PROCESSOR AND POSTPROCESSOR FOR A PLANE FRAME ANALYSIS PROGRAM ON THE IBM PC

by

Fawwaz I. Ghabra

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Civil Engineering

APPROVED:

S. M. Holzer, Chairman

R. M. Barker

A. E. Somers, Jr.

February, 1985
Blacksburg, Virginia

ABSTRACT

A PROCESSOR AND A POSTPROCESSOR FOR A PLANE FRAME ANALYSIS PROGRAM ON THE IBM PC

by

Fawwaz I. Ghabra

Committee Chairman: S. M. Holzer

Civil Engineering

(ABSTRACT)

In this thesis, a PROCESSOR and a POSTPROCESSOR are developed for a plane frame analysis computer program on the IBM PC. The PROCESSOR reads the data prepared by a PREPROCESSOR and solves for the unknown joint displacements using the matrix displacement method. The POSTPROCESSOR uses the results of the PROCESSOR to obtain the required responses of the structure. A chapter on testing procedures is also provided.

ACKNOWLEDGEMENTS

The author would like to express his gratitude to Dr. S. M. Holzer for serving as the author's major advisor and for providing guidance throughout this thesis.

Appreciation is given to Dr. R. M. Barker and Dr. A. E. Somers,Jr. for their helpful comments and for serving on the author's committee.

The Virginia Electric Power Company, VEPCO, is thanked for sponsoring the work done in this thesis.

The author is also grateful to his parents for their continued love and financial support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter I

PURPOSE

The main purpose of this thesis is the development and testing of a PROCESSOR and a POSTPROCESSOR of a computer program which analyzes plane frames on the IBM PC. This program starts with the computer program developed by Holzer (Ref. 5), adapts it to the IBM PC, and modifies it to meet the specifications described in chapter 2. The resulting program, called the PROCESSOR, uses the data prepared by the PREPROCESSOR (Ref. 3) to solve for the unknown joint displacements. The output of the PROCESSOR includes the fixed-end forces and the joint displacements. The POSTPROCESSOR reads that output and obtains the responses of the structure. These responses include element-end forces, joint forces, internal element forces and deflections at specified sections, maximum joint deflections, and maximum stresses.

Another purpose of this thesis is the brief description of some of the systematic procedures followed in testing computer programs. Concentration is directed on procedures that are practical and can be easily implemented by beginning programmers. Chapter 5 handles the details of suggested testing techniques and tools.

Chapter II

PROGRAM DESCRIPTION

2.1 SPECIFICATIONS

The program analyzes linear plane frames subjected to static loads. The plane frames analyzed can have several features. The structural features include: linear prismatic elements, internal member releases to eliminate bending moments, and linear translational and rotational springs at supports. The loading features cover: joint loads, element loads, multiple load cases, load combinations, and prescribed joint displacements. The design features (which are the task of the PREPROCESSOR (Ref. 3)) require: user friendly data input for ease of problem definition, minimum data input requirements, check of input data for validity, and error messages. The user output options are: echo input data (a task of PREPROCESSOR (Ref. 3)); for each load case and for load combinations print member forces, joint displacements, joint reactions, combinations of member forces stresses and deflections, and maximum stress conditions.

2.2 LAYOUT AND LIMITATIONS

The general layout of the program structure is shown in figure 2.1. The sign convention used in this program is the same as that used by Holzer (Ref. 5). The member load types allowed are depicted in figure 2.2 . The member types allowed are shown in figure 3.1 . It should be noted that structures with zero degree of freedom (i.e. all elements have completely fixed ends) are not permitted. Moreover, the user should be aware that a prescribed displacement or a spring at a support must correspond to a degree of freedom. The number of loading conditions permitted is three plus the dead weight of the structure as a fourth loading condition. This limitation is imposed by the PREPROCESSOR (Ref. 3) and the POSTPROCESSOR because of memory limitations by the IBM PC. The number of load combinations is not limited.

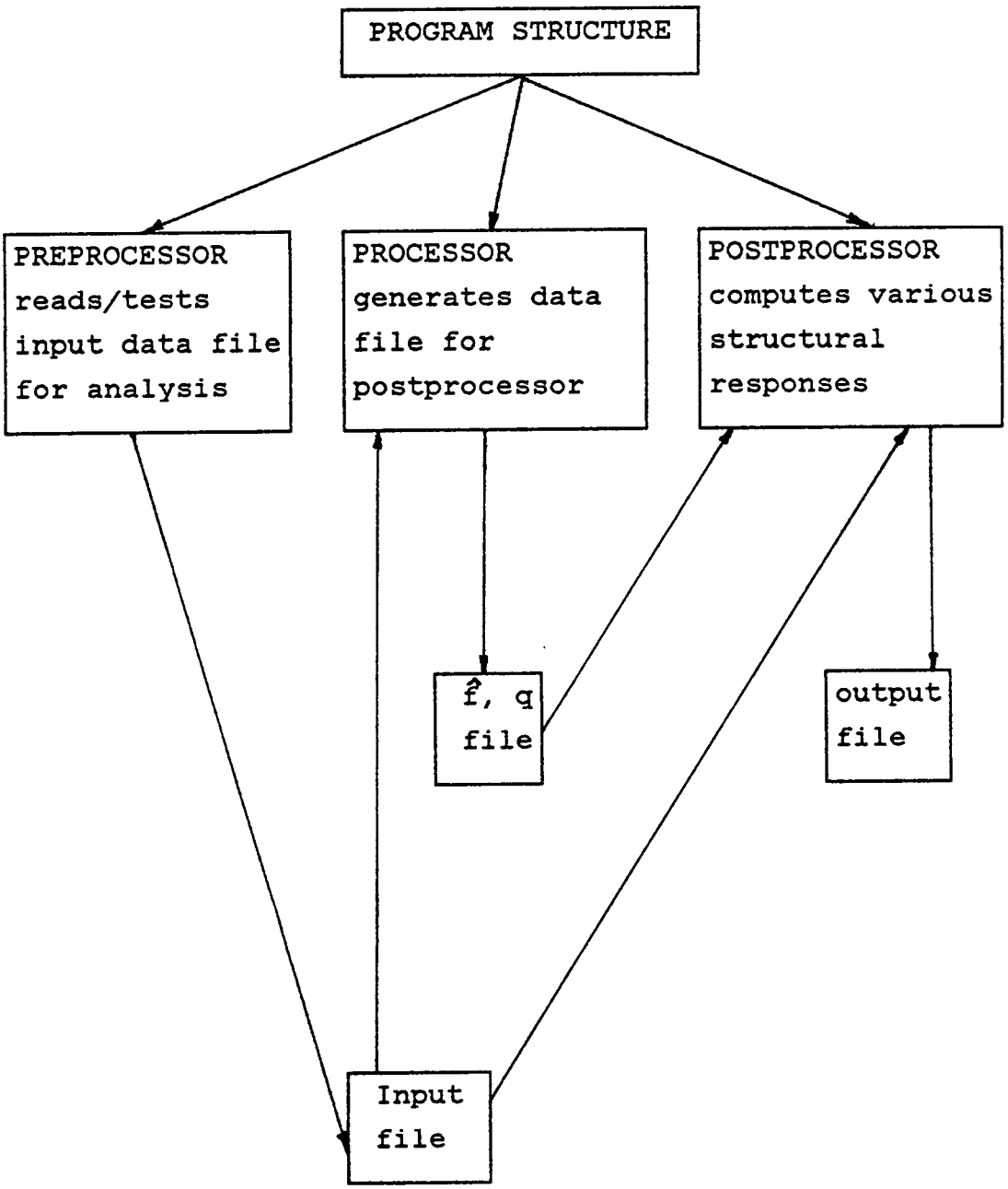
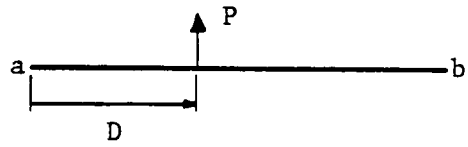


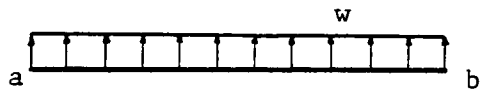
Figure 2.1 Program Layout

(a)



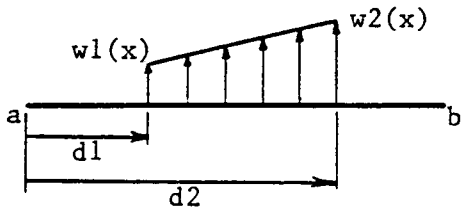
LT=1
P1=P
D1=D
P2=0
D2=0

(b)



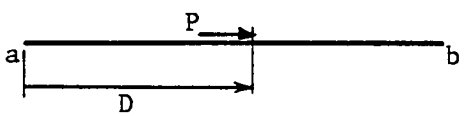
LT=2
P1=w
D1=P2=D2=0

(c)



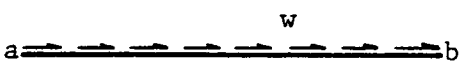
LT=3
P1=w1(x)
D1=d1
P2=w2(x)
D2=d2

(d)



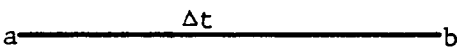
LT=4
P1=P
D1=D
P2=D2=0

(e)



LT=5
P1=w
D1=P2=D2=0

(f)



LT=6
P1= Δt
D1=P2=D2

Figure 2.2 Load Types Allowed

Chapter III

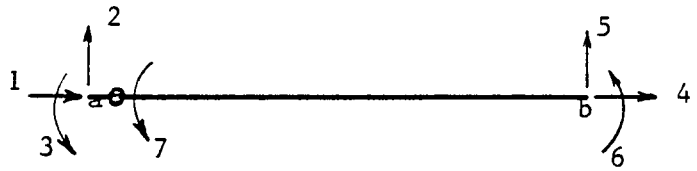
FORMULATIONS

The new features of the PROCESSOR are elements with moment releases (hinges) near their ends, extensional and rotational springs at supports, and prescribed joint displacements. The POSTPROCESSOR determines various structural responses, such as joint forces and internal element forces, due to each loading condition. In addition, the POSTPROCESSOR allows combinations of loading conditions and finds maximum and design stresses and joint deflections. A design value is defined as the maximum of the maxima. The following sections deal with the formulations needed in the program.

3.1 ELEMENT MODELS FOR END-MOMENT RELEASES

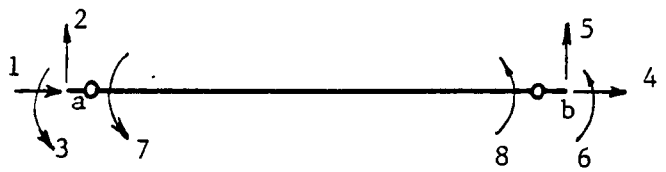
The various types of elements allowed in this program are shown in figure 3.1 . The element in figure 3.1(b) or 3.1(c) can be thought of as a continuous segment attached to a hinge as shown in figures 3.2(b) and 3.2(c). The element in figure 3.1(d) consists of two hinges and a continuous segment as shown in figure 3.2(d). This allows all elements shown in figure 3.1 to be analyzed by the procedure developed for the continuous element in figure 3.1(a).

(a) Type 1: prismatic continuous element



(b) Type 2: prismatic with hinge at a-end

(c) Type 3: prismatic with hinge at b-end



(d) Type 4: prismatic with a hinge at each end

Figure 3.1 Elements Allowed in PROCESSOR

(a) Type 1: continuous segment=ab

(b) Type 2: continuous segment = cb
hinge is ac = infinitesmal

(c) Type 3: continuous segment = ac
hinge is cb = infinitesmal

(d) Type 4: continuous segment = cd
hinges are ac and db

Figure 3.2 Element Types and their components

This procedure requires the displacements and fixed-end forces near the ends of the continuous segments indicated in figure 3.2. Accordingly, the elements in figures 3.1(b)-(d) will be represented by complex elements with the internal degrees of freedom condensed out and at the end of the system analysis procedure recovered (Ref. 5).

In formulating the local and global models of the above mentioned elements, attention is focused on the bending model. This is because the hinge has no effect on the axial deformation model of the element. Hence, once the bending model has been formulated, the complete frame model is immediately deduced.

The derivation of models for the complex element shown in figure 3.3 is discussed by Holzer (Ref. 5). The complex element model is expressed as

$$\begin{bmatrix} \bar{f}_e \\ f_i \end{bmatrix} = \begin{bmatrix} k_{ee} & k_{ei} \\ k_{ie} & k_{ii} \end{bmatrix} \begin{bmatrix} d_e \\ d_i \end{bmatrix} \tag{3.1}$$

where $$\bar{f}_e = \begin{bmatrix} \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \\ \bar{f}_4 \end{bmatrix} \tag{3.2}$$

(b) Displacements

(a) Forces

Figure 3.3 Complex Element with Rotational Springs at both ends

$$f_i = \begin{bmatrix} f_5 \\ f_6 \end{bmatrix} \tag{3.3}$$

$$d_e = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \tag{3.4}$$

and $$d_i = \begin{bmatrix} d_5 \\ d_6 \end{bmatrix} \tag{3.5}$$

all shown in figure 3.3.

Let $\alpha = EI/L^3$ (3.6)

In eq.(3.1),

$$k_{ie} = \alpha \begin{bmatrix} 6L & -4Ls_a^2 & -6L & 0 \\ & 6L & 0 & -6L & -4L^2 s_b \end{bmatrix} \tag{3.7}$$

$$k_{ii} = \alpha \begin{bmatrix} 4L^2(1+s_a) & 2L^2 \\ 2L^2 & 4L^2(1+s_b) \end{bmatrix} \tag{3.8}$$

$$k_{ii}^{-1} = 1/\alpha 4L^2 s \begin{bmatrix} 2(1+s_b) & -1 \\ -1 & 2(1+s_a) \end{bmatrix} \tag{3.9}$$

In these expressions,

$$s = 2(1+s_a)(1+s_b) - 1/2 \tag{3.10}$$

where s_a and s_b are nondimensional rotational spring stiffnesses near the a- and b-end, respectively. The rotational spring constant are $4\alpha L^2 s_a$ and $4\alpha L^2 s_b$.

