ compute $\utilde{F}(\utilde{x}_k)$

(ii) compute the Jacobian matrix $\utilde{F}'(\utilde{x}_k)$

(iii) solve $\utilde{F}'(\utilde{x}_k)\ \utilde{d}_k = -\utilde{F}\ (\utilde{x}_k)$ for $\utilde{d}_k$

(iv) compute s_k from a line search

(v) update $\utilde{x}_{k+1} = \utilde{x}_k + s_k\ \utilde{d}_k$

(vi) test for convergence

(vii) terminate if converged or repeat (i) to (vi)

There are several procedures which may be used to terminate the iteration. These include:

(a) $||\utilde{F}(\utilde{x}_k)|| < \epsilon_1 \max_i ||\ \utilde{F}(\utilde{x}_i)\ ||$

(b) $||\utilde{u}_k|| < \epsilon_2 \max_i ||(\utilde{u}_i)||$

(c) $||\utilde{d}_k|| < \epsilon_3 \max_i ||(\utilde{d}_i)||$

(d) $|\utilde{d}_k \cdot \utilde{F}(\utilde{x}_k)|| < \epsilon_4\ \max_i |\utilde{d}_i \cdot \utilde{F}(\utilde{x}_i)|$

where ϵ_i are small positive constants. In our work we used Method (a). Method (d) is computable for symmetric matrices when equations are solved; however, for indefinite Jacobian may not be applicable.

3.1 Line Searches for Newton's Method

In the above presentation we have included a step size, s_k, to make Newton's method more stable. The magnitude of s_k is determined in such a way as to minimize a norm of the residual, $\utilde{F}(\utilde{x}_k)$. A common procedure used to determine s_k is to solve [3,4]

$$G(s_k) = \utilde{d}_k^{\ T}\ \utilde{F}(\utilde{x}_k + s_k\ \utilde{d}_k) = 0 \tag{18}$$

It should be noted that (18) need not be solved accurately since $\utilde{x}_{k+1}$

computed by (17) is, in general, not a solution and the additional function evaluations of $\underset{\sim}{F}$ may be costly in finite element analyses. In fact, when possible, the line search should be omitted. Matthies and Strang suggest computing $G(s_k)$ for s_k equal to 0 and 1 using existing data and then set a threshold value for the need of a line search. The value is called s_{tol} in [3], and we have used an s_{tol} value of 0.9 in all our calculations.

3.2 Operation Counts for Newton's Method

For purposes of subsequent cost comparisons, operation counts for Newton's method are estimated to indicate the relative efforts needed in each step. For an n-dimensional $\underset{\sim}{x}$, the operation count estimates are:

(i)	computation of $\underset{\sim}{F}(\underset{\sim}{x}_k)$	-0(n)
(ii)	computation of $\underset{\sim}{F}'(\underset{\sim}{x}_k)$	-0(n^2)
(iii)	direct solution of equations	
	-triangular decomposition of $\underset{\sim}{F}'$	-0(n^3)
	-forward/backsubstitution	-0(n^2)
(iv)	line search	-0(n)
(v)	update of solution	-0(n)
(vi)	convergence test	-0(n)

While order of magnitude estimates are given, substantial differences in real effort exist between, for example, (i) and (vi). The majority of effort is associated with Steps (i) to (iv), and comparisons between Steps (i) to (iv) are meaningful. There is an order of magnitude increase in Step (ii) over Step (i) or in Step (iii) over Step (ii). Consequently, considerable efficiency can be achieved by eliminating or reducing the number of expensive steps required.

3.3 Convergence of Newton's Method

The rate of convergence of an algorithm will determine how fast the iteration $\underset{\sim}{x}_k$ approaches a solution $\underset{\sim}{x}^*$. An acceptable algorithm must be at least linearly convergent [4]; that is, given a solution $\underset{\sim}{x}^*$, then

$$||\underset{\sim}{x}_{k+1} - \underset{\sim}{x}^*|| \leq \alpha||\underset{\sim}{x}_k - \underset{\sim}{x}^*|| \tag{19}$$

where α is a positive constant less than unity. Although (19) ensures that the error norm is reduced by the factor α in each iteration, to be competitive it is generally acknowledged that an algorithm must have better than linear convergence. When Newton's method has continuously differentiable $\underset{\sim}{F}$ and a solution $\underset{\sim}{x}^*$ in $\mathcal{D}$, then the error norm satisfies the stronger condition

$$\begin{gathered} ||\underset{\sim}{x}_{k+1} - \underset{\sim}{x}^*|| \leq \alpha_k||\underset{\sim}{x}_k - \underset{\sim}{x}|| \\ \alpha_k \rightarrow 0 \end{gathered} \tag{20}$$

which is called super-linear convergence. In addition, for cases where $\underset{\sim}{F}$ is twice differentiable in the neighborhood of $\underset{\sim}{x}^*$, Newton's method is quadratically convergent with

$$||\underset{\sim}{x}_{k+1} - \underset{\sim}{x}^*|| < \beta||\underset{\sim}{x}_k - \underset{\sim}{x}^*||^2 \tag{21}$$

For finite element applications, Newton's method will almost always have at least super-linear, and most problems are such that quadratic convergence will be achieved.

3.4 Advantages and Disadvantages of Newton's Method

Newton's method has at least two very desirable properties:

(i) Any $\underset{\sim}{x}_k$ in $\mathcal{D}$ results in an $\underset{\sim}{x}_{k+1}$ in $\mathcal{D}$; consequently, the method is stable and convergent once any iterate is in $\mathcal{D}$.

(ii) The method possesses at least super-linear convergence and often quadratic convergence [4].

On the negative side, we note that:

(i) If $\mathcal{D}$ is small, then very good initial approximations to x* are required.

(ii) Evaluation of the Jacobian matrix and its triangular decomposition is very costly in large finite element problems.

The requirement of a good initial guess may be avoided in part by using line searches and, for quasi-static problems, an evolution of the load application [1]. In the sequel we address the possibility of reducing computational factorizations of the tangent matrix.

4. MODIFIED NEWTON'S METHOD

For large systems of equations, the main cost in Newton's method is the computation and triangular decomposition of the Jacobian matrix. It is often advocated to use a previously computed and factored Jacobian matrix as an approximation to the current tangent matrix. Such a method is called modified (or simplified) Newton's method. The algorithm is given for k = 1,2,... as

(i) given $\underset{\sim}{x}_k$ compute $\underset{\sim}{F}(\underset{\sim}{x}_k)$

(ii) solve $\underset{\sim}{B}_k\underset{\sim}{d}_k = -\underset{\sim}{F}(\underset{\sim}{x}_k)$, where $\underset{\sim}{B}_k = \underset{\sim}{F}'(\underset{\sim}{x}_1)$; $1 \leq k$

(iii) compute s_k from a line search

(iv) update $\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + s_k\underset{\sim}{d}_k$

(v) test convergence

(vi) terminate if converged or else repeat (i) to (v)

Comparing this algorithm with that for Newton's method, we observe that the $O(n^2)$ operations to compute the Jacobian matrix and the $O(n^3)$ operations to compute the triangular decomposition are avoided. The algorithm still requires $O(n^2)$ operations to perform a resolution using the triangular factors of $\underset{\sim}{B}_k$. However, these savings are achieved at the expense of the convergence rate since the modified Newton's method only converges linearly, as given by (19). There will exist some number of iterations p such that computation of a new Jacobian matrix will make the modified Newton's method most efficient. Unfortunately, the value of p is problem dependent (depends on the degree of problem nonlinearity) and cannot be estimated easily.