Equation (3.1) gives:

$$d_i = A.d_e + k_{ii}^{-1}.f_i \qquad (3.11)$$

where $A = -k_{ii}^{-1}.k_{ie}.d_e$ (3.12)

$$A = 1/s \begin{bmatrix} -3/2L(1+2s_b) & 2s_a(1+s_b) & 3/2L(1+2s_b) & -s_b \\ -3/2L(1+2s_a) & -s_a & 3/2L(1+2s_a) & 2s_b(1+s_a) \end{bmatrix}$$

The condensed model is

$$\bar{f}_e = f_e + f_e^* \qquad (3.13a)$$

$$f_e = k_e.d_e \qquad (3.13b)$$

$$k_e = \alpha \begin{bmatrix} 12s_1 & 6Ls_2 & -12s_1 & 6Ls_4 \\ & 4L^2s_3 & -6Ls_2 & 2L^2s_5 \\ & & 12s_1 & -6Ls_4 \\ \text{symm.} & & & 4L^2s_6 \end{bmatrix} \qquad (3.14)$$

where $s_1 = (1/2s)(s_a + s_b + 4s_as_b)$ (3.15a)

$s_2 = (s_a/s)(1+2s_b)$ (3.15b)

$s_3 = (s_a/2s)(3+4s_b)$ (3.15c)

$s_4 = (s_b/s)(1+2s_a)$ (3.15d)

$s_5 = (2/s)(s_as_b)$ (3.15e)

$s_6 = (s_b/2s)(3+4s_a)$ (3.15f)

and $$f_e^* = \begin{bmatrix} f_1^* \\ f_2^* \\ f_3^* \\ f_4^* \end{bmatrix} = \begin{bmatrix} (3/2Ls)((1+2s_b)f_5 + (1+2s_a)f_6) \\ (s_a/s)(-2(1+s_b)f_5 + f_6) \\ (-3/2Ls)((1+2s_b)f_5 + (1+2s_a)f_6) \\ (s_b/s)(f_5 - 2(1+s_a)f_6) \end{bmatrix} \qquad (3.16)$$

As mentioned by Holzer (Ref. 5), f_e^* is the external force vector induced by load f_i alone with $d_e = 0$ (Figure 3.4a). From eq.(3.16), notice that once f_i is known, f_e^* is known. $\hat{f}$ is the external force vector induced by the actual load on the structure with $d_e = 0$; i.e., it is the fixed-end force vector of the condensed model (Figure 3.4b). Also define $\tilde{f}$ to be the fixed-end force vector of the continuous segment under transverse loads(d_e=0 and d_i=0; see figure 3.4c).

3.1.1 Element with Hinge near a-End

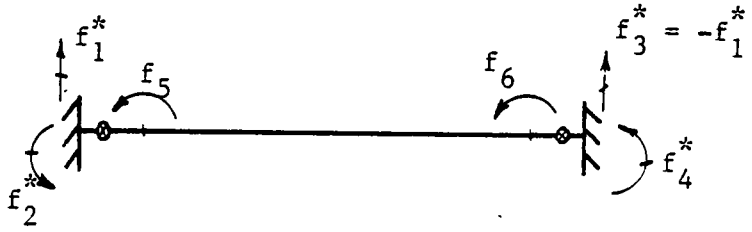
3.1.1.1 Fixed-End Forces $\hat{f}$

One approach to find $\hat{f}$ is to use the condensed element model of figure 3.3. This is done by letting $s_a \to 0$ and $s_b \to \infty$ (Ref. 5). In eq.(3.16), $s_a \to 0$ and $s_b \to \infty$ gives

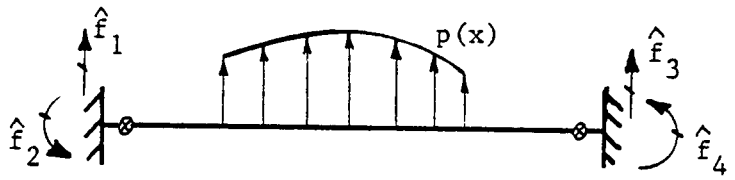
$$f_e^* = \begin{bmatrix} 3f_5/2L \\ 0 \\ -3f_5/2L \\ f_5/2 - f_6 \end{bmatrix} \qquad (3.17)$$

but what are f_5 and f_6, and how can f_e^* be used to find $\hat{f}$? The answer to this question is found in figure 3.5. From figure 3.5:

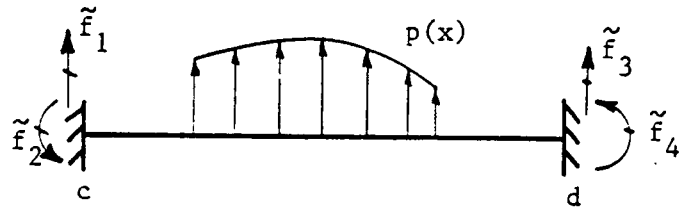
$$\hat{f}_1 = \tilde{f}_1 + f_1^* \qquad \text{and} \qquad f_5 = -\tilde{f}_2$$
$$\hat{f}_2 = 0 + 0 \qquad\qquad f_6 = -\tilde{f}_4$$
$$\hat{f}_3 = \tilde{f}_3 + f_3^*$$
$$\hat{f}_4 = 0 + f_4^*$$



(a) f_e^* : f_e^* is induced by d_e=0 and f_i (f_5 & f_6) acting alone



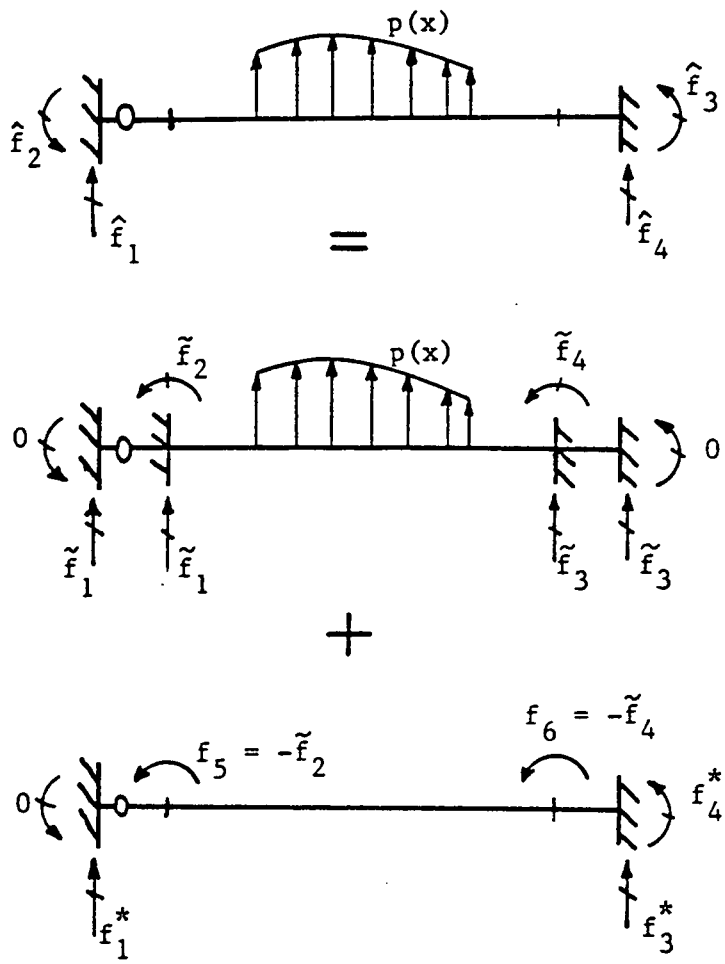
(b) $\hat{f}$: $\hat{f}$ is induced under the actual element load p(x). If the element load consists of only f_5 and f_6 , then $\hat{f} = f^*$ (p(x) =0)



(c) Fixed end forces on continuous segment cd

NOTE: f_5 and f_6 are hypothetical here; no element has such moments acting on it; only transverse forces are allowed (see subroutine MLOAD and figure 2.2)

Figure 3.4 External Force Vectors for Complex Element



$\bar{Q} = 0$
$q = (d_5 \quad d_6)$

$\hat{Q} = (\tilde{f}_2 \quad \tilde{f}_4)$
$\hat{q} = 0$

$Q = \bar{Q} - \hat{Q}$
$= (-\tilde{f}_2 \quad -\tilde{f}_4)$
$q = \tilde{q}$ since $\hat{q}=0$

Figure 3.5 Fixed End Forces for Element Type 2

and using eq.(3.17) leads to

$$\hat{f} = \begin{bmatrix} \tilde{f}_1 - 3\tilde{f}_2/2L \\ 0 \\ \tilde{f}_3 + 3\tilde{f}_2/2L \\ -\tilde{f}_2/2 + \tilde{f}_4 \end{bmatrix} \tag{3.18}$$

3.1.1.2 Element Stiffness Matrix

For this type of element $s_a \rightarrow 0$ and $s_b \rightarrow \infty$. Substituting in eqs.(3.14) and (3.15) gives (Ref. 5):

$$k_e = \alpha \begin{bmatrix} 3 & 0 & -3 & 3L \\ 0 & 0 & 0 & 0 \\ -3 & 0 & 3 & -3L \\ 3L & 0 & -3L & 3L^2 \end{bmatrix}$$

or, including the axial deformation

$$k_e = \alpha \begin{bmatrix} \beta & 0 & 0 & -\beta & 0 & 0 \\ 0 & 3 & 0 & 0 & -3 & 3L \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\beta & 0 & 0 & \beta & 0 & 0 \\ 0 & -3 & 0 & 0 & 3 & -3L \\ 0 & 3L & 0 & 0 & -3L & 3L^2 \end{bmatrix}$$

The global element stiffness matrix is

$$K_e = \begin{bmatrix} g_1 & g_2 & 0 & -g_1 & -g_2 & g_4 \\ & g_2 & 0 & -g_2 & -g_3 & g_5 \\ & & 0 & 0 & 0 & 0 \\ & & & g_1 & g_2 & -g_4 \\ & & & & g_3 & -g_5 \\ \text{sym.} & & & & & g_6 \end{bmatrix} \qquad (3.19a)$$

where $g_1 = \alpha(\beta c_1^2 + 3c_2^2)$ (3.19b)

$g_2 = \alpha c_1 c_2(\beta-3)$

$g_3 = \alpha(\beta c_2^2 + 3c_1^2)$

$g_4 = -\alpha 3Lc_2$

$g_5 = \alpha 3Lc_1$

$g_6 = \alpha 3L^2$

Note: If you doubt that k_e is the condensed element stiffness matrix, i. e. $\bar{f}_e = k_e.d_e$, then remember that $f_i = (f_5, f_6)$ is always zero for our loads (see chapter 2 for allowed element loads); but $f_e^* = M.f_i$ implies that if $f_i = 0$ then $f_e^* = 0$ giving (from eq.(3.13)) $\bar{f}_e = f_e$.

3.1.1.3 Internal Displacement d_5

As $s_a \to 0$ and $s_b \to \infty$, equation (3.12) yields

$$A = \begin{bmatrix} -3/2L & 0 & 3/2L & -1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.20)$$

and equation (3.9) gives

$$k_{ii}^{-1} = \begin{bmatrix} 1/4\alpha L^2 & 0 \\ 0 & 0 \end{bmatrix} \qquad (3.21)$$

Substituting eqs. (3.20), (3.21), (3.3), (3.5), and (3.4) into eq.(3.11) gives:

$$d_5 = 3(d_3 - d_1)/2L - d_4/2 - \tilde{f}_2/4\alpha L^2 \qquad (3.22)$$

3.1.2 Element with Hinge near b-end

In a procedure similar to that described in section 3.1.1, and by letting $s_a \to \infty$ and $s_b \to 0$ the following results are obtained:

3.1.2.1 Fixed-End Forces $\hat{f}$

The fixed-end force vector is

$$\hat{f} = \begin{bmatrix} \tilde{f}_1 - 3\tilde{f}_4/2L \\ \tilde{f}_2 - \tilde{f}_4/2 \\ \tilde{f}_3 + 3\tilde{f}_4/2L \\ 0 \end{bmatrix} \qquad (3.23)$$

3.1.2.2 Element Stiffness Matrix

$$k_e = \alpha \begin{bmatrix} 3 & 3L & -3 & 0 \\ & 3L^2 & -3L & 0 \\ & & 3 & 0 \\ \text{symm.} & & & 0 \end{bmatrix}$$

With the axial deformation included,

$$k_e = \alpha \begin{bmatrix} \beta & 0 & 0 & -\beta & 0 & 0 \\ 0 & 3 & 3L & 0 & -3 & 0 \\ 0 & 3L & 3L^2 & 0 & -3L & 0 \\ -\beta & 0 & 0 & \beta & 0 & 0 \\ 0 & -3 & -3L & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the global stiffness matrix is

$$K_e = \begin{bmatrix} g_1 & g_2 & g_4 & -g_1 & -g_2 & 0 \\ & g_3 & g_5 & -g_2 & -g_3 & 0 \\ & & g_6 & -g_4 & -g_5 & 0 \\ & & & g_1 & g_2 & 0 \\ & & & & g_3 & 0 \\ & \text{symm.} & & & & 0 \end{bmatrix} \qquad (3.24)$$

where g_i, i=1 to 6, are defined in eqs.(3.19b).

3.1.2.3 Internal Displacement d_5

$$d_5 = 3(d_3 - d_1)/2L - d_2/2 - \tilde{f}_4/4\alpha L^2 \qquad (3.25)$$

3.1.3 Element with Hinge at Each End

In this case $s_a \to 0$ and $s_b \to 0$. The following results can be obtained in a procedure similar to that described in section 3.1.2:

3.1.3.1 Fixed-End Forces $\hat{f}$

$$\hat{f} = \begin{bmatrix} \tilde{f}_1 - (\tilde{f}_2 + \tilde{f}_4)/L \\ 0 \\ \tilde{f}_3 + (\tilde{f}_2 + \tilde{f}_4)/L \\ 0 \end{bmatrix} \qquad (3.26)$$

3.1.3.2 Element Stiffness Matrix

$k_e = 0$; zero matrix.

With the axial deformation included, the global stiffness matrix becomes

$$K_e = \begin{bmatrix} g_e & g_2 & 0 & -g_1 & -g_2 & 0 \\ & g_3 & 0 & -g_2 & -g_3 & 0 \\ & & 0 & 0 & 0 & 0 \\ & & & g_1 & g_2 & 0 \\ & & & & g_3 & 0 \\ & \text{symm.} & & & & 0 \end{bmatrix} \qquad (3.26a)$$

where

$$g_1 = \alpha\beta c_1^2, \; g_2 = \alpha\beta c_1 c_2, \text{ and } g_3 = \alpha\beta c_2^2 \qquad (3.26b)$$

3.1.3.3 Internal Displacement d_5

$$d_5 = (d_3-d_1)/L + (-2\tilde{f}_2 + \tilde{f}_4)/6\alpha L^2 \qquad (3.27)$$

Note: d_5 is needed for computation of transverse deflection (TD); the quantity $(-2\tilde{f}_2 + \tilde{f}_4)$ is stored in F(7,MN).

3.2 SPRINGS AT SUPPORTS

The simplest approach to deal with a spring is to add its stiffness to the system stiffness matrix, SS. Suppose that there is a spring at joint J, in direction N, and with stiffness K. N at joint J should correspond to a degree of freedom, say, i. Then the system stiffness would be modified by simply adding K to SS(i,i); i.e., SS(i,i) = SS(i,i) + K. That is, only diagonal elements of SS are affected by having springs attached to individual degrees of freedom. Other elements of SS are not affected because springs do not link degrees of freedom.

3.3 PRESCRIBED JOINT DISPLACEMENTS

The approach used here is the Penalty Method (Bathe, Ref. 2). Prescribed displacements are imposed at the system level (as opposed to the element level). Basically, a spring of large stiffness is added at the degree of freedom j where the displacement is prescribed. The spring stiffness is chosen to be much larger than the $\max(k_{ii})$ where i=1, ..., NEQ.

Suppose $q_j = D_0$ = prescribed displacement. Notice that j is a degree of freedom. The system model, Kq=Q, is modified as follows:

$K_{jj} = C$

and $Q_j = D_0 . C$

where $C >> \max (K_{ii})$ $\quad i=1, \ldots, NEQ$

C = Penalty Number

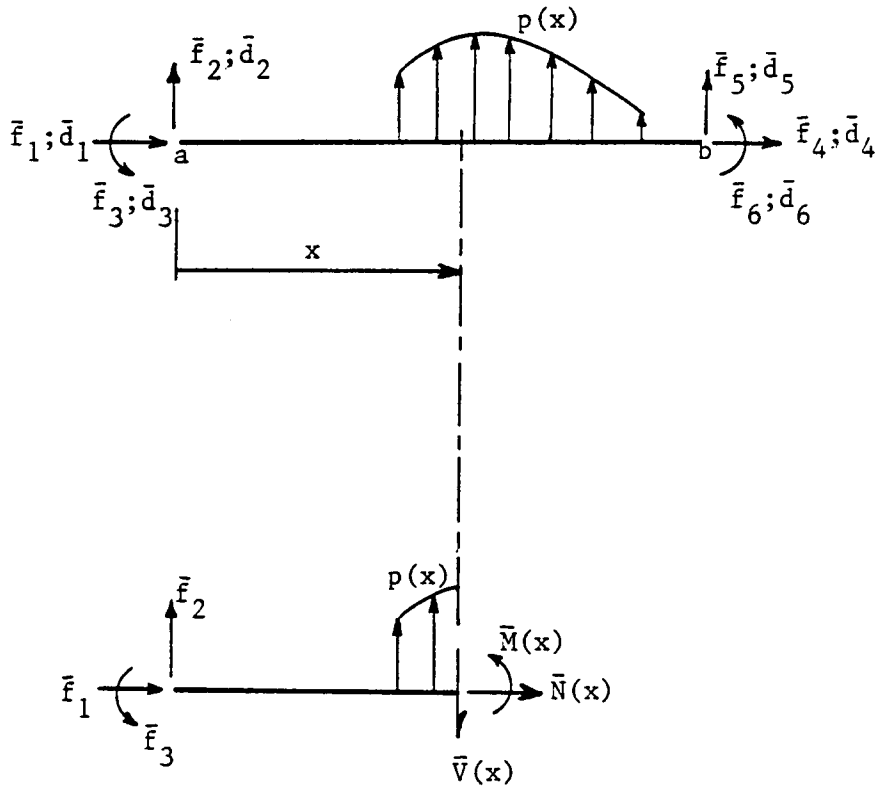
It seems that $C/\max(K_{ii}) = 1000.$ is proven by experience to be a good choice (Ref. 2).

3.4 FORMULATIONS FOR INTERNAL ELEMENT RESPONSES

Internal element responses include axial force, shear force, bending moment, and transverse deflections. Let r(x) denote the total response at distance x from the a-end of an element. Let $r^f(x)$ denote the response due to element-end forces (or displacements), and $r^p(x)$ the response due to member loads (or actions). Then,

$$r(x) = r^f(x) + r^p(x) \tag{3.28}$$

Figure 3.6 explains the terms used in the subsequent derivations.



p(x) = total transverse load on element at distance x from a-end

ab is the continuous segment of element (see figure 3.2)

Figure 3.6 Section forces

3.4.1 Shear

$$d\bar{V}/dx = p(x)$$

$$d\bar{V} = p(x).dx$$

$$\int_0^x d\bar{V} = \int_0^x p(x).dx$$

$$\bar{V}(x) - \bar{V}(0) = \int_0^x p(x).dx$$

but $\bar{V}(0) = \bar{f}_2$

therefore, $\bar{V}(x) = \bar{f}_2 + \int_0^x p(x).dx$

Let $\bar{V}^p(x) = \int_0^x p(x).dx$ (3.29)

then,

$$\bar{V}(x) = \bar{f}_2 + \bar{V}^p(x) \quad (3.30)$$

Notice that $\bar{V}(x)$ corresponds to $r(x)$ in eq.(3.28), $\bar{f}_2$ to $r^f(x)$, and $\bar{V}^p(x)$ to $r^p(x)$.

3.4.2 Moment

$$d\bar{M}/dx = \bar{V}(x)$$

$$\int_0^x d\bar{M} = \int_0^x \bar{V}(x).dx$$

$$\bar{M}(x) - \bar{M}(0) = \int_0^x \bar{V}(x).dx$$

but $\bar{M}(0) = -\bar{f}_3$

Substituting for $\bar{V}(x)$ from eq.(3.30):

$$\bar{M}(x) = -\bar{f}_3 + \bar{f}_2.x + \int_0^x \bar{V}^p(x).dx$$

let $\bar{M}^p(x) = \int_0^x \bar{V}^p(x).dx$ (3.31)

then,

$$\bar{M}(x) = -\bar{f}_3 + \bar{f}_2.x + \bar{M}^p(x) \quad (3.32)$$

3.4.3 Transverse Deflection

$EI.d^2\bar{v}/dx^2 = \bar{M}(x)$, but $\bar{\theta}(x) = d\bar{v}/dx$

$$_0\int^x EI\,d\bar{\theta} = {}_0\int^x \bar{M}(x).dx$$

$$EI\{\bar{\theta}(x) - \bar{\theta}(0)\} = {}_0\int^x \bar{M}(x).dx$$

but $\bar{\theta}(0) = \bar{d}_3$ for MTYP=1 and MTYP=3

$= \bar{d}_7$ for MTYP=2 and MTYP=4

this is because only the continuous segment is of concern.

$$EI\bar{\theta}(x) = EI\bar{d}_3 + {}_0\int^x \bar{M}(x).dx$$

Substituting for $\bar{M}(x)$ from eq.(3.32),

$$EI\bar{\theta}(x) = EI\bar{d}_3 - \bar{f}_3x + \bar{f}_2x^2/2 + {}_0\int^x \bar{M}^p(x).dx$$

Let $\bar{\theta}^p(x) = (1/EI)\ {}_0\int^x \bar{M}^p(x).dx$ (3.33)

then,

$$EI.\bar{\theta}(x) = EI\bar{d}_3 - \bar{f}_3x + \bar{f}_2x^2/2 + EI.\bar{\theta}^p(x) \quad (3.34)$$

But $d\bar{v}(x)/dx = \bar{\theta}(x)$

$$_0\int^x d\bar{v} = {}_0\int^x \bar{\theta}(x).dx$$

$$\bar{v}(x) - \bar{v}(0) = {}_0\int^x \bar{\theta}(x).dx$$

but $\bar{v}(0) = \bar{d}_2$

then,

$$\bar{v}(x) = \bar{d}_2 + {}_0\int^x \bar{\theta}(x).dx$$

using eq.(3.34):

$$\bar{v}(x)=\bar{d}_2+x.\bar{d}_3+(1/EI)(-\bar{f}_3x^2/2+\bar{f}_2x^3/6)+{}_0\int^x \bar{\theta}^p(x)dx$$

let $_0\int^x \bar{\theta}^p(x).dx = \bar{v}^p(x)$ (3.35)

then,

$$\bar{v}(x)= \bar{d}_2 +x\bar{d}_3 +(1/EI)(-\bar{f}_3x^2/2 +\bar{f}_2x^3/6) + \bar{v}^p(x)$$

... (3.36a)

again, $\bar{d}_3$ is $\bar{d}_3$ for MTYP=1 or 3)

) (3.36b)

$\bar{d}_3$ is $\bar{d}_7$ for MTYP=2 or 4)

3.4.4 Specialization to Member Loads Used in this Program

3.4.4.1 Member Loads of Type 1

From (Ref. 5),

$$p(x) = P \langle x-a \rangle^{-1} \tag{3.37}$$

$\langle x-a \rangle^n$ is called the singularity function: $0<x<L$ and $0<a<L$.

For information on this fuction see (Ref. 4, p.17).

Equations (3.29) and (3.37) give

$$\bar{V}^p(x) = {}_0\int^x P \langle x-a \rangle^{-1}.dx$$

$$\bar{V}^p(x) = P \langle x-a \rangle^0 \tag{3.38}$$

Equations (3.31) and (3.38) give

$$\bar{M}^p(x) = {}_0\int^x \bar{V}^p(x).dx = {}_0\int^x P \langle x-a \rangle^0 \, dx$$

$$\bar{M}^p(x) = P \langle x-a \rangle^1 \tag{3.39}$$

Equations (3.33) and (3.39) give

$$\bar{\theta}^p(x) = (1/EI)\,{}_0\int^x P \langle x-a \rangle^1 \, dx$$

$$\bar{\theta}^p(x) = (P/2EI) \langle x-a \rangle^2 \tag{3.40}$$

Equations (3.35) and (3.40) give

$$\bar{v}^p(x) = {}_0\int^x (P/2EI) \langle x-a \rangle^2 \, dx$$

$$\bar{v}^p(x) = (P/6EI) \langle x-a \rangle^3 \tag{3.41}$$

3.4.4.2 Member Loads of type 2

p(x) = w = constant

Equation (3.29) gives

$$\bar{V}^p(x) = {}_0\int^x w.dx$$

$$\bar{V}^p(x) = w.x \qquad (3.42)$$

Equations (3.42) and (3.31) give

$$\bar{M}^p(x) = {}_0\int^x w.x.dx = w.x^2/2$$

$$\bar{M}^p(x) = w.x^2/2 \qquad (3.43)$$

Equations (3.43) and (3.33) give

$$\bar{\theta}^p(x) = (1/EI)\ {}_0\int^x wx^2/2.dx = wx^3/6EI$$

$$\bar{\theta}^p(x) = wx^3/6EI \qquad (3.44)$$

Equations (3.44) and (3.35) give

$$\bar{v}^p(x) = {}_0\int^x (wx^3/6EI)dx = wx^4/24EI$$

$$\bar{v}^p(x) = wx^4/24EI \qquad (3.45)$$

3.4.4.3 Member Loads of Type 3

Again from (Ref. 4 and Ref. 5),

$$p(x) = P_a\langle x-a\rangle^0 + s\langle x-a\rangle^1 - P_b\langle x-b\rangle^0 - s\langle x-b\rangle^1 \qquad (3.46a)$$

where

$$s = (P_b-P_a)/(b-a) \qquad (3.46b)$$

Equations (3.29) and (3.46a) give

$$\bar{V}^p(x) = {}_0\int^x \{P_a\langle x-a\rangle^0 + s\langle x-a\rangle^1 - P_b\langle x-b\rangle^0 - s\langle x-b\rangle^1\}\ dx$$

$$\bar{V}^p(x) = P_a\langle x-a\rangle^1 + (s/2)\langle x-a\rangle^2 - P_b\langle x-b\rangle^1 - (s/2)\langle x-b\rangle^2 \qquad (3.47)$$

Equations (3.31) and (3.47) give

$$\bar{M}^p(x)=(P_a/2)\langle x-a\rangle^2+(s/6)\langle x-a\rangle^3-(P_b/2)\langle x-b\rangle^2-(s/6)\langle x-b\rangle^3 \quad \ldots\ldots \quad (3.48)$$

Equations (3.33) and (3.48) give

$$\bar{\theta}^p(x)=(1/EI)\{(P_a/6)\langle x-a\rangle^3+(s/24)\langle x-a\rangle^4-(P_b/6)\langle x-b\rangle^3 -(s/24)\langle x-b\rangle^4\} \quad (3.49)$$

Equations (3.35) and (3.49) give

$$\bar{v}^p(x)=(1/EI)\{(P_a/24)\langle x-a\rangle^4+(s/120)\langle x-a\rangle^5 -(P_b/24)\langle x-b\rangle^4-(s/120)\langle x-b\rangle^5\} \quad (3.50)$$

3.4.4.4 Other Types of Member Loads

Other member loads contribute only to the axial force. Their equations are easily formulated (see chapter 4), and will not be discussed here.

For temperature action the axial force, $\hat{N}(x)$, is constant all through the element. $N(x)$ is always constant. But $\bar{N}(x) = N + \hat{N}$; this implies that $N(x)$ is also constant.