In the next section we consider quasi-Newton methods which replace the exact calculation of the tangent matrix by an update of the previous tangent (or its inverse). It will be shown that many quasi-Newton methods preserve the desirable $O(n^2)$ operation count of the modified Newton's method but can be constructed to have super-linear convergence. Modified Newton's method is not cost competitive with quasi-Newton methods due to the difference in rate of convergence.

5. QUASI-NEWTON METHODS

Quasi-Newton methods are a generalization of the one-dimensional secant method to the n-dimensional problem. In the secant method, an approximation to $\underset{\sim}{F}'(\underset{\sim}{x}_k)$ (i.e., $\underset{\sim}{B}_k$) is used for each iteration. This concept is applied in multi-dimensions and a simple updating algorithm is deduced to compute the new $\underset{\sim}{B}_k$ from the previous value $\underset{\sim}{B}_{k-1}$. The starting matrix $\underset{\sim}{B}_o$ is normally taken as $\underset{\sim}{F}'(\underset{\sim}{x}_o)$; however, other choices are possible (e.g., finite difference approximations [4]). The convergence rate for all practical quasi-Newton methods is super-linear, and the number of numerical operations for each iteration not requiring a new Jacobian (e.g., a $\underset{\sim}{B}_o$) is $O(n^2)$.

To deduce the "secant equation," we write a linear backward Taylor formula·

$$\underset{\sim}{F}(\underset{\sim}{x}_k) \approx \underset{\sim}{F}(\underset{\sim}{x}_{k-1}) - \underset{\sim}{F}'(\underset{\sim}{x}_k)(\underset{\sim}{x}_k - \underset{\sim}{x}_{k-1}) \tag{22}$$

If we use (22) as an equation to deduce the approximate Jacobian $\underset{\sim}{B}_k$, we can write

$$\underset{\sim}{B}_k \, \underset{\sim}{d}_{k-1} = \underset{\sim}{y}_{k-1} \tag{23}$$

where

$$\underset{\sim}{d}_{k-1} = \underset{\sim}{x}_k - \underset{\sim}{x}_{k-1} \tag{24}$$

and

$$\underset{\sim}{y}_{k-1} = \underset{\sim}{F}(\underset{\sim}{x}_k) - \underset{\sim}{F}(\underset{\sim}{x}_{k-1}) \tag{25}$$

The values in (24) and (25) must be computed to perform the Newton step

$$\underset{\sim}{B}_k \, \underset{\sim}{d}_k = -\underset{\sim}{F}(\underset{\sim}{x}_k) \tag{26}$$

and (23) is called the quasi-Newton equation, which must be satisfied by $\mathbf{B}_k$. In a one-dimensional problem, (23) is sufficient to determine $\mathbf{B}_k$ completely; however, for multi-dimensional problems additional information is required to specify $\mathbf{B}_k$. This gives rise to many possibilities for defining $\mathbf{B}_k$, and the best one for each class of problems probably will be determined only after considerable numerical experimentation. In the following sectins we will summarize two possibilities, Broyden's Method and the BFGS Method.

5.1 Broyden's Method

In 1965 Broyden proposed a method for the approximate specification of $\mathbf{B}_k$ by a simple update of the previous value. To compute $\mathbf{B}_k$, Broyden assumed that $\mathbf{B}_k$ does not differ from $\mathbf{B}_{k-1}$ when acting on any vector orthogonal to $\mathbf{d}_{k-1}$. Accordingly,

$$\mathbf{B}_k \mathbf{z} = \mathbf{B}_{k-1} \mathbf{z}; \quad \mathbf{z}^T \mathbf{d}_{k-1} = 0 \tag{27}$$

Not only does (27) give an update relation for $\mathbf{B}_k$, but when combined with (23), a unique specification for $\mathbf{B}_k$ will result. Broyden's method is given by

$$\mathbf{B}_k = \mathbf{B}_{k-1} + \frac{(\mathbf{y}_{k-1} - \mathbf{B}_{k-1} \mathbf{d}_{k-1}) \mathbf{d}_{k-1}^T}{\mathbf{d}_{k-1}^T \mathbf{d}_{k-1}} \tag{28}$$

Multiplying (28) by $\mathbf{d}_{k-1}$ and $\mathbf{z}$ gives (23) and (27), respectively, thus demonstrating the applicability of (28). If we use (26) in (28), we may write the computationally more attractive form

$$\mathbf{B}_k = \mathbf{B}_{k-1} + \frac{\mathbf{F}(\mathbf{x}_k) \mathbf{d}_{k-1}^T}{\mathbf{d}_{k-1}^T \mathbf{d}_{k-1}} \tag{29}$$

Broyden's method is super-linearly convergent [4]. Given an initial $\underset{\sim}{x}_o$ and $\underset{\sim}{B}_o$ we can use the available information to perform the quasi-Newton step. The algorithm would be identical to that for Newton's method, except the computation of the Jacobian would be replaced by (29). The factorization of $\underset{\sim}{B}_k$ at each step is still required; consequently, it is best to directly update the inverse of $\underset{\sim}{B}_{k-1}$ to obtain the inverse of $\underset{\sim}{B}_k$. This will eliminate the need to compute the triangular decomposition of $\underset{\sim}{B}_k$ and will result in an algorithm with $O(n^2)$ operations in each step.

5.2 Computation of the Inverse of $\underset{\sim}{B}_R$

The inverse of matrices defined similar to (29) may be written as

$$(\underset{\sim}{B} + \underset{\sim}{v}\underset{\sim}{w}^T)^{-1} = \underset{\sim}{B}^{-1} + \frac{1}{\sigma} \underset{\sim}{B}^{-1} \underset{\sim}{v}\underset{\sim}{w}^T \underset{\sim}{B}^{-1} \tag{30}$$

where $\sigma = 1 + \underset{\sim}{w}^T \underset{\sim}{B}^{-1} \underset{\sim}{v}$.