Chapter IV

DOCUMENTATION

4.1 PROCESSOR

FUNCTION: The PROCESSOR produces a data file to be used by the POSTPROCESSOR. This data file contains the member code matrix, the joint displacement vector for each loading condition, and the fixed end force matrix for each loading condition. The PROCESSOR uses the matrix displacement method to find, for each loading condition, the unknown joint displacements of the linear plane frame under analysis. The plane frame can be composed of prismatic elements of types shown in figure (3.1). Static loads applied to the frame may be joint loads or member loads (actions) shown in figure (2.2). Translational and rotational linear springs at the joints of the structure are allowed. In addition, prescribed joint displacements can be specified. The number of multiple loading conditions that can be handled is not limited. However, the PREPROCESSOR (Ref. 3) and POSTPROCESSOR cannot deal with more than three loading conditions plus the dead weight of the structure as a fourth loading condition.

DESIGN: The structure of the PROCESSOR is shown in figure (4.1). The subroutines are described in section (4.1.3).

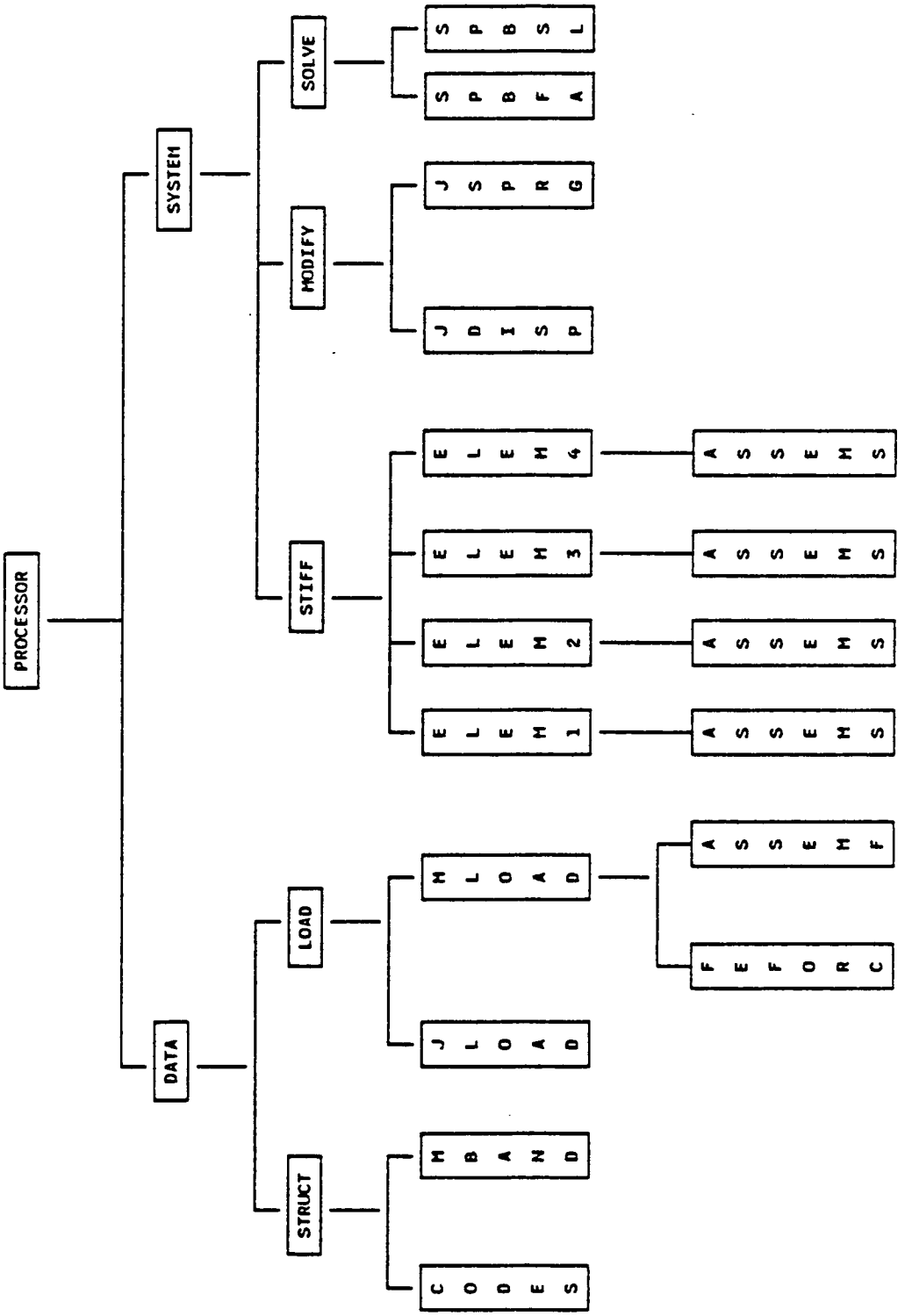


Figure 4.1 Tree Chart of PROCESSOR

4.1.1 List of Variables For PROCESSOR

Parameters:

MX	Maximum number of elements or joints
MXNEQ	Maximum number of equations (degrees of freedom)

Variables:

AREA	Cross-sectional area
C1(I),C2(I)	Direction cosines of element I
COEF	Spring stiffness coefficient
D1,D2	Distances from a-end of element to points of application of loads (figure 2.2)
DISP	Prescribed joint displacement
ELENG(I)	Length of element I
EMOD	Modulus of elasticity
F(7,MX)	Local element fixed-end force matrix: For L=1 to 6: F(L,I)= Lth force of element I (fig. 3.1a) For L=7: F(7,I) stores F3 for element I of type 2 F(7,I) stores quantity (F6-2*F3) for type 4
F2,F3,F5,F6	Fixed-end force vector for a continuous element (type 1) under transverse loads
FORCE	Applied joint force
G(L)	Global element stiffness coefficients

	defined by the following:
	For element type 1: eq.(3.64) of Ref.5;L=1,7
	For types 2 and 3: eq.(3.26b) ; L=1 to 6
	For type 4: eq.(3.26b) ; L=1 to 3
INDEX(6,6)	Index matrix defined by the following:
	For element type 1: eq.(7.4) of Ref. 5
	For element type 2: eq.(3.19a)
	For element type 3: eq.(3.24)
	For element type 4: eq.(3.26a)
J	In CODES, joint at the a-end of element
JCODE(3,MX)	Joint code matrix (see Ref. 5)
JDIR	Joint direction
JNUM	Joint number
K	In CODES, joint at the b-end of element
LC	Load condition
LT	Element load (or action) type:
	LT=1 for load in figure 2.2a
	LT=2 for load in figure 2.2b
	LT=3 for load in figure 2.2c
	LT=4 for load in figure 2.2d
	LT=5 for load in figure 2.2e
	LT=6 for load in figure 2.2f
MBD	Half bandwidth (eq. 6.2 of Ref. 5)
MCODE(6,MX)	Member code matrix
	(see sections 2.3 & 2.4 of Ref. 5)

MN	Member number
MTYP(I)	Member type of element I: MTYP(I)=1 for element in figure 3.1a MTYP(I)=2 for element in figure 3.1b MTYP(I)=3 for element in figure 3.1c MTYP(I)=4 for element in figure 3.1d
NA(I)	Number of loads (actions) on element I.
NE	Number of elements
NEQ	Number of equations (degrees of freedom)
NJ	Number of joints
NJD	Number of prescribed joint displacements
NJL	Number of applied joint loads
NJS	Number of joint springs
NLC	Number of loading conditions
NML	Number of member loads (actions)
P1,P2	Member loads (actions) as shown in figure 2.2
PFAC	Penalty factor= penalty number multiplier; A common value is 1000. (Ref. 2)
PNUM	Penalty number= PFAC*max(Kii)
Q(MXNEQ)	Joint load,displacement vector (see Ref. 5)
SS(MXNEQ, MXNEQ)	System stiffness band matrix stored in the form of figure 6.3c of Ref. 5 in the region SS(MBD+1,NEQ)
ZI	Moment of inertia about the local z-axis

of element

Input/output parameters:

IG	The unit number of file GEOMETRY.FRA
IP	The unit number of file PROP.FRA

4.1.2 Data Files

Data file GEOMETRY.FRA

	Read in:
1. NE, NJ, NLC	PROCESSOR
2. ELENG(I), C1(I), C2(I), MTYP(I); I=1	DATA
: : : : :	
: : : : :	
: : : : :	
NE	
3. JCODE(1,J), JCODE(2,J), JCODE(3,J); J=1	STRUCT
: : : :	
: : : :	
: : : :	
NJ	
4. J, K; I=1	CODES
: : :	
: : : member incidences	
: : :	
NE	
5. NEQ	DATA
11. NJD	MODIFY
12. JNUM, JDIR, DISP; 1	JDISP
: : : : prescribed joint	
: : : : displacements	
NJD	

13. NJS MODIFY

14. JNUM, JDIR, COEF; 1 JSPRG

: : : : joint springs

: : : :

: : : :

NJS

Data files LCi.FRA, where i= 1 to 4

Read in:

6. NJL — LOAD

7. JNUM, JDIR, FORCE; 1 — JLOAD

: : : :

: : : :

: : : :

NJL

8. NML — LOAD

9. MN, LT, P1, D1, P2, D2; 1 — MLOAD

: : : : : : :

: : : : : : :

: : : : : : :

NML

Data file PROP.FRA

Read in:

10. AREA, EMOD, ZI; I=1 ELEM1,2,3,or4

: : : :

: : : :

: : : :

NE

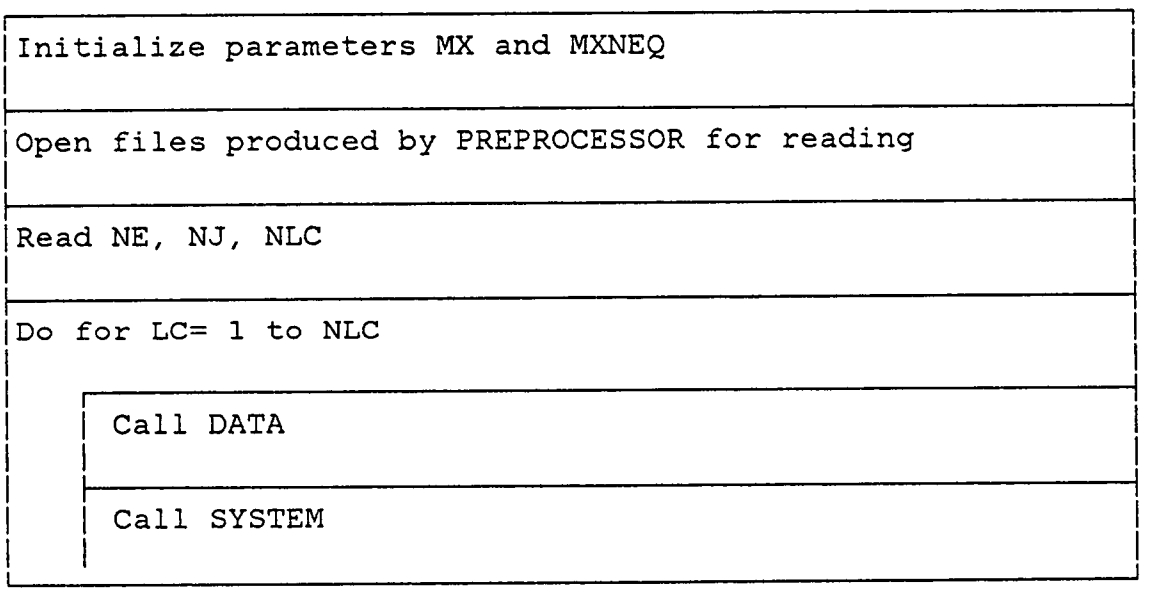
4.1.3 N-S Diagrams

PROCESSOR

The processor consists of the subroutines DATA and SYSTEM (Fig. 4.1). DATA reads data from files produced by the preprocessor and generates the data required in the matrix displacement analysis. SYSTEM constructs the system stiffness matrix and solves the system equations for the joint displacements.

Function: Iinitialize MX and MXNEQ, where MXNEQ=3*(MX-1). Read NE, NJ, NLC; for each loading condition, call DATA and SYSTEM.

NS-diagram:



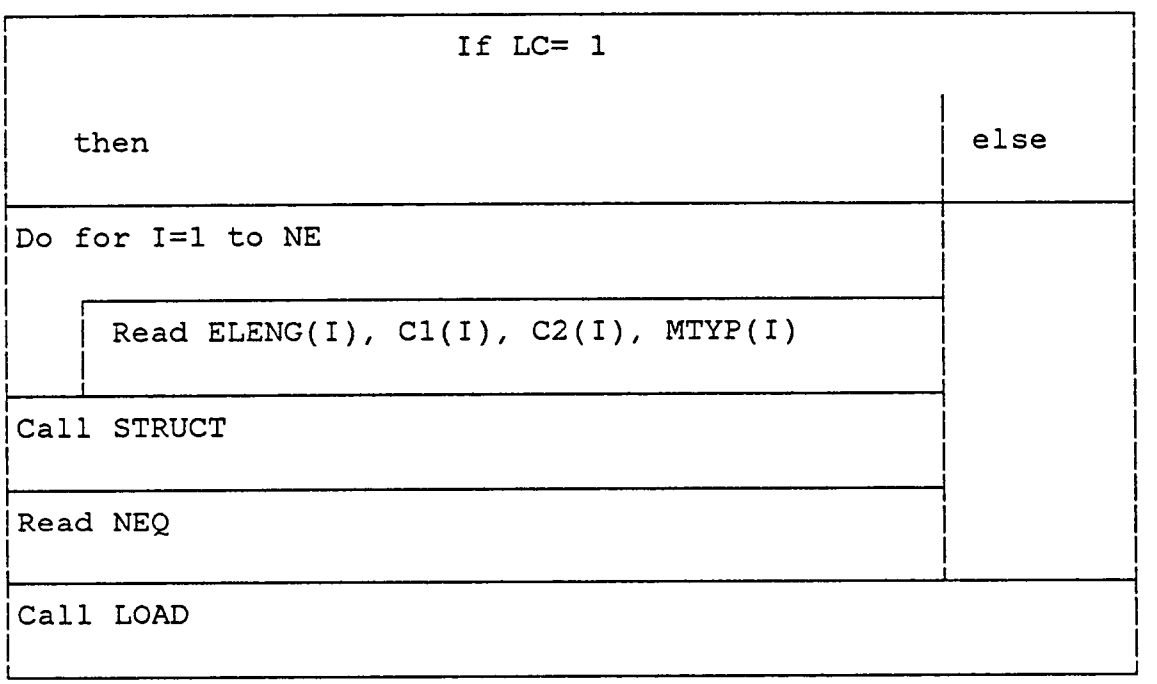
Subroutine DATA

Function: For the first loading condition, LC=1, read ELENG, C1, C2, and MTYP; then call STRUCT and read NEQ. For all loading conditions, call LOAD. STRUCT is called to process structural data. LOAD is called to process load data.