If we let $\underset{\sim}{H}_k$ be the inverse of $\underset{\sim}{B}_k$, Broyden's method for the update of the inverse becomes

$$\underset{\sim}{H}_k = \underset{\sim}{H}_{k-1} + \frac{(\underset{\sim}{d}_{k-1} - \underset{\sim}{H}_{k-1} \underset{\sim}{y}_{k-1}) \underset{\sim}{d}^T_{k-1} \underset{\sim}{H}_{k-1}}{\underset{\sim}{d}^T_{k-1} \underset{\sim}{H}_{k-1} \underset{\sim}{y}_{k-1}} \tag{31}$$

provided $\underset{\sim}{d}^T_{k-1} \underset{\sim}{H}_{k-1} \underset{\sim}{y}_{k-1}$ is nonzero. Broyden's method may be implemented as

$$\underset{\sim}{d}_k = -\underset{\sim}{H}_k \underset{\sim}{F}(\underset{\sim}{x}_k) \tag{32}$$

and requires two resolves per iteration (which may be computed as two right-hand sides simultaneously).

5.3 Convergence of Quasi-Newton Methods

Convergence properties of quasi-Newton methods are discussed by Dennis and More in [4]. In this study they restate an earlier result that an iterative method is super-linearly convergent if

$$\lim_{k\to\infty} \frac{\|[B_k - F'(x^*)](x_{k+1} - x_k)\|}{\|x_{k+1} - x_k\|} = 0 \qquad (33)$$

If B_k converges to $F'(x^*)$, as for Newton's method where B_k equals $F'(x_k)$, then convergence is always super-linear. However, the important result of (33) is that when B_k only converges to $F'(x^*)$ in the direction d_k, convergence is also super-linear. Both the Broyden and the Broyden-Fletcher-Goldfarb-Shano quasi-Newton methods (see below) satisfy (33) for continuously differentiable F and are thus super-linearly convergent.

5.4 Methods for Symmetric Positive Definite Jacobians

Brodlie, Gourlay and Greenstadt have shown that certain rank-one and rank-two corrections to symmetric positive definite matrices may be conveniently expressed in the product form [7]

$$H_k = (I + w_k v_k^T) H_{k-1} (I + v_k^T w_k) \qquad (34)$$

Matthies and Strang have demonstrated that this form has both the advantages of preserving symmetry and positive definiteness as well as computational advantages in the updating procedure. In [3] the algorithm is related to the BFGS algorithm, which gives

$$w_k = \frac{1}{d_{k-1}^T y_{k-1}} d_{k-1} \qquad (35)$$

and

$$v_R = F(x_{k-1}) \left[1 - \left(\frac{d_{k-1} y_{k-1}}{d_{k-1}^T F(x_{k-1})} \right)^{\frac{1}{2}} \right] - F(x_k) \qquad (36)$$

The computational steps for implementing the BFGS method are very straight-forward and consist of solving (32) with the following steps:

(i) for each k compute $\underset{\sim}{F}(\underset{\sim}{x}_k)$ and $\underset{\sim}{z}_k = -(\underset{\sim}{I} + \underset{\sim}{v}_k \underset{\sim}{w}_k^T)\, \underset{\sim}{F}(\underset{\sim}{x}_k)$

(ii) solve $\underset{\sim}{H}_{k-1}^{-1}\, \underset{\sim}{u}_k = \underset{\sim}{z}_k$

(iii) compute $\underset{\sim}{d}_k = (\underset{\sim}{I} + \underset{\sim}{w}_k \underset{\sim}{v}_k^T)\, \underset{\sim}{u}_k$

(iv) if required, compute a line search for s_k

(v) update $\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + s_k \underset{\sim}{d}_k$

(vi) check convergence

It should be noted that $\underset{\sim}{H}_{k-1}$ may result from a previous BFGS update. Thus, Steps (i) and (iii) may require several updates before the resolve step is performed. Furthermore, this form of the algorithm is strictly limited to positive definite tangent matrices. Indefinite forms resulting from Lagrange multiplier constraints may be considered by BFGS but an alternative implementation is required (see [3] and [4]).

6. APPLICATION OF THE BFGS METHOD TO PLANE FRAME PROBLEMS IN VISCOPLASTICITY

The BFGS algorithm described in [3] has been implemented in the finite element analysis program FEAP, which is described in Chapter 24 of [1]. The Algol program given in [3] was closely followed and translated into a FORTRAN module for FEAP. A listing for this module is included in the Appendix. This program is experimental and undoubtedly several improvements may be made to improve the computational performance. For subsequent comparisons between Newton's, modified Newton's, and quasi-Newton methods, we will only consider the number of iterations required to reach a convergenced solution. This is justified because of the rather small problems we have considered to date and possible inefficiencies that result in using the research-oriented program FEAP.

The first example we consider is a plane frame subjected to the single load, as shown in Fig. 1. The load is large enough to cause inelastic moments at both ends and directly under the load on the girder. The problem is solved using Newton's, modified Newton's and the BFGS quasi-Newton methods. Convergence was based on the residual force vector with ϵ_1 equal to 10^{-4}. For the first three time steps, the number of iterations required for each method are summarized in Table 1. No line searches are used for the Newton or the modified Newton methods. It is evident in Table 1 that the quasi-Newton method is effective, especially for the first step where the initial guess of zero is very far from the solution $\underset{\sim}{x}^*$. At later stages there is very little difference since only small changes in the solution occur and the initial tangent is good for all methods.

Time Step	Method		
	Newton	Modified Newton	BFGS Quasi-Newton
1	5	17	10
2	4	5	4
3	3	4	3

TABLE 1. NUMBER OF ITERATIONS REQUIRED FOR PLANE FRAME EXAMPLE

As a second example, we consider the dynamic loading of a simply supported beam. The load is applied at the beam center with a value

$$P(t) = \begin{cases} P_0 \sin^2 a_0 t; & 0 < t < \pi/a_0 \\ 0 & ; \ \pi/a_0 < t \end{cases}$$

The beam undergoes both loading and unloading, and the entire beam behaves inelastically during a portion of the analysis. Table 2 indicates the number of iterations required to achieve tolerances for the residual forces of 10^{-4} and 10^{-5} of the maximum within each step. The importance of super-linear convergence is evident by comparing the differences in the total iterations to achieve one additional digit of accuracy over the entire first 15 time steps for each method. Newton's method requires only 6 iterations more to achieve the extra digit, the BFGS quasi-Newton method requires 7 iterations, while the modified Newton solution requires 12 iterations. The most significant difference occurs in the 14th time step where considerable unloading occurs. The effectiveness of the quasi-Newton method is evident in this step.

Time Step	Newton		Modified Newton		BFGS Quasi-Newton	
	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	10^{-5}
1	2	2	2	2	2	2
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	3	3	3	4	3	4
5	4	4	5	6	4	5
6	4	4	5	6	5	5
7	4	5	5	5	5	5
8	4	5	5	6	5	5
9	4	4	4	5	4	5
10	4	4	4	5	4	5
11	4	4	4	5	4	5
12	3	4	5	6	4	5
13	3	4	4	5	4	4
14	4	5	7	8	6	6
15	4	5	5	7	5	6
Total	52	58	63	75	60	67

TABLE 2. NUMBER OF ITERATIONS FOR DYNAMIC LOADING OF BEAM EXAMPLE

7. CLOSURE AND RECOMMENDATIONS FOR ADDITIONAL STUDIES

In this report we have summarized some of our preliminary experiences in applying quasi-Newton methods to nonlinear finite element equations for viscoplasticity problems. The results are quite encouraging and support those reported previously in [3]. The advantages of the super-linear convergence of the BFGS method is clearly evident in both the quasi-static analysis and the dynamic analysis we performed. The added effort over that required for modified Newton methods is negligible for any significant finite element analysis.

At the present time, quasi-Newton methods to solve nonlinear finite element equations have been restricted to those with positive definite Jacobians. We attempted to apply our implementation to a simple contact problem where the contact condition is enforced by Lagrange multiplier constraints on those nodes indicating penetration and/or compressive contact loads. The algorithm failed to converge. At this time it is not clear whether this is due to the particular implementation of BFGS we used (e.g., see [3] for one possible alternative not restricted to positive definite Jacobians) or due to lack of differentiability in the nonlinear Lagrange multiplier equations for the contact condition. Needless to say, modified Newton's method fails for this problem, too. However, Newton's method is successful.

The current implementation of the quasi-Newton BFGS algorithm requires occasional computation and factorization of the Jacobian matrix. For large problems this can still remain the most costly step in the analysis. Techniques to avoid this step should be explored. Recently, much effort has been directed to the solution of linear equations by iterative methods. One such method is the Lanczos algorithm as proposed by Parlett [8], which is related to the conjugate gradient method. Iterative methods are competitive only with proper preconditioning of the equation; techniques to precondition nonlinear finite element equations need to be investigated to find methods which are cost effective when used with particular iterative methods (e.g., use of incomplete Choleski factorization with the conjugate gradient method enhances convergence). The Lanczos method consists of representing a coefficient matrix $\underset{\sim}{B}$ in terms of an orthogonal matrix $\underset{\sim}{Q}$ and a tridiagonal matrix $\underset{\sim}{T}$ as

$$\underset{\sim}{B}\,\underset{\sim}{Q} = \underset{\sim}{Q}\,\underset{\sim}{T}$$

where

$$\underset{\sim}{Q}^T \underset{\sim}{Q} = \underset{\sim}{I}$$

The possibility of directly updating $\underset{\sim}{T}$ (with sparse updates) instead of Broyden or product updates of $\underset{\sim}{B}_k$ or $\underset{\sim}{H}_k$ is attractive. It would combine the advantages of traditional iterative methods with those of the quasi-Newton method. Eventual success of such endeavors would ultimately depend on how many vectors must be used for columns of the orthogonal matrix $\underset{\sim}{Q}$.

Other directions to pursue are the one-step BFGS forms of Crisfield [9]. This work uses simple one-step BFGS updates of modified Newton iterates, requires very little extra effort and very little additional storage. It may be particularly effective for dynamic problems with significant inertia forces. The effectiveness of Crisfield's approach should be assessed for finite element equations resulting from nonlinear material (and geometric) behavior.

It is evident that the application of quasi-Newton methods to nonlinear finite element equations presents several alternative approaches.

The surface has just been scratched and much additional effort is required before the most effective methods are delineated. The result of studies in this direction should be algorithms which are not only cheaper but are more stable and reliable than those in use today.

REFERENCES

1. O. C. Zienkiewicz, The Finite Element Method, 3rd Edition, McGraw-Hill Book Co., London, 1977.

2. P. Perzyna, "Fundamental Problems in Viscoplasticity," Adv. Appl. Mech., v. 9, pp. 243-377, 1966.

3. H. Matthies and G. Strang, "The Solution of Nonlinear Finite Element Equations," Int. J. Num. Meth. Engr., v. 14, pp. 1613-1626, 1979.

4. J. R. Dennis, Jr. and J. J. More, "Quasi-Newton Methods, Motivation and Theory," SIAM Review, v. 19, pp. 46-86, 1977.

5. N. M. Newmark, "A Method of Computation for Structural Dynamics," J. Eng. Mech. Div., ASCE, V. 85, EM3, pp. 67-94, 1959.

6. J. Slater, "Mixed-Model Finite Element Analysis of Inelastic Plane Frames," CE299 Report, SESM Division, Department of Civil Engineering, University of California, Berkeley, 1980.

7. K. W. Brodlie, A. R. Gourlay and J. Greenstadt, "Rank-One and Rank-Two Corrections to Positive Definite Matrices Expressed in Product Form," J. Inst. Math. Appl., v. 11, pp. 73-82, 1973.

8. B. N. Parlett, "A New Look at the Lanczos Algorithm for Solving Symmetric Systems of Linear Equations," in Linear Algebraandits Applications, Elsevier North Holland Press, pp. 323-346, 1980.

9. M. A. Crisfield, Letter to the Editor, Int. J. Num. Meth. in Engr., v. 15, pp. 1419-1420, 1980.

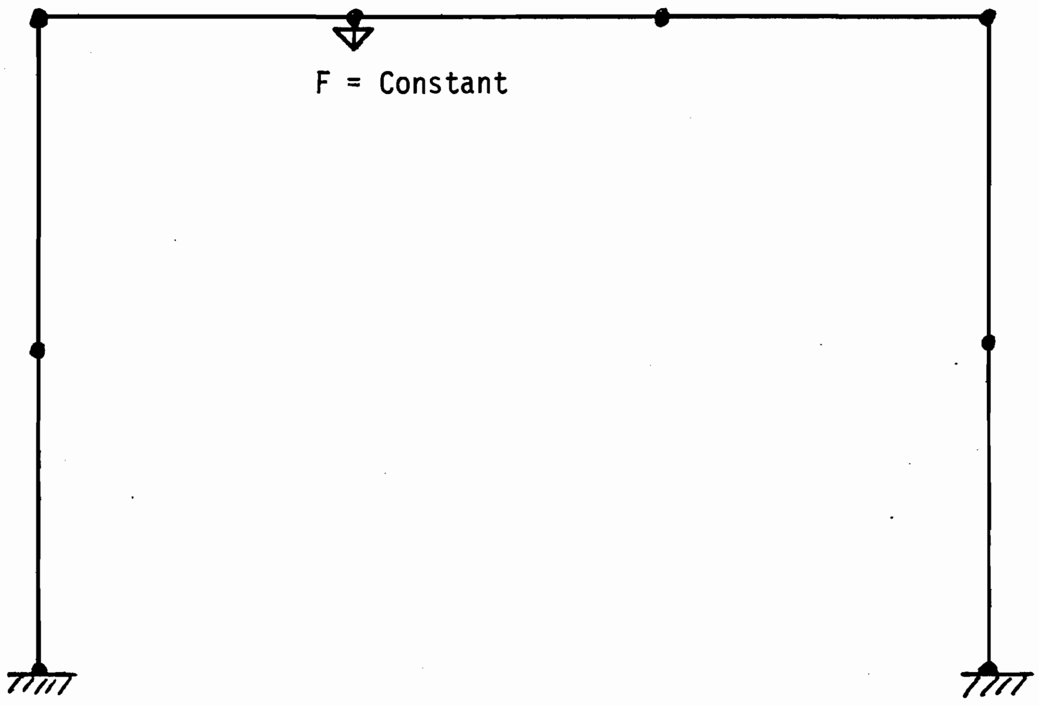

Figure 1. Quasi-Static Analysis of Plane Viscoplastic Frame.