Input arguments: IG, LC, NE, NJ

Output arguments: C1, C2, ELENG, F, JCODE, MBD, MCODE, MTYP, NA, NEQ, Q.

NS-diagram:



Subroutine STRUCT

Function: Read the JCODE matrix; call CODES and MBAND.

Input arguments: IG, NE, NJ.

output arguments: JCODE, MBD, MCODE.

NS-diagram:

Read JCODE(L,J) for J=1 to NJ and for L=1 to 3
Call CODES
MBD= MBAND(MCODE, NE)

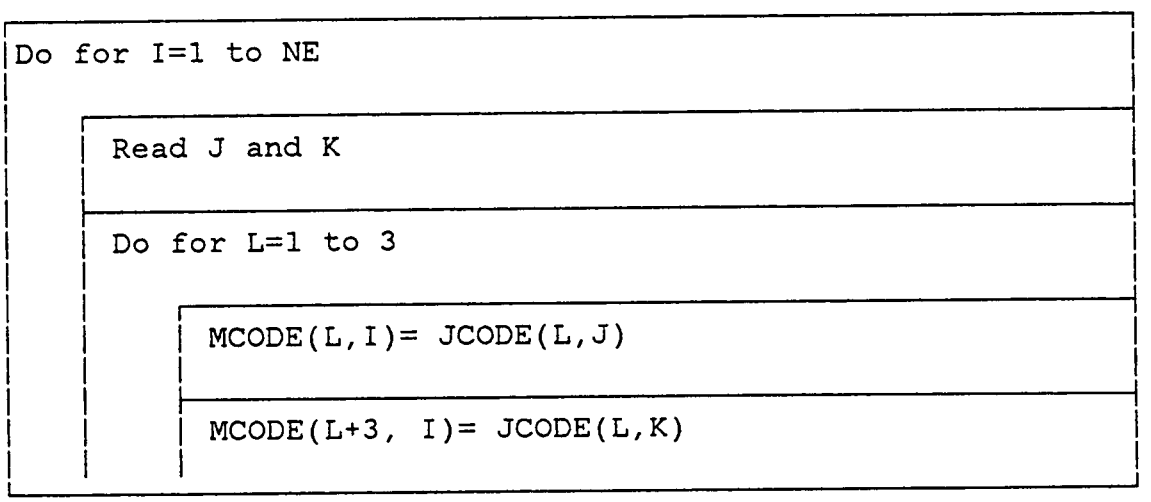
Subroutine CODES

Function: Read the member incidences, J and K, of element I. Use these incidences to generate the member code, MCODE, by transferring columns of JCODE into columns of MCODE.

Input arguments: IG, JCODE, NE.

Output arguments: MCODE.

NS-diagram:



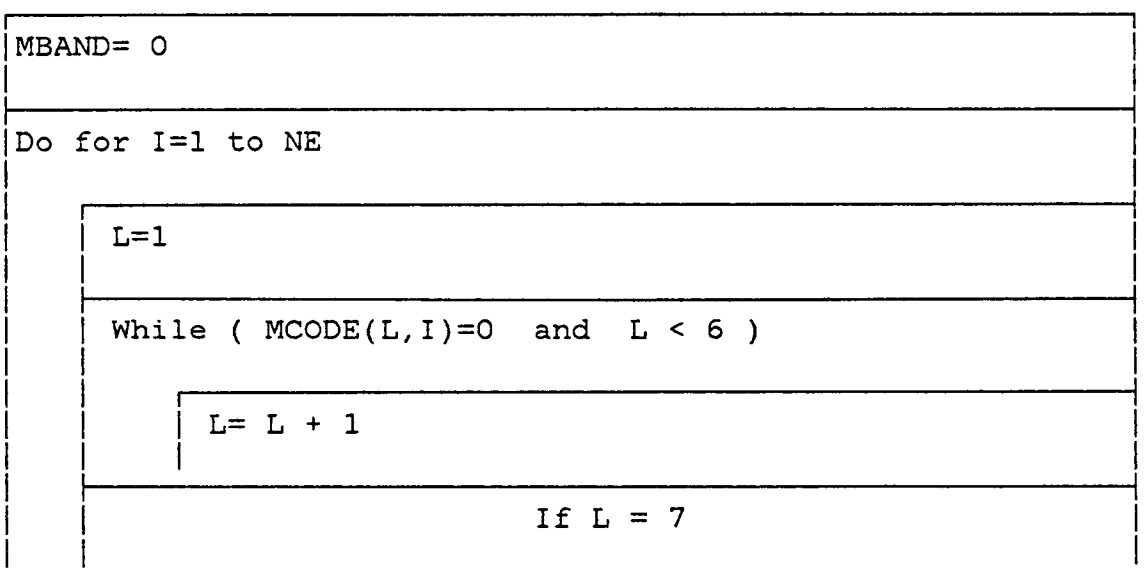
Function MBAND

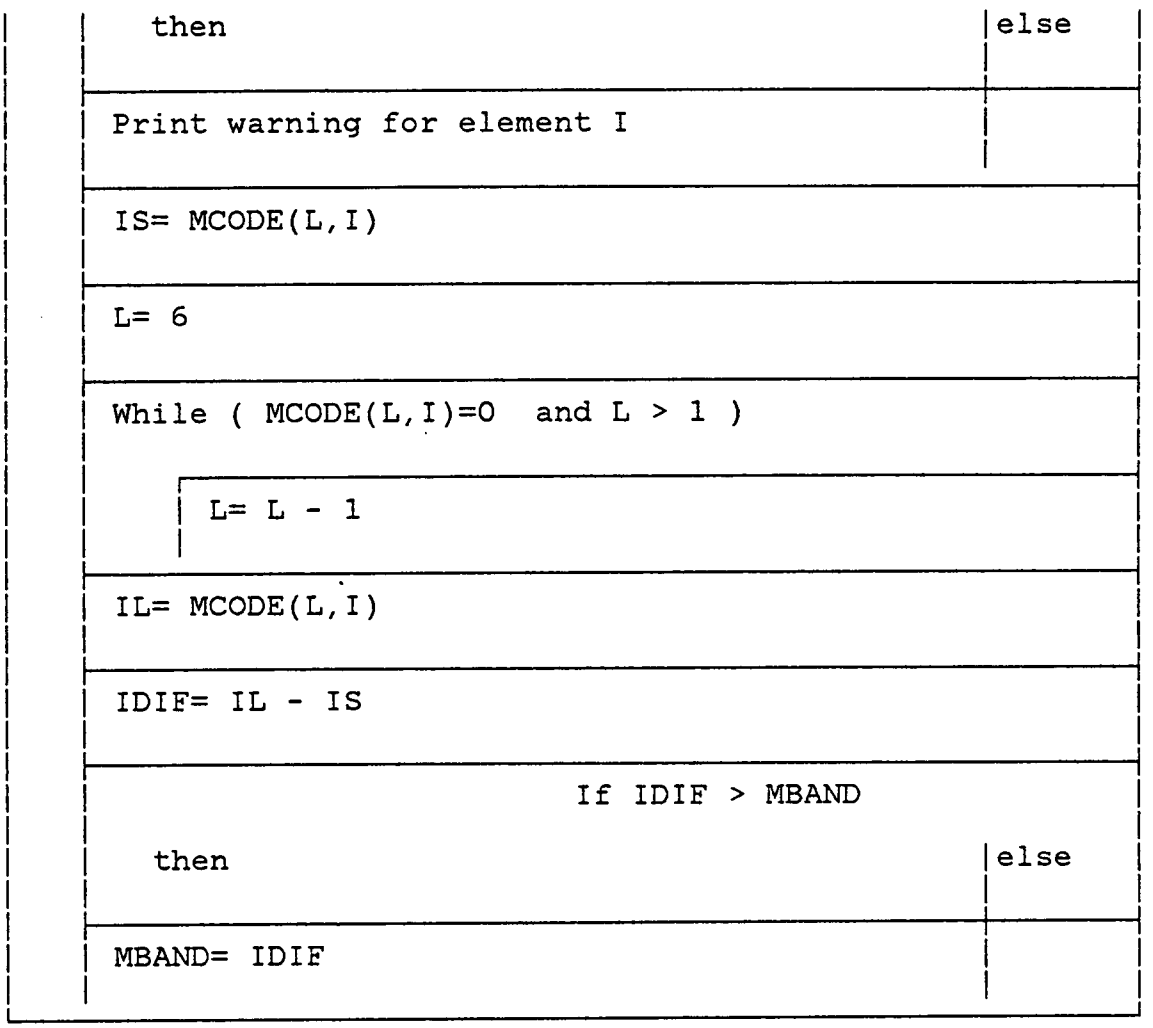
Function: Compute the half-bandwidth, MBAND, by equation 6.2 (Ref. 5): In each column of MCODE, the first and last nonzero integers are the smallest and largest nonzero integers, respectively, of that column. MBAND is the maximum difference of the nonzero integers in any column of MCODE.

Input arguments: MCODE, NE.

Output argument: MBAND.

NS-diagram:





Subroutine LOAD

Function: Read, for the current LC, NJL and NML. Initialize to zero the joint load vector, Q, the local element force vector, F, and the number of member actions vector, NA. If there are joint loads, that is NJL is not zero, call JLOAD. If there are member loads, that is NML is not zero, call MLOAD.

Input arguments: C1, C2, ELENG, IG, JCODE, LC, MCODE, MTYP, NE, NEQ.

Output arguments: F, NA, Q.

NS-diagram:

Read NJL from file LC	
Initialize Q(K), F(L,I), and NA(I) to zero; for K=1 to NEQ, for I= 1 to NE, and for L=1 to 6.	
If NJL is not zero	
then	else
Call JLOAD	
Read NML from file LC	
If NML is not zero	
then	else
Call MLOAD	

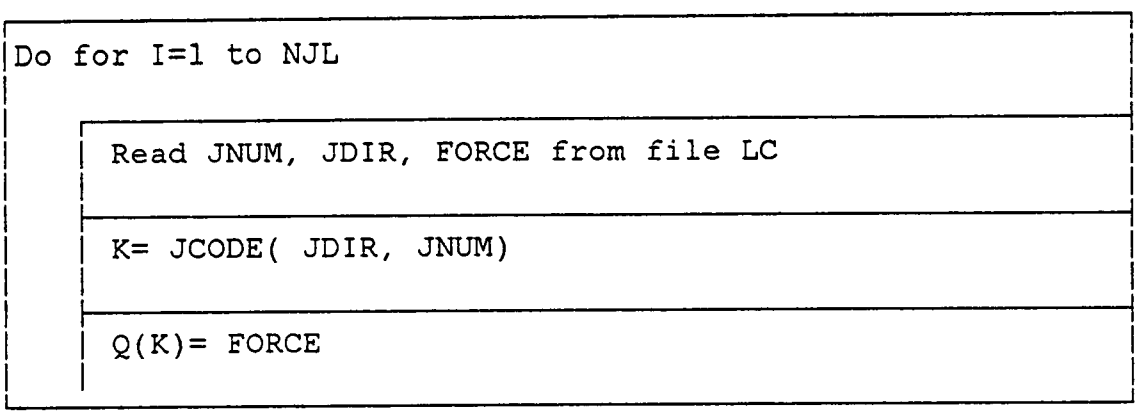
Subroutine JLOAD

Function: For each one of the NJL joint loads, read the joint number, JNUM, the joint direction, JDIR, and the applied force, FORCE; then store FORCE in Q corresponding to JNUM and JDIR of FORCE.

Input arguments: JCODE, LC, NJL, Q.

Output arguments: Q.

NS-diagram:



Subroutine MLOAD

Function: For each one of the NML member loads, read the member number, MN, the member load type, LT, the load param-

eters, P1 and P2, and the distances D1 & D2; If LT is 4, 5, or 6 compute and accumulate the fixed-end forces, F(1,I) and F(4,I). If LT is 1, 2, or 3, then compute F2, F3, F5, and F6 (the fixed-end forces for a continuous member); call FEFORC for actual fixed-end forces. Call ASSEMF.

Input arguments: C1, C2, ELENG, F, LC, MCODE, MTYP, NA, NE, NML, Q.

Output arguments: F, NA, Q.

NS-diagram:

<table>
<tr><td colspan="6">Do for I=1 to NML</td></tr>
<tr><td colspan="6">Read MN, LT, P1, D1, P2, and D2 from file LC</td></tr>
<tr><td colspan="6">NA(MN)= NA(MN) + 1</td></tr>
<tr><td colspan="6">Select case LT</td></tr>
<tr><td>LT= 1</td><td>LT= 2</td><td>LT= 3</td><td>LT= 4</td><td>LT=5</td><td>LT= 6</td></tr>
<tr><td>Compute
$\tilde{f}2$, $\tilde{f}3$,
$\tilde{f}5$, and
$\tilde{f}6$</td><td><--same</td><td><--same</td><td>Compute
$\hat{f}1$ and
$\hat{f}4$</td><td><--same</td><td><--same</td></tr>
</table>

Call FEFORC	<--same	<--same			
Call ASSEMF					

Subroutine FEFORC

Function: For the member MN of type 2, store F3 in F(7,MN). For type 4, store the quantity (F6-2*F3) in F(7,MN). For types 2,3, and 4 compute the fixed-end forces by eqs.(3.18), (3.23), and (3.26). For all types, accumulate the fixed-end forces F(2,MN), F(3,MN), F(5,MN), and F(6,MN).

Input arguments: ELENG, F, F2, F3, F5, F6, MN, MTYP.

Output argument: F.