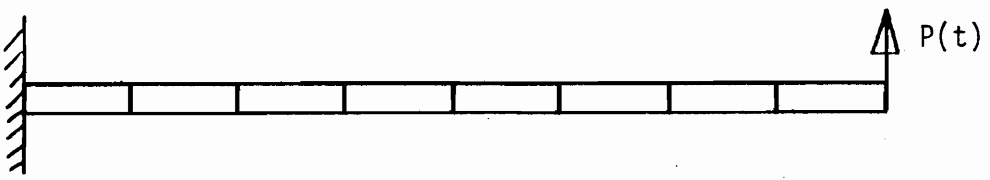

Figure 2. Dynamic Analysis of Viscoplastic Beam.

Appendix

Listing of modules added to FEAP to implement BFGS method and the viscoplastic frame element used in the analysis.

```
Oct 13 16:42 1980  iterat.f Page 1

      subroutine iterat(stiff,jdiag,u,rsd,oldrsd,d,utol,ftol,neq,
     1                  accrcy,v,w,prt)
      implicit double precision(a-h,o-z)
c
c.... the routine will return the solution and residual to
c.... the equation K(u)*u - b = 0 . The routine evaluates the
c.... RHS of the equation through the routine operat.
c
      common/index/iform
      common/update/updt
      dimension stiff(1),jdiag(1),u(1),rsd(1),oldrsd(1),d(1)
      dimension v(1),w(1)
      logical accrcy,prt,updt
c.... tolerance for line search
      iform=0
      stol=0.9d0
      accrcy=.false.
      itmax=15
      nupd=0
      s=1.0d0
      g0=0.0d0
      g=0.0d0
      q1=0.0d0
      q2=0.0d0
      oldnrm=1.0d0
c.... initialization
      call pzero(u,neq)
      call pzero(oldrsd,neq)
      call pzero(d,neq)
      call operat(rsd,u,neq)
      updt=.false.
      rnmax = sqrt(dot(rsd,rsd,neq))
      dnmax = 0.0d0
      iform=iform+1
c.... loop for equilibrium iteration
      do 100 i=1,itmax
      if(.not.prt) go to 30
      write(6,2000)  i
      do 20 l=1,neq
20    write(6,2001) l,u(l),rsd(l),oldrsd(l),d(l)
30    continue
c.... compute the search direction
      call dfind(stiff,jdiag,d,u,rsd,oldrsd,nupd,g0,g,s,dnorm,neq,v,w)
c.... do line search if necessary
      s=1.0d0
      do 50 j=1,neq
50    w(j)=u(j)-d(j)
      call operat(rsd,w,neq)
      iform=iform+1
      g0=dot(d,oldrsd,neq)
      g=dot(d,rsd,neq)
      chek=stol*dabs(g0)
      if(dabs(g).gt.chek) call serchl(g0,g,rsd,u,d,stol,w,neq,s)
c.... increment u
      check=-s
```

```
Oct 13 16:42 1980  iterat.f Page 2

c.... compute norms and contraction factor q
      dnorm=sqrt(dot(d,d,neq))
      dnorm=dnorm*s
      dnmax = dmax1(dnmax,dnorm)
      rnorm=sqrt(dot(rsd,rsd,neq))
      rnmax = dmax1(rnmax,rnorm)
      q2=q1
      q1=0.0d0
      if(i.gt.1) q1=dnorm/oldnrm
      q=q1
      if(q1.lt.q2) q=q2
      oldnrm=dnorm
      chek=dabs((1.0d0-q)*utol)
c.... check for accuracy
      accrcy=(rnorm.lt.ftol*rnmax).and.(dnorm*dabs(q).lt.chek*dnmax)
      if(accrcy) go to 900
100   continue
900   write(6,2002) iform,rnmax,rnorm
      write(6,2003) i
      return
2000  format(4x,'start of iteration no. ',i5/
     1       4x,'d.o.f.',5x,'u vector',5x,'rsid. vector',4x,'oldrisidual
     2',4x,'search vector')
2001  format(2x,i5,4d16.5)
2002  format(2x,'**macro instruction form executed by bfgs',i3,' times',
     1       /6x,'rnmax = ',g14.5,'    rnorm = ',g14.5)
2003  format(2x,i3,' iterations required to converge')
      end
```

Oct 13 16:42 1980 serchl.f Page 1

```
      subroutine serchl(g0,g,rsd,u,d,stol,t,neq,step)
      implicit double precision(a-h,o-z)
      common/index/iform
      dimension rsd(1),u(1),d(1),t(1)
c.... this routine makes a line search in direction d and returns the
c.... steplength step
      linmax=10
      smax=16.0d0
      sb=0.0d0
      sa=1.0d0
      gb=g0
      ga=g
c.... find bracket on zero
10    a=ga*gb
      if(a.lt.0.0d0.or.sa.gt.smax) go to 30
      sb=sa
      sa=sa+sa
      gb=ga
      do 20 i=1,neq
20    t(i)=u(i)-sa*d(i)
      call operat(rsd,t,neq)
      iform=iform+1
      ga=dot(d,rsd,neq)
      go to 10
30    continue
      step=sa
      g=ga
c.... illinois algorithm to find zero
      do 100 i=1,linmax
      a=ga*gb
      b=stol*dabs(g0)
      ca=dabs(sb-sa)
      cb=0.5*stol*(sb+sa)
      if(a.ge.0.0d0.or.(dabs(g).lt.b.and.ca.lt.cb)) return
      step=sa-ga*(sa-sb)/(ga-gb)
      do 120 j=1,neq
120   t(j)=u(j)-step*d(j)
      call operat(rsd,t,neq)
      iform=iform+1
      g=dot(d,rsd,neq)
      a=g*ga
      if(a.gt.0.0d0) go to 130
      gb=ga
      sb=sa
      go to 140
130   gb=0.5*gb
140   continue
      sa=step
      ga=g
100   continue
      return
      end
```