NS-diagram:

Select case MTYP(MN)			
1	2	3	4
Accumulate $\hat{f}2$, $\hat{f}3$, $\hat{f}5$, and $\hat{f}6$	Store and accumulate $\tilde{f}3$ in F(7,MN)	Compute & accumulate $\hat{\tilde{f}}2$, $\hat{\tilde{f}}3$, $\hat{\tilde{f}}5$,	Store and accumulate ($\tilde{f}6$ - 2*$\tilde{f}3$) in F(7,MN)

	Compute & accumulate $\hat{f}2$, $\hat{f}3$, $\hat{f}5$, & $\hat{f}6$ by eq.(3.18)	& $\hat{f}6$ by eq.(3.23)	Compute & accumulate $\hat{f}2$, $\hat{f}3$, $\hat{f}5$, & $\hat{f}6$ by eq.(3.26)

Subroutine ASSEMF

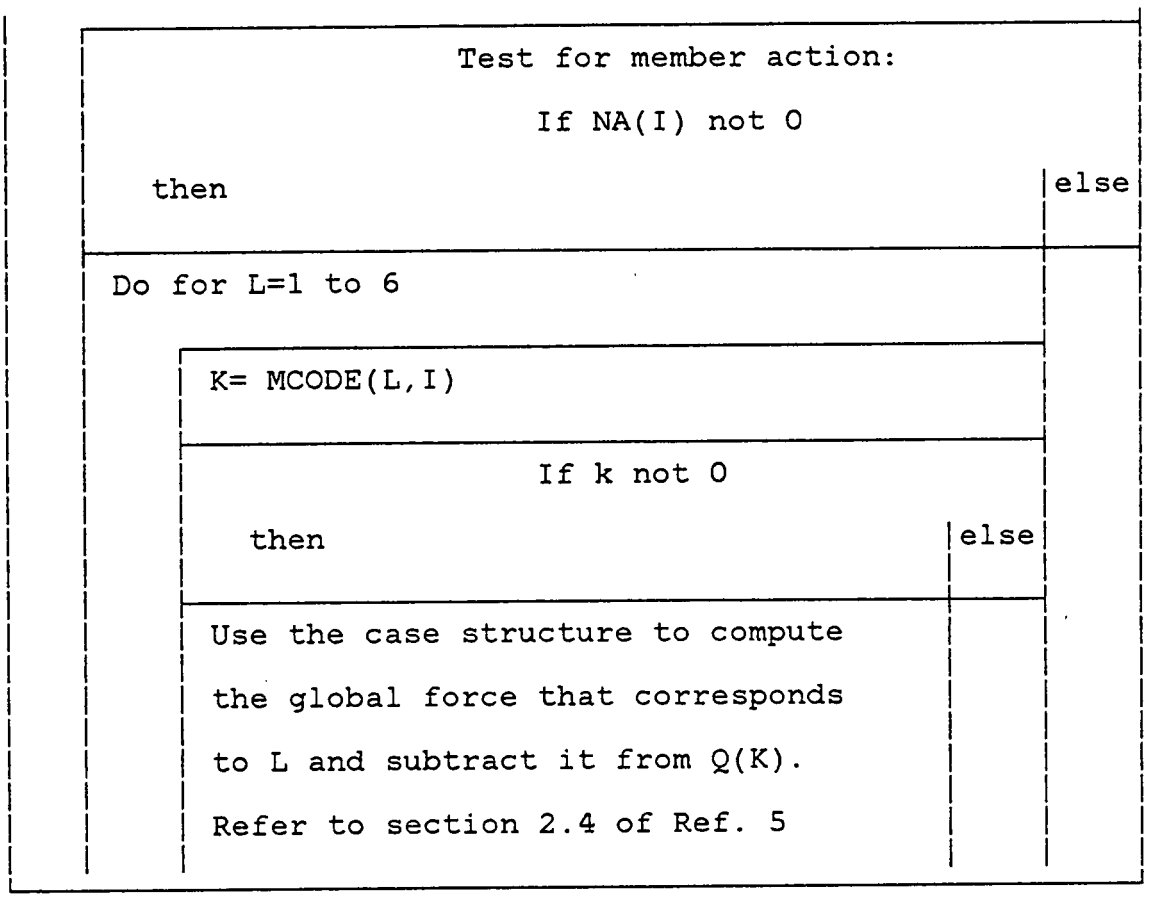
Function: Transform and assemble the local fixed-end forces, F(L,I), to produce the equivalent joint load vector, Q, by equations 2.34, 2.37, 3.92-3.94, and the force transformation of section 2.4 (Ref. 5).

Input arguments: C1, C2, F, MCODE, NA, NE, Q.

Output argument: Q.

NS-diagram:

Statement function for equations 2.34, & 2.37 (REF. 5)
FG(C1I, C2I, FLX, FLY)= C1I*FLX + C2I*FLY
Do for I=1 to NE



Subroutine SYSTEM

Function: For the first load condition, LC=1, call STIFF and MODIFY ; for all load conditions, call SOLVE.

Input arguments: C1, C2, ELENG, IG, IP, JCODE, LC, MBD, MCODE, MTYP, MXNEQ, NE, NEQ, Q.

Output arguments: Q, SS.

Note: The input argument Q stores the joint loads; the output argument Q stores the joint displacements.

NS-disgram:

<table>
<tr><td colspan="2">If LC=1</td></tr>
<tr><td>then</td><td>else</td></tr>
<tr><td>Call STIFF</td><td rowspan="2"></td></tr>
<tr><td>Call MODIFY</td></tr>
<tr><td colspan="2">Call SOLVE</td></tr>
</table>

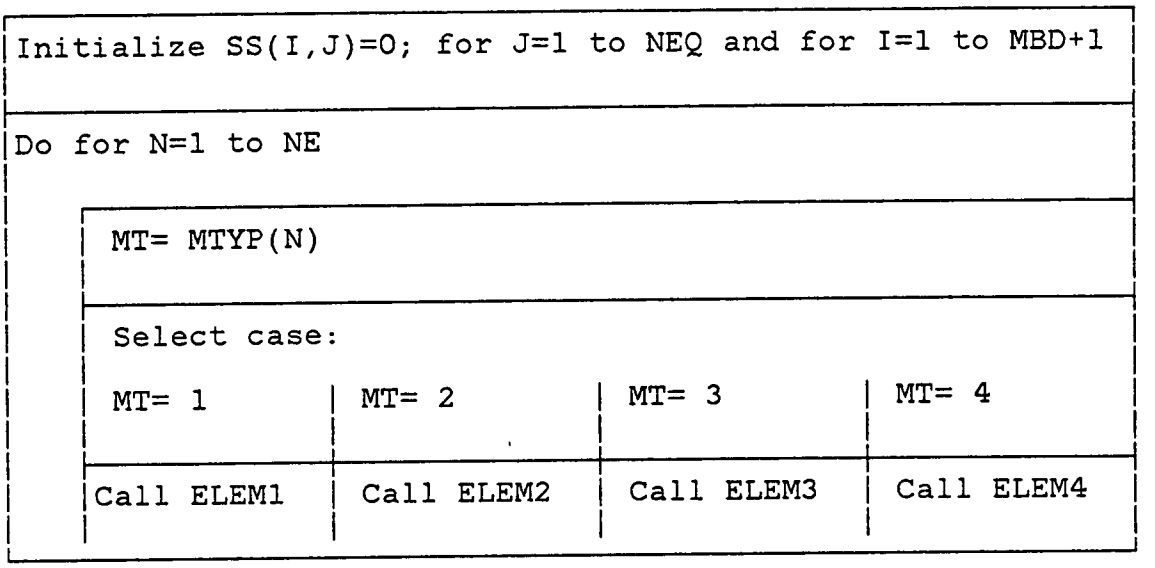
Subroutine STIFF

Function: Initialize SS to zero; for each element call ELEM1, ELEM2, ELEM3, or ELEM4.

Input arguments: C1, C2, ELENG, IP, MBD, MCODE, MTYP, MXNEQ, NE, NEQ.

Output argument: SS.

NS-diagram:



Subroutine ELEM1

Function: For the continuous element N, initialize INDEX(6,6) by equation 7.4(Ref. 5), read AREA, ZI, EMOD, compute global stiffness coefficients G(7), by equations 3.64(Ref. 5), and call ASSEMS.

Input arguments: C1, C2, ELENG, IP, MBD, MCODE, MXNEQ, N, SS.

Output arguments: SS.

NS-diagram:

Initialize INDEX(6,6) by equation 7.4 (REF. 5)
Read AREA, EMOD, ZI
Compute G(1)-G(7) using equations 3.64 (REF. 5)
Call ASSEMS

Subroutine ELEM2

Function: For element N with hinge at a-end, initialize INDEX(6,6) by equation (3.19a), read AREA, ZI, EMOD, compute global stiffness coefficients, G(6), by equations (3.19b), and call ASSEMS.

Input arguments: C1, C2, ELENG, IP, MBD, MCODE, MXNEQ, N, SS.

Output arguments: SS.

NS-diagram:

Initialize INDEX(6,6) by equation (3.19a)
Read AREA, EMOD, ZI
Compute G(1)-G(6) using equations (3.19b)

Call ASSEMS

Subroutine ELEM3

Function: For element N with hinge at b-end, initialize INDEX(6,6) by equation (3.24), compute global stiffness coefficients, G(6), by equations (3.19b), and call ASSEMS.

Input arguments: C1, C2, ELENG, IP, MBD, MCODE, MXNEQ, N, SS.

Output arguments: SS.

NS-diagram:

Initialize INDEX(6,6) by equation (3.24)
Read AREA, EMOD, ZI
Compute G(1)-G(6) using equations (3.19b)
Call ASSEMS

Subroutine ELEM4

Function: For element N with hinge at each end, initialize INDEX(6,6) by equation (3.26a), compute global stiffness coefficients, G(3), by equations (3.26b), and call ASSEMS.

Input arguments: C1, C2, ELENG, IP, MBD, MCODE, MXNEQ, N, SS.

Output arguments: SS.

NS-diagram:

Initialize INDEX(6,6) by equation (3.26a)
Read AREA, EMOD
Compute G(1)-G(3) by equations (3.26b)
Call ASSEMS

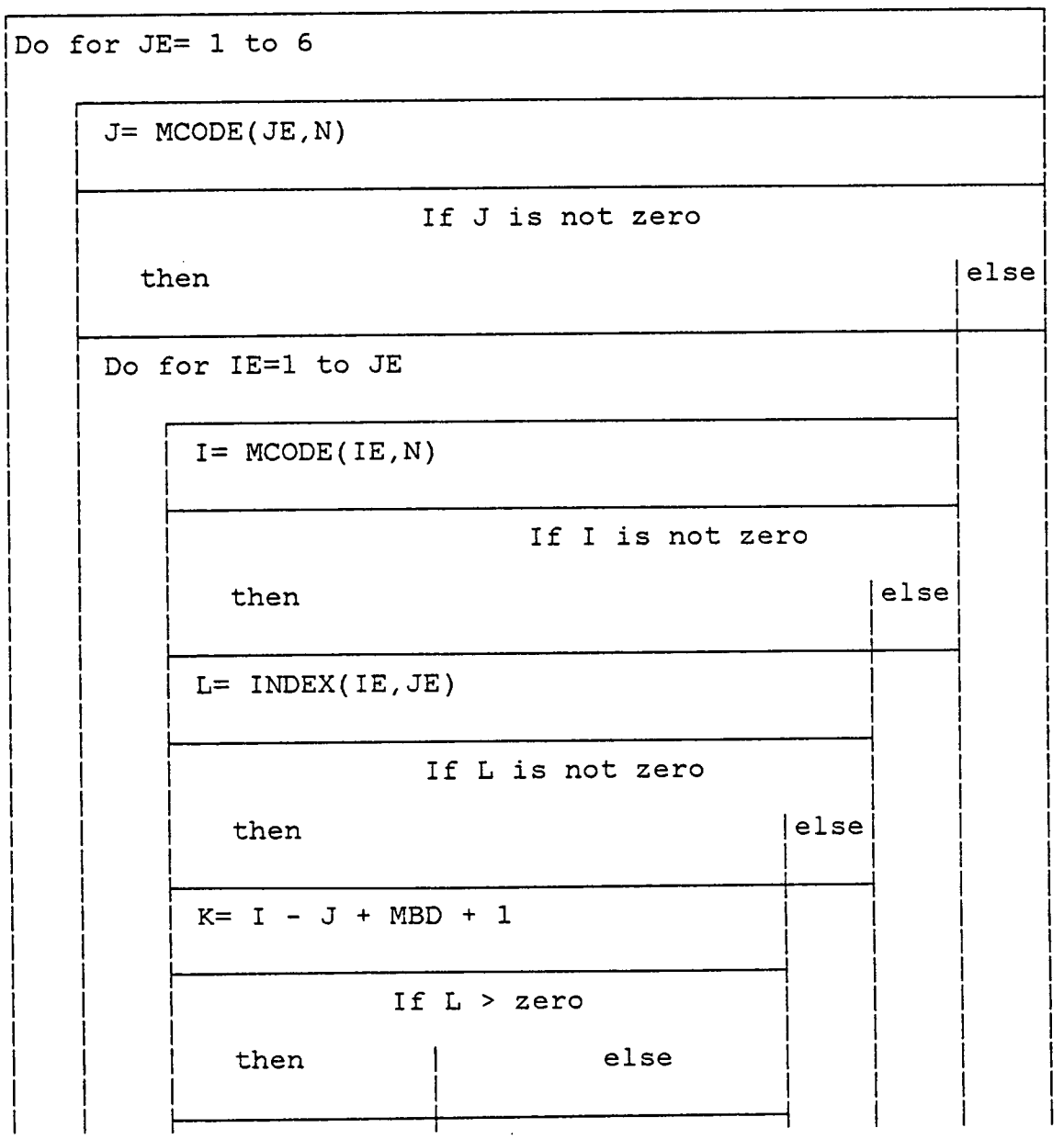
Subroutine ASSEMS

Function: Assign global stiffness coefficients, G, of element N to the system stiffness band matrix, SS, using INDEX, MCODE, and equation 6.7 (Ref. 5).

Input arguments: G, INDEX, MBD, MCODE, MXNEQ, N, SS.

Output argument: SS.

NS-diagram:



		SS(K,J)= G(L) + SS(K,J)	SS(K,J)= -G(-L) + SS(K,J)				

Subroutine MODIFY

Function: Modify the system stiffness matrix, SS, (and Q if necessary) due to prescribed nonzero joint displacements and spring-supported joints: Read NJD; if NJD is not zero, call JDISP. Read NJS; if NJS is not zero, call JSPRG.

Input arguments: IG, JCODE, MBD, MXNEQ, NEQ, Q, SS.

Output arguments: Q, SS.

NS-diagram:

Read NJD	
If NJD is not zero	
then	else
Call JDISP	
Read NJS	
If NJS is not zero	
then	else

Call JSPRG	

Subroutine JDISP

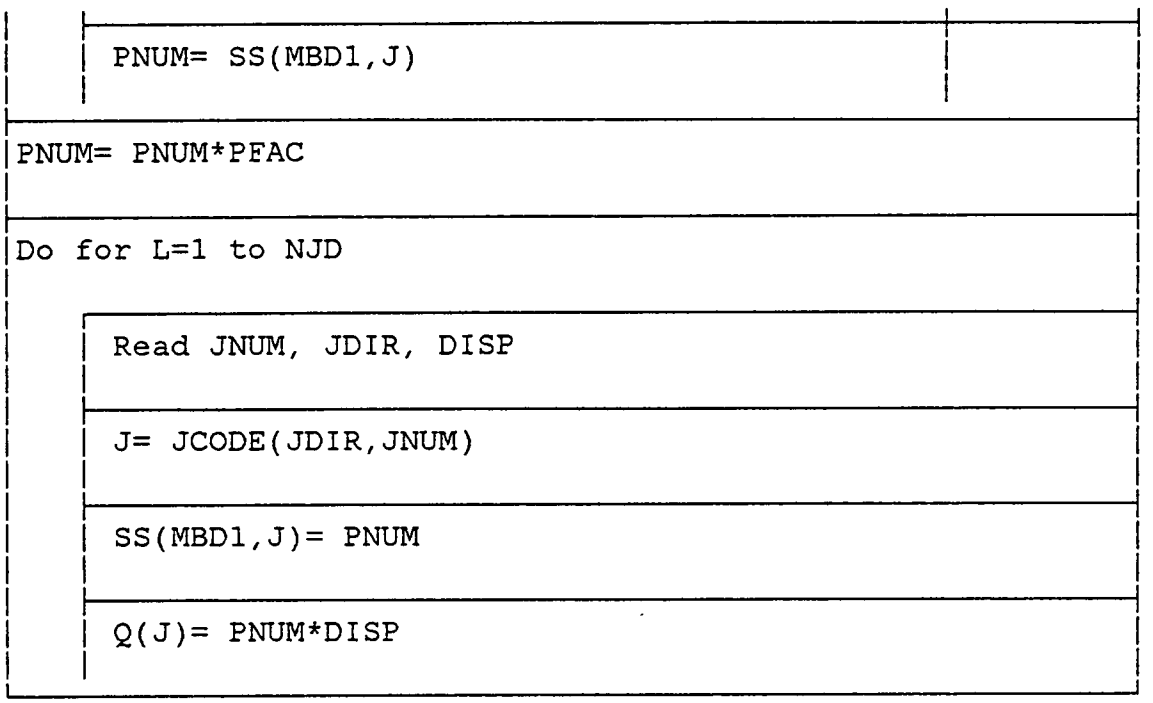
Function: Initialize PFAC using DATA statement. Impose prescribed nonzero joint displacements on the system model using the penalty method (Ref. 2): Compute the penalty number as PNUM=max(Kii)*PFAC; for each d.o.f. J with prescribed value DISP, set Kjj=PNUM and Qj=PNUM*DISP.

Input arguments: IG, JCODE, MBD, MXNEQ, NEQ, NJD, Q, SS.

Output arguments: Q, SS.

NS-diagram:

Initialize PFAC (use data statement)	
MBD1= MBD + 1	
PNUM= 0.	
Do for J=1 to NEQ	
If SS(MBD1,J) > PNUM	
then	else



Subroutine JSPRG

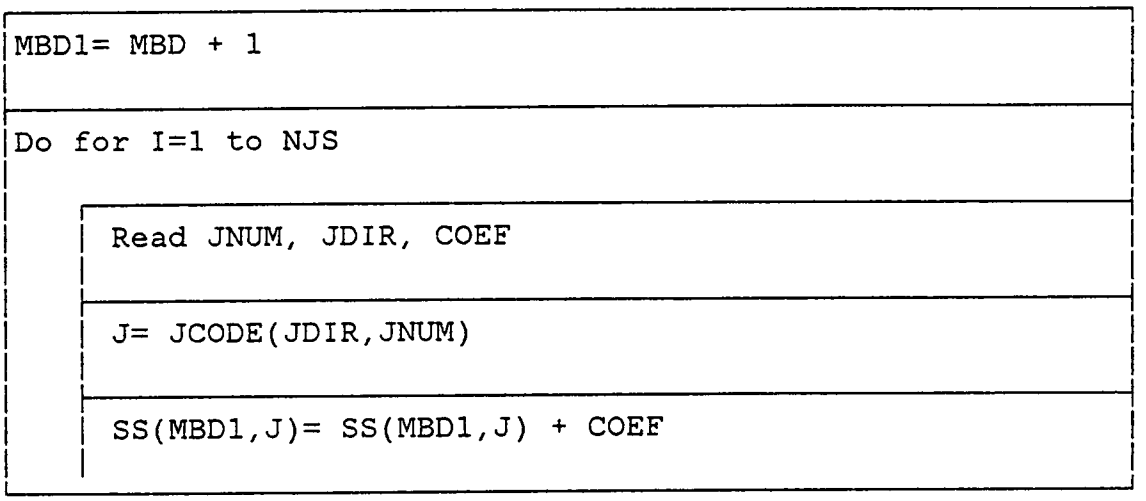
Function: Add the joint spring stiffness coefficients to the system stiffness matrix.

Input arguments: IG, JCODE, MBD, MXNEQ, NJS, SS.

Output arguments: SS.

Note: A SPRG or DISP of JNUM,JDIR corresponds to a degree of freedom; the user should be aware of this.

NS-diagram:



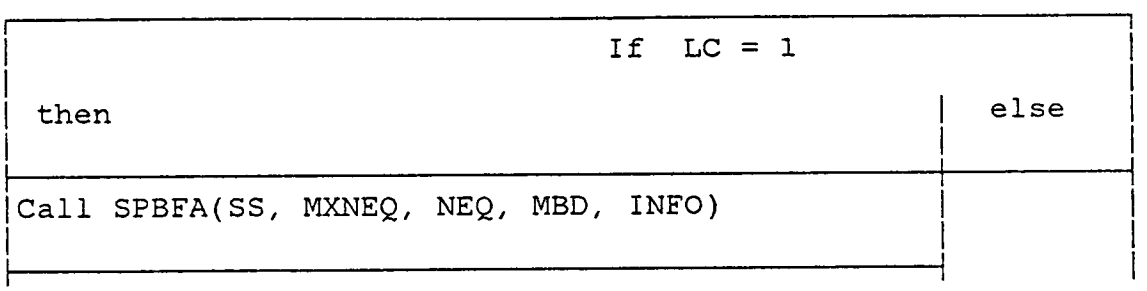
Subroutine SOLVE

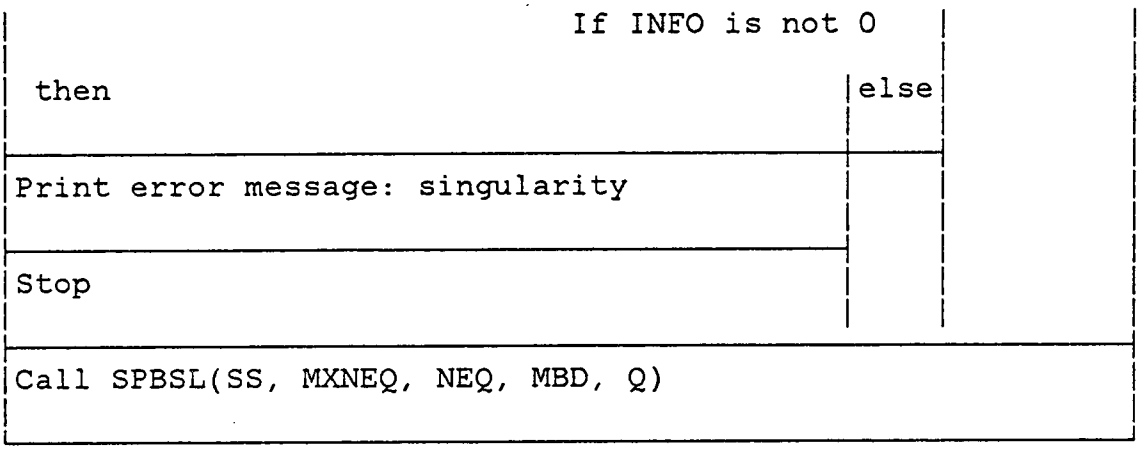
Function: For the first loading condition, LC=1, call SPBFA and SPBSL; for each loading condition, LC>1, call SPBSL.

Input arguments: LC, MBD, MXNEQ, NEQ, Q, SS.

Output argument: Q.

NS-diagram:




If INFO is not 0
then
else
Print error message: singularity
Stop
Call SPBSL(SS, MXNEQ, NEQ, MBD, Q)

4.2 POSTPROCESSOR

FUNCTION: For each loading condition and load combination the POSTPROCESSOR computes the element end forces, joint forces, and internal element responses. For load combinations only, the POSTPROCESSOR finds the maximum bending moments and stresses associated with them, the maximum joint deflections, and the design values of stresses and joint displacements. A design value is defined as the maximum of the maxima.

DESIGN: The structure of the POSTPROCESSOR is shown in figure (4.2). The subroutines are described in section (4.2.3).

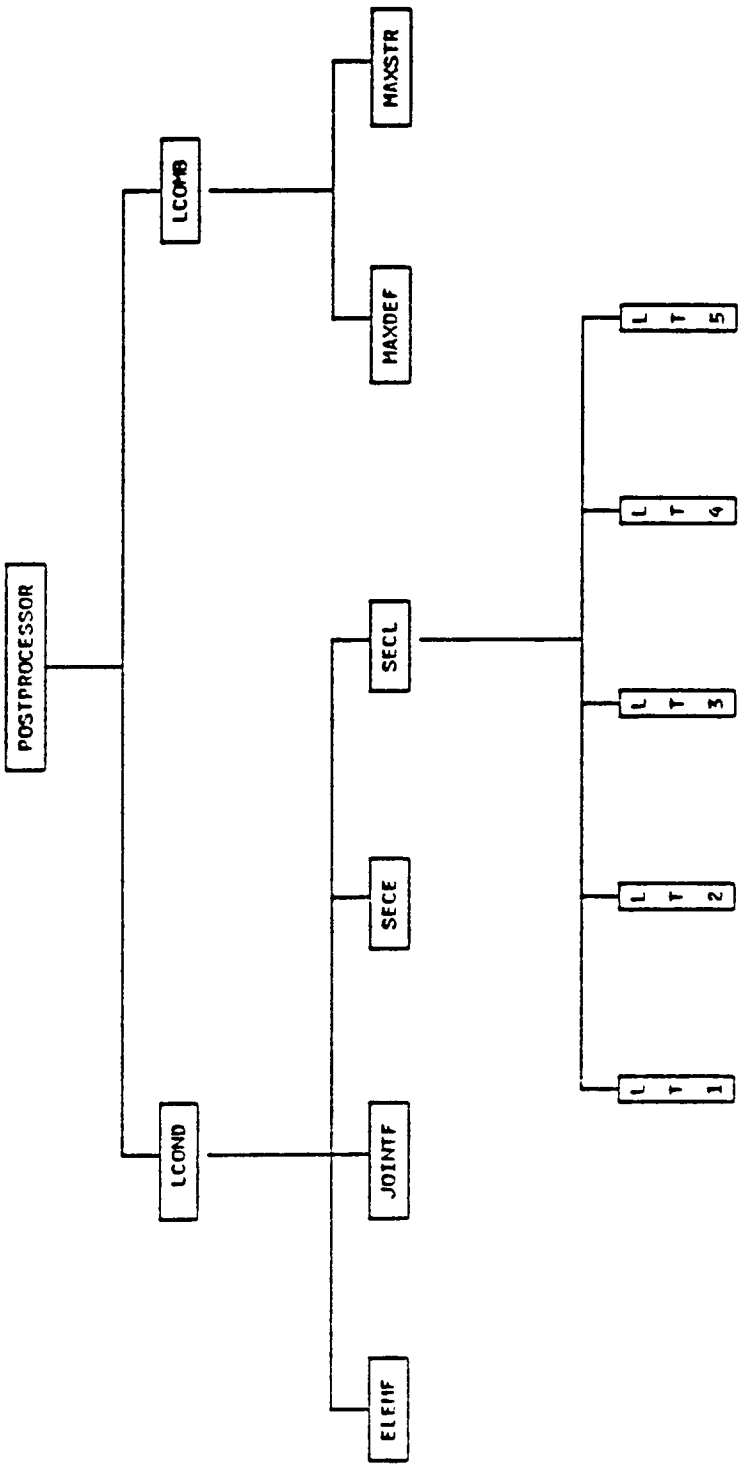


Figure 4.2 Tree Chart of POSTPROCESSOR

4.2.1 List of Variables

Parameters:

MX	Maximum number of elements or joints
MXNEQ	Maximum number of equations (degrees of freedom)

Variables:

AF(11,4*MX)	Axial section force matrix. AF(L,I+LCI)= axial force at Lth section of element I, due to loading condition LC. See LCI
AFMAX	Axial section force associated with BMAX
AREA(I)	Cross-sectional area of element I
BM(11,4*MX)	Section bending moment matrix. BM(L,I+LCI)= bending moment at Lth section of element I, due to loading condition LC. See LCI
BMAX	Maximum bending moment of an element due to a load combination. The bending moments at each section of an element are compared; BMAX designate their maximum
C1(I),C2(I)	Direction cosines of element I
COEF(4)	Load factors
D(6)	Element end displacement vector
D1,D2	Distances from a-end of element to points

of application of loads (figure 2.2)

DESAF(I) Design axial force of element I. It is the axial force associated with DESBM(I)

DESBM(I) Design bending moment of element I. It is the maximum of the maxima.