Oct 13 16:42 1980 dfind.f Page 1

```
      subroutine dfind(stiff,jdiag,d,u,rsd,oldrsd,nupd,g0,g,s,dnorm,neq
     1                 ,v,w)
      implicit double precision(a-h,o-z)
      dimension stiff(1),jdiag(1),d(1),u(1),rsd(1),oldrsd(1),v(1),w(1)
      logical up
c.... this routine finds a new search direction using bfsg updating
c.... method in factored form
      maxup=15
      condmx=1.0d5
c
      delgam=s*(g0-g)
      dlkdl=s*s*g0
      do 50 i=1,neq
50    v(i)=rsd(i)
      up=delgam.gt.0.0d0.and.dlkdl.gt.0.0d0
      if(.not.up) go to 200
      fact1=1.0d0+s*sqrt(delgam/dlkdl)
      fact2=-s/delgam
c.... compute updating vectors and put residual into d
      do 100 i=1,neq
      aux=rsd(i)
      v(i)=fact1*oldrsd(i)-aux
      w(i)=d(i)
      d(i)=aux
100   oldrsd(i)=aux
c.... check estimate on increase of condition number
      vv=dot(v,v,neq)
      ww=dnorm/delgam
      ww=ww*ww
      vw4=4.0*fact2*(fact1*g0-g) + 4.0d0
      up=up.and.vw4.ne.0.0d0
      stcond=0.0d0
      if(.not.up) go to 120
      stcond=vv*ww
      aux=stcond+vw4
      aux=dabs(aux)
      stcond=sqrt(stcond)+sqrt(aux)
      stcond=stcond*stcond/dabs(vw4)
120   continue
      up=up.and.stcond.lt.condmx
      if(.not.up) go to 140
c.... save updating factors,with fact2 to be included later in w
      call store(v,w,fact2,nupd+1,1)
c.... the rightmost factor
      coef=fact2*g
      do 130 i=1,neq
130   d(i)=d(i)+coef*v(i)
140   continue
      go to 300
200   do 210 i=1,neq
      d(i)=rsd(i)
210   oldrsd(i)=rsd(i)
300   continue
      if(nupd.eq.0) go to 325
c.... right half of updatind
      do 320 i=1,nupd
```

```
Oct 13 16:42 1980  dfind.f Page 2

      ii=nupd-i+1
      call store(v,w,fact2,ii,2)
      coef=fact2*dot(w,d,neq)
      do 310 j=1,neq
310   d(j)=d(j)+coef*v(j)
320   continue
c.... backsubstitution
325   call actcol(stiff,d,jdiag,neq,.false.,.true.)
      if(up) nupd=nupd+1
      if(nupd.eq.0) go to 350
c.... left half of updating
      do 340 i=1,nupd
      if(i.gt.1) call store(v,w,fact2,i,2)
      coef=fact2*dot(v,d,neq)
      do 330 j=1,neq
330   d(j)=d(j)+coef*w(j)
340   continue
350   nupd=mod(nupd,maxup)
      return
      end
```

```
Oct 13 16:42 1980  stepdu.f Page 1

      subroutine stepdu(u,r,d,s,neq)
      implicit double precision(a-h,o-z)
c
c.... this routine updates the displacement vector u and sets the
c.... updt flag for the updating of stresses in the element routines.
c
      common/update/updt
      logical updt
      dimension u(1),r(1),d(1)
      call padd(u,d,s,neq)
      updt = .true.
      call operat(r,u,neq)
      updt = .false.
      return
      end
```

```
Oct 13 16:42 1980  operat.f Page 1

      subroutine operat(dr,du,neq)
      implicit double precision(a-h,o-z)
      common m(1)
      common /mdata/ nn,n0,n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13
      common /prlod/ prop
      common /itbfgs/ ndf,ndm,nen1,nst,nneq,mt,mst
      common /eldata/ dm,n,ma,mct,iel,nel
      dimension dr(1),du(1)
      call pzero(m(mt),nneq)
      call padd(m(mt),du,1.0d0,neq)
      call padd(m(mt),m(n13),1.0d0,nneq)
      call pload(m(n7),m(n10),m(1),nneq,prop)
      call pform(m(nn),m(n0),m(n1),m(n2),m(n3),m(n4),m(n5),m(n6),m(n7),
     2    m(n8),m(n9),m(n10),m(n11),m(n12),m(1),m(1),m(1),ndf,ndm,nen1,
     3    nst,6,m(mt),m(1),.false.,.true.,.false.,.false.)
      call pzero(dr,neq)
      call padd(dr,m(1),-1.0d0,neq)
      return
      end
```

```
      subroutine padd(a,b,s,n)
      implicit double precision(a-h,o-z)
      dimension a(1),b(1)
      do 100 i=1,n
100   a(i)=a(i)+s*b(i)
      return
      end
```

```
Oct 13 16:42 1980  store.f Page 1

      subroutine store(v,w,fact,np,itrn)
      implicit double precision(a-h,o-z)
      common m(1)
      common /itbfgs/ ndf,ndm,nen1,nst,nneq,mt,mst
      common /cdata/ o,head(20),numnp,numel,nummat,nen,neq,ipr
      dimension v(1),w(1)
      ns=neq+neq+1
      go to (1,2),itrn
1     mtp = mst + np*ns*ipr
      call setmem(mtp)
      mtp = mtp - ns*ipr
      call pcopy(m(mtp),v,neq)
      mtp = mtp + neq*ipr
      call pcopy(m(mtp),w,neq)
      mtp = mtp + neq*ipr
      call pcopy(m(mtp),fact,1)
      return
2     mtp = mst + (np - 1)*ns*ipr
      call pcopy(v,m(mtp),neq)
      mtp = mtp + neq*ipr
      call pcopy(w,m(mtp),neq)
      mtp = mtp + neq*ipr
      call pcopy(fact,m(mtp),1)
      return
      end
```

```
      subroutine pcopy(a,b,nn)
      implicit double precision(a-h,o-z)
      dimension a(1),b(1)
      do 100 i=1,nn
100   a(i) = b(i)
      return
      end
```

Oct 13 16:42 1980 elmt09.f Page 1

```
      subroutine elmt09(d,ul,xl,ix,tl,s,p,ndf,ndm,nst,isw)
      implicit double precision (a-h,o-z)
c.... viscoplastic frame element for feap
c.... mixed model formulation
      dimension d(1),xl(ndm,1),ul(ndf,1),ix(1),tl(1),s(nst,1),p(nst)
      dimension qq(5,150),v(600),qqq(5,5),vv(20)
      logical updt
      common/cdata/o,head(20),numnp,numel,nummat,nen,neq,ipr
      common/eldata/dm,n,ma,mct,iel,nel
      common/tdata/time,dt,c1,c2,c3,c4,c5
      common/update/updt
c.... transfer to correct processor
      go to (1,3,3,3,3,3,7), isw
c.... input the material properties
1     read(5,1000) (d(ii),ii=1,10)
      nnn= d(10)
      write(6,2000) (d(ii),ii=1,9),nnn
      d(1) = 3.*d(1)
      d(2) = 3.*d(2)
      d(3) = d(3)/2.
      d(6) = d(6)*d(6)
      if(d(3).ne.0.0d0) call vplas(s,qq,p,d,v,ul,xx,isw)
      call pzero(v,450)
      r1 = 5./6.
      r2 = 2./3.
      return
c.... update loads due to integration of rate equation
2     continue
      r0 = xx/d(1)
      r3 = xx/d(2)
      k = irow
      v(k+16) =  r1*v(k)-v(k+3)/6.-r2*v(k+10)-(v(k+6)-v(k+7)/2.)*r0
      v(k+17) =  r1*v(k+3)-v(k)/6.-r2*v(k+10)-(v(k+7)-v(k+6)/2.)*r0
      v(k+18) = (v(k+1)-v(k+4))/xx + v(k+2) - (v(k+8)-v(k+9)/2.)*r3
      v(k+19) = (v(k+1)-v(k+4))/xx + v(k+5) - (v(k+9)-v(k+8)/2.)*r3
      return
c.... compute the element tangent array
3     continue
      cs = xl(1,2)-xl(1,1)
      sn = xl(2,2)-xl(2,1)
      xx = sqrt(cs*cs+sn*sn)
      cs = cs/xx
      sn = sn/xx
      icol = 5*n-4
      irow = 20*n - 19
      go to (1,2,31,4,5,6,7),isw
c.... compute tangent stiffness based on current iterates.
31    call vplas(s,qqq,p,d,v(irow),ul,xx,isw)
      call global(s,p,cs,sn,6,3,isw)
      if(n.eq.numel) updt = .true.
      return
c.... compute and output the element variables
c     call global(s,ul,cs,sn,6,3,4)
c     call force(ul,v(irow),qq(1,icol),xx)
c.... print internal forces for all elements.
4     mct = mct - 1
```

Oct 13 16:42 1980 elmt09.f Page 2

```
      if(mct.gt.0) go to 41
      write(6,2002) o,head
      mct = 50
c     do 81 n = 1,numel
41    ia = n*20 -13
      xx = -v(ia)
      yy = v(ia+1)
      xn = -v(ia+2)
      xm = v(ia+3)
      write(6,2001) n,ma,xx,yy,xn,xm
c     continue
      return
c.... compute element mass arrays
5     call beamms(s,p,d(7),xl,cs,sn,nst,ndf)
      return
c.... compute the element residual vector
6     call global(s,ul,cs,sn,6,3,4)
      call swap(v(irow),vv,20,1)
      call swap(qq(1,icol),qqq,5,5)
      call force(ul,vv,qqq,xx)
      call vplas(s,qqq,p,d,vv,ul,xx,isw)
      call global(s,p,cs,sn,6,3,isw)
      if(.not.updt) return
      call swap(vv,v(irow),20,1)
      call swap(qqq,qq(1,icol),5,5)
      return
7     call plotl(xl(1,1),xl(2,1),0.,3)
      call plotl(xl(1,2),xl(2,2),0.,2)
      return
c.... formats
1000  format(8f10.0)
2000  format(5x,'viscoplastic frame element ',/10x,'material constants',
     1      /5x,'e*area    =',e15.5/10x,'e*inertia=',e15.5/
     2      10x,'gamma     =',e15.5/10x,'alpha     =',e15.5/
     3      10x,'m zero    =',e15.5/10x,'height    =',e15.5/
     4      10x,'p yield   =',e15.5/10x,'beta      =',e15.5/
     5      10x,'p zero    =',e15.5/10x,'power     =',i11)
2001  format(2i10,6e13.4)
2002  format(a1,20a4//5x,21hbeam element stresses//10h   element,6x,4hma
     1tl,4x,7h1-force,6x,7h2-force,5x,8h1-moment,5x,8h2-moment/)
      end
      subroutine vplas(s,q,p,d,v,u,xl,isw)
      implicit double precision (a-h,o-z)
c.... compute rate dependent force and moment for mixed model frame element
      common/eldata/dm,n,ma,mct,iel,nel
      common/tdata /time,dt,c1,c2,c3,c4,c5
      dimension q(5,5),p(1),d(1),v(1),gx(4),w(4),b1(4),b2(4),b11(4),
     1   b12(4),b22(4),u(1),s(6,6)
      double precision mx,m0,m1,m2
      if(isw.ne.1) go to 10
c.... compute gauss-lobatto points and weights.
1     gx(1) = 0.
      gx(2) = (5.-sqrt(5.))/10.
      gx(3) = 1.-gx(2)
      gx(4) = 1.0
      w(1) = 1./6.
      w(2) = 5./6.
      w(3) = w(2)
      w(4) = w(1)
      npts = 4
      gx(2) = 0.5
      gx(3) = 1.0
      w(1) = 1./3.d0
      w(2) = 4./3.d0
      w(3) = w(1)
      npts = 3
      do 5 i = 1,npts
      b1(i) = gx(i)-1.0
      b2(i) = gx(i)
      b11(i) = b1(i)*b1(i)
      b12(i) = b1(i)*b2(i)
      b22(i) = b2(i)*b2(i)
5     continue
      return
10    gam = -d(3)*xl*dt
      alp = d(4)
      m0 = d(5)
      hh = d(6)
      y  = d(7)
      bet= d(8)
      p0 = d(9)
      nnn= d(10)
      p1 = v(7)
      p2 = v(8)
      m1 = v(9)
      m2 = v(10)
      fpp = (bet+bet)/y
      fmm = (alp+alp)/y/hh
c.... assemble the elastic compliance terms
      call pzero(q,25)
      z1 = -xl/d(1)
      z2 = -xl/d(2)
      z3 = -2./3.
      q(1,1) = z1
      q(1,2) = -z1/2.
      q(2,2) = z1
      q(3,3) = z2
```

```
      q(3,4) = -z2/2.
      q(4,4) = z2
      q(1,5) = z3
      q(2,5) = z3
      if(isw.eq.3) go to 20
      v(12) = q(1,1)*p1+q(1,2)*p2+q(1,5)*v(11)-v(17)
      v(13) = q(1,2)*p1+q(2,2)*p2+q(2,5)*v(11)-v(18)
      v(14) = q(3,3)*m1+q(3,4)*m2-v(19)
      v(15) = q(3,4)*m1+q(4,4)*m2-v(20)
      v(16) = q(1,5)*(p1+p2)
20    if(gam.ge.0.0) go to 51
      do 50 i = 1,npts
c.... compute moment and axial force at gauss points
      px = p1*b1(i) + p2*b2(i) - p0
      mx = m1*b1(i) + m2*b2(i) - m0
c.... compute value of yield function and derivatives at gauss points
      f = (alp*mx*mx/hh + bet*px*px)/y - y
      if(f.le.0.0d0) go to 50
      phi = (f/y)**nnn
      phif= nnn*phi/f
      fp  = fpp*px
      fm  = fmm*mx
      zz  = gam*w(i)
      if(isw.eq.3) go to 30
      zz1 = zz*phi*fp
      zz2 = zz*phi*fm
      v(12) = zz1*b1(i) + v(12)
      v(13) = zz1*b2(i) + v(13)
      v(14) = zz2*b1(i) + v(14)
      v(15) = zz2*b2(i) + v(15)
30    t = (phi*fpp + phif*fp*fp)*zz
      q(1,1) = b11(i)*t + q(1,1)
      q(1,2) = b12(i)*t + q(1,2)
      q(2,2) = b22(i)*t + q(2,2)
      t = phif*fp*fm*zz
      q(1,3) = b11(i)*t + q(1,3)
      q(1,4) = b12(i)*t + q(1,4)
      q(2,4) = b22(i)*t + q(2,4)
      t = (phi*fmm + phif*fm*fm)*zz
      q(3,3) = b11(i)*t + q(3,3)
      q(3,4) = b12(i)*t + q(3,4)
      q(4,4) = b22(i)*t + q(4,4)
50    continue
      q(2,3) = q(1,4)
51    do 52 i = 1,4
      ii = i + 1
      do 52 j = ii,5
52    q(j,i) = q(i,j)
c.... invert compliance matrix and construct 6 x 6 stiffness matrix
      call invert(q,5,5)
      if(isw.ne.3) go to 60
      call sparse(s,q,xl)
      return
c.... compute out of balance forces
60    v(12) = (5.*v(1)-v(4))/6. + v(12)
      v(13) = (-v(1)+5.*v(4))/6. + v(13)
```

Oct 13 16:42 1980 vplas.f Page 3

```
      v(14) = (v(2)-v(5))/x1 + v(3) + v(14)
      v(15) = (v(2)-v(5))/x1 + v(6) + v(15)
c     do 65 i = 1,6
c5    v(i) = u(i)
      do 70 i = 1,4
      p(i) = -v(i+6)
      do 70 j = 1,5
70    p(i) = p(i) + q(i,j)*v(j+11)
      t = p(1)
      p(1) = (5.*p(1)-p(2))/6.
      p(6) = p(4)
      p(4) = (5.*p(2)-t)/6.
      p(2) = (p(6)+p(3))/x1
      p(5) = -p(2)
      return
      end
```

Oct 13 16:42 1980 sparse.f Page 1

```
      subroutine sparse(s,q,x1)
      implicit double precision (a-h,o-z)
c.... compute effective stiffness matrix for mixed model frame element
      dimension s(6,6),q(5,5)
      s(1,1) = -(25.*q(1,1) - 10.*q(1,2) + q(2,2))/36.
      s(1,3) = -(5.* q(1,3) - q(2,3))/6.
      s(1,4) =  (5.*(q(1,1) + q(2,2)) - 26.*q(1,2))/36.
      s(1,6) = -(5.* q(1,4) - q(2,4))/6.
      s(1,2) = (s(1,3) + s(1,6))/x1
      s(1,5) = -s(1,2)
      s(2,3) = -(q(3,4) + q(3,3))/x1
      s(2,6) = -(q(3,4) + q(4,4))/x1
      s(2,2) = (s(2,3) + s(2,6))/x1
      s(2,5) = -s(2,2)
      s(3,3) = -q(3,3)
      s(3,4) = -(5.*q(2,3) -q(1,3))/6.
      s(3,5) = -s(2,3)
      s(3,6) = -q(3,4)
      s(4,4) = -(25.*q(2,2) - 10.*q(1,2) + q(1,1))/36.
      s(4,6) = -(5.*q(2,4) - q(1,4))/6.
      s(2,4) = (s(4,6) + s(3,4))/x1
      s(4,5) = -s(2,4)
      s(5,5) =  s(2,2)
      s(5,6) = -s(2,6)
      s(6,6) = -q(4,4)
      do 10 i = 1,5
      i1 = i+1
      do 10 j = i1,6
10    s(j,i) = s(i,j)
      return
      end
```

Oct 13 16:42 1980 force.f Page 1

```
      subroutine force(u,v,q,xl)
      implicit double precision (a-h,o-z)
      dimension u(1),v(1),q(5,5)
c.... compute local increment to displacements and add to previous increment
      do 10 i = 1,6
      t = u(i)
      u(i) = t - v(i)
10    v(i) = t
c.... compute increment to forces
      t1 = u(1)
      u(1) = (5.*t1 - u(4))/6.
      t2 = u(2)
      u(2) = (5.*u(4) - t1)/6.
      t1 = (t2 - u(5))/xl
      u(3) = t1 + u(3)
      u(4) = t1 + u(6)
      u(5) = 0,0
      do 20 i = 1,5
      do 20 j = 1,5
20    v(i+6) = v(i+6) - q(i,j)*(u(j)+v(j+11))
      return
      end
```

Oct 13 16:42 1980 invert.f Page 1

```
      subroutine invert(a,nmax,ndm)
      implicit double precision (a-h,o-z)
      dimension a(ndm,ndm)
      do 200 n = 1,nmax
      d = a(n,n)
      do 100 j = 1,nmax
100   a(n,j) = -a(n,j)/d
      do 150 i = 1,nmax
      if(n.eq.i) go to 150
      do 140 j = 1,nmax
      if(n.ne.j) a(i,j) = a(i,j) +a(i,n)*a(n,j)
140   continue
150   a(i,n) = a(i,n)/d
      a(n,n) = 1.0/d
200   continue
      return
      end
```

Oct 13 16:42 1980 global.f Page 1

```
      subroutine global(s,p,cs,sn,nst,ndf,isw)
c.... rotate stiffnss and load to global or local coordinates
      implicit double precision (a-h,o-z)
      dimension s(nst,nst),p(nst)
      if(cs.gt.0.999999999d0) return
      go to (1,1,1,2,1,2,1,2),isw
1     do 13 i = 1,nst,ndf
      j = i + 1
      do 11 n = 1,nst
      t      = s(n,i)*cs - s(n,j)*sn
      s(n,j) = s(n,i)*sn + s(n,j)*cs
11    s(n,i) = t
13    continue
      do 14 i = 1,nst,ndf
      j = i + 1
      do 12 n = 1,nst
      t      = s(i,n)*cs - s(j,n)*sn
      s(j,n) = s(i,n)*sn + s(j,n)*cs
12    s(i,n) = t
14    continue
      return
c.... rotate displacement
2     if(isw.eq.6) sn = -sn
      t    =  p(1)
      p(1) =  cs*t + sn*p(2)
      p(2) = -sn*t + cs*p(2)
      t    =  p(4)
      p(4) =  cs*t + sn*p(5)
      p(5) = -sn*t + cs*p(5)
      return
      end
```