DSEC(11,MX) Section distances matrix.
DSEC(L,I)= distance from a-end to Lth section of element I

ELENG(I) Length of element I

EMOD(I) Modulus of elasticity of element I

F(7,4*MX) Local element fixed-end force matrix on entry to ELEMF, and actual local element force matrix on return.
For L=1 to 6:
F(L,I+LCI)=Lth force of element I(fig. 3.1a) due to loading condition LC
For L=7:
F(7,I+LCI) stores F3 for element I of type 2
F(7,I+LCI) stores (F6-2*F3) for type 4
See LCI

FS(6,MX) Local element force matrix due to load combinations

ILC(I) Loading condition associated with DESBM(I)

IR Flag used in LCOND:
IR=0 if no internal responses required

IR=1 if internal responses required

JCODE(3,MX) Joint code matrix (see Ref. 5)

JXDES Joint corresponding to XDES

JXMAX Joint corresponding to XMAX

JYDES Joint corresponding to YDES

JYMAX JOINT corresponding to YMAX

LC Loading condition

LCI = MX*(LC-1)

LCK = MXNEQ*(LC-1)

LCM Load combination

LDES(I) Section associated with DESBM(I)

LMAX Section corresponding to BMAX

LT Element load (or action) type:

LT=1 for load in figure 2.2a

LT=2 for load in figure 2.2b

LT=3 for load in figure 2.2c

LT=4 for load in figure 2.2d

LT=5 for load in figure 2.2e

LT=6 for load in figure 2.2f

LXDES Load combination (LCM) causing XDES

LYDES Load combination (LCM) causing YDES

MCODE(6,MX) Member code matrix

(see sections 2.3 & 2.4 of Ref. 5)

MINC(2,MX) Member incidence matrix:

MINC(1,I)= joint at a-end of element I

	MINC(2,I)= joint at b-end of element I
MN	Member number
MTYP(I)	Member type of element I:
	MTYP(I)=1 for element in figure 3.1a
	MTYP(I)=2 for element in figure 3.1b
	MTYP(I)=3 for element in figure 3.1c
	MTYP(I)=4 for element in figure 3.1d
NCOMB	Number of load combinations
NE	Number of elements
NEQ	Number of equations (degrees of freedom)
NJ	Number of joints
NLC	Number of loading conditions
NML	Number of member loads (actions)
NSEC(I)	Number of sections of element I
P(3,4*MX)	Joint force matrix:
	P(L,J+LCI) = force at joint J in direction L,
	due to loading condition LC. See LCI
P1,P2	Member loads (actions) as shown in figure 2.2
PS(3,MX)	Joint force matrix due to load combinations
Q(4*MXNEQ)	Joint load,displacement vector (see Ref. 5)
	Q(K+LCK)= Kth joint load or displacement
	due to loading condition LC. See LCK
QS(MXNEQ)	Joint displacement vector due to load
	combination
SEC(44,MX)	Section responses matrix due to load

combinations.

SEC(L+11*N,I)= response at Lth section of element I.

If N=0, response is axial force.

If N=1, response is bending moment.

If N=2, response is shear force.

If N=3, response is transverse deflection

SF(11,4*MX) Shear force matrix.

SF(L,LCI+I)= shear force at Lth section of element I due to loading condition LC. See LCI

TD(11,4*MX) Transverse deflection matrix.

TD(L,LCI+I)= transverse (local y-axis) deflection at Lth section of element I due to loading condition LC. See LCI

XDES The design X-displacement. It is the maximum of all load combinations

XMAX Maximum joint deflection in the X direction due to a load combination

YDES The design Y-displacement. It is the maximum of all load combinations

YMAX Maximum joint deflection in the Y-direction due to a load combination

ZI(I) Moment of inertia about the local z-axis of element I

ZS(I)	Section modulus about the local z-axis of element I

Input/output parameters:

IG	The unit number of file GEOMETRY.FRA
IP	The unit number of file PROP.FRA
IPRT	The unit number of output (e.g. printer)

4.2.2 Data Files

Data file GEOMETRY.FRA

		Read in:
1.	NE, NJ, NLC	Main program
2.	ELENG(I), C1(I), C2(I), MTYP(I); I=1	Main program
	: : : : :	
	: : : : :	
	: : : : :	
	NE	
3.	JCODE(1,J), JCODE(2,J), JCODE(3,J); J=1	Main program
	: : : :	
	: : : :	
	: : : :	
	NJ	
4.	MINC(1,I), MINC(2,I); I=1	Main program
	: : :	
	: : :	
	: : :	
	NE	

Data file PROP.FRA

						Read in:
5.	AREA(I),	EMOD(I),	ZI(I),	ZS(I);	I=1	Main program
	:	:	:	:	:	
	:	:	:	:	:	
	:	:	:	:	:	
					NE	
6.	NSEC(I);	I=1				Main program
	:	:				
	:	:				
		NE				
7.	DSEC(L,I);	L=1	;	I=1 to NE		Main program
	:	:				
		NSEC(I)				
14.	NCOMB					Main program
15.	COEF(1),	COEF(2),	COEF(3),	COEF(4);	LCM=1	LCOMB
	:	:	:	:	:	
	:	:	:	:	:	
					NCOMB	

Data file PROC.OUT

Read in:

8. NEQ — Main program

9. MCODE(1,I), MCODE(2,I), MCODE(3,I),
MCODE(4,I), MCODE(5,I), MCODE(6,I); I=1 — Main program
: : : : : : :
: : : : : : :
NE

10. For LC=1 to NLC: — Main program
(LCI=MX*(LC-1); LCK=MXNEQ*(LC-1))
1) F(1,I+LCI), F(2,I+LCI), ..., F(7,I+LCI); I=1
: : : : :
: : : : :
NE

2) Q(K+LCK); K=1
: :
: :
NEQ

Data files LCi.FRA, where i=1 to 4

	Read in:
11. NML	SECL
12. MN, LT, P1, D1, P2, D2; 1	SECL
: : : : : : :	
: : : : : : :	
: : : : : : :	
NML	

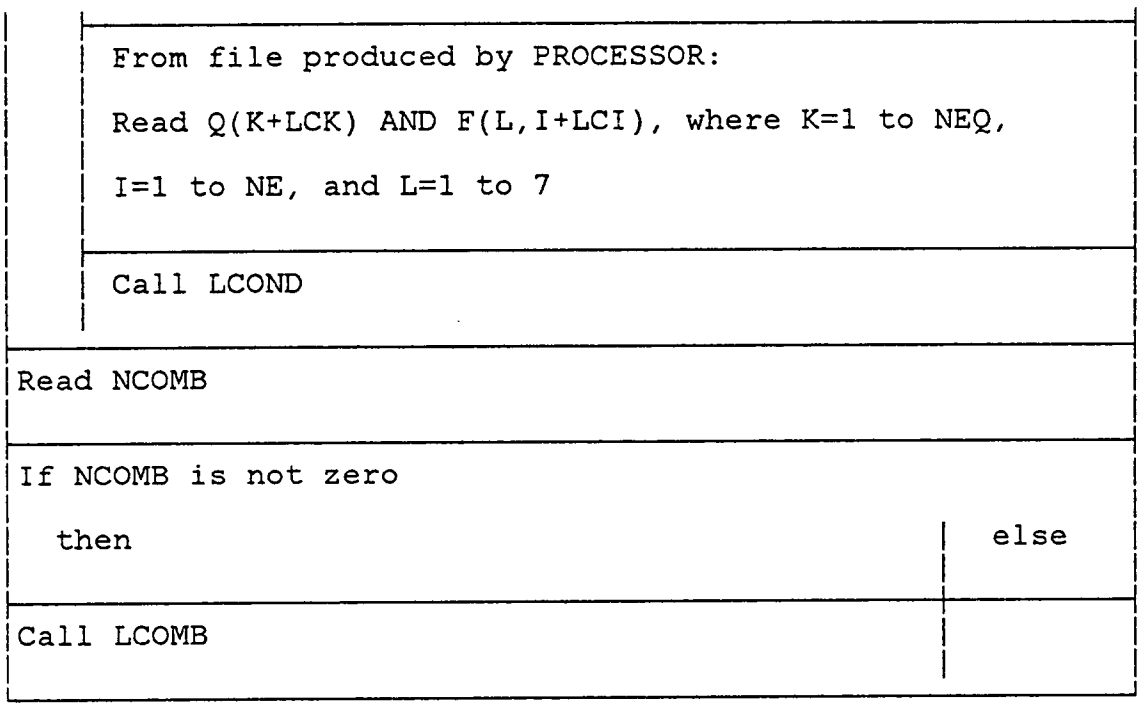
4.2.3 N-S Diagrams

Main Program:

Function: The main program reads output provided by PROCESSOR and obtains the required responses of the structure. It calls LCOND for element and joint responses (F, P, AF, BM, SF, TD) due to loading conditions. It calls LCOMB for responses due to load combinations.

NS-diagram:

From file GEOMETRY.FRA: Read NE, NJ, NLC, ELENG(I),C1(I),C2(I),MTYP(I), MINC(I), and JCODE(L,J); I=1 to NE, J=1 to NJ, and L=1 to 3 From file PROP.FRA: Read AREA(I), EMOD(I), ZI(I), ZS(I), NSEC(I), and DSEC(L,I) ; I=1 TO NE and L=1 to NSEC(I)
From file produced by PROCESSOR: Read NEQ and MCODE(L,I); I=1 to NE and L=1 to 6
Initialize MX = 30 and MXNEQ = 3*(MX - 1) Initialize P(L,J) = 0. ; L=1 to 3 and J=1 to 4*MX
Do for LC=1 to NLC LCI = (LC - 1)*MX and LCK = (LC - 1)*MXNEQ



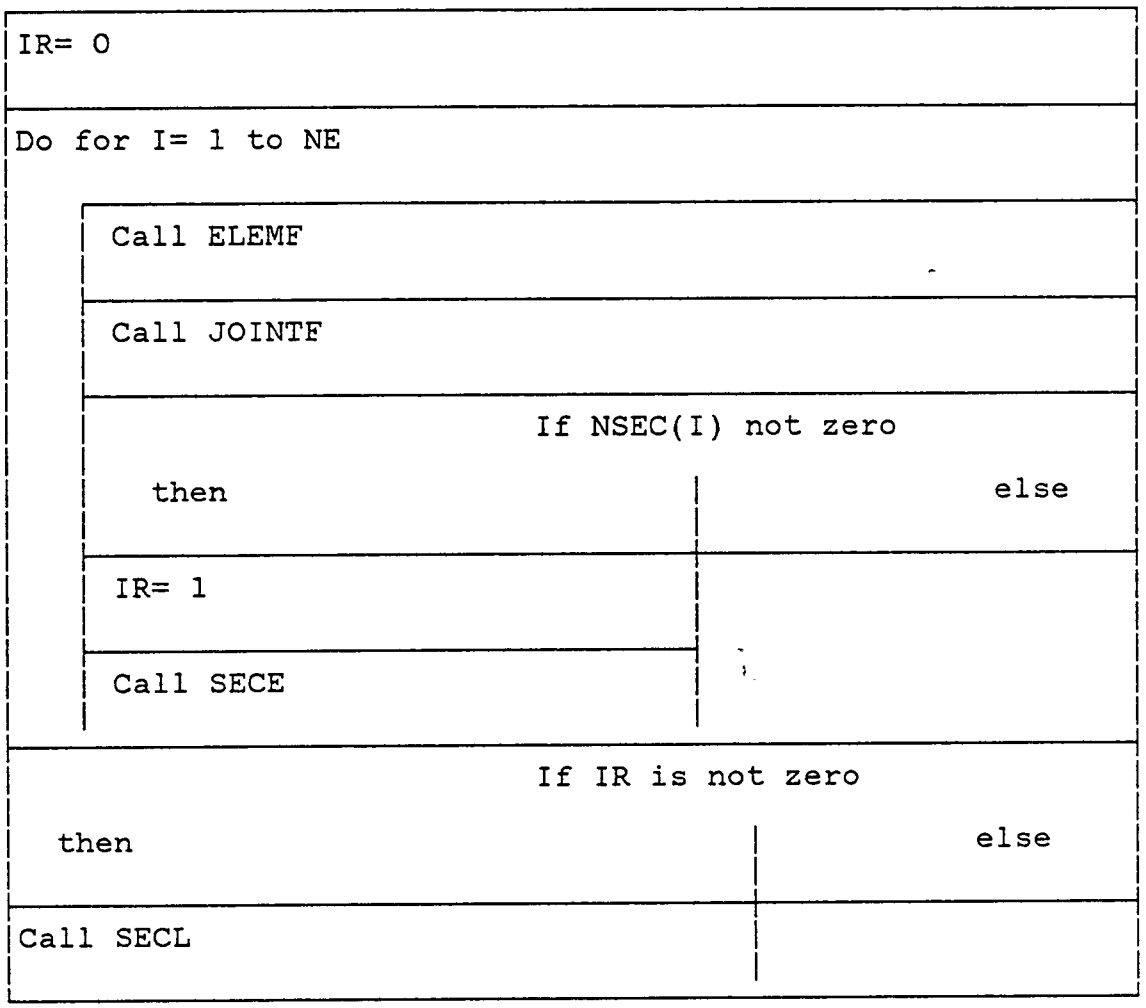
Subroutine LCOND

Function: compute the element-end forces, the joint forces, the section forces(axial, shear, bending moment), and the section transverse deflection: For each element call ELEMF, JOINTF, and if internal responses are required call SECE and call SECL.

Input arguments: AREA, C1, C2, DSEC, ELENG, EMOD, IG, F, LC, LCI, LCK, MCODE, MINC, MTYP, NE, NSEC, P, Q, ZI.

output argument: AF, BM, SF, TD, F, P.

NS-diagram:



Subroutine ELEMF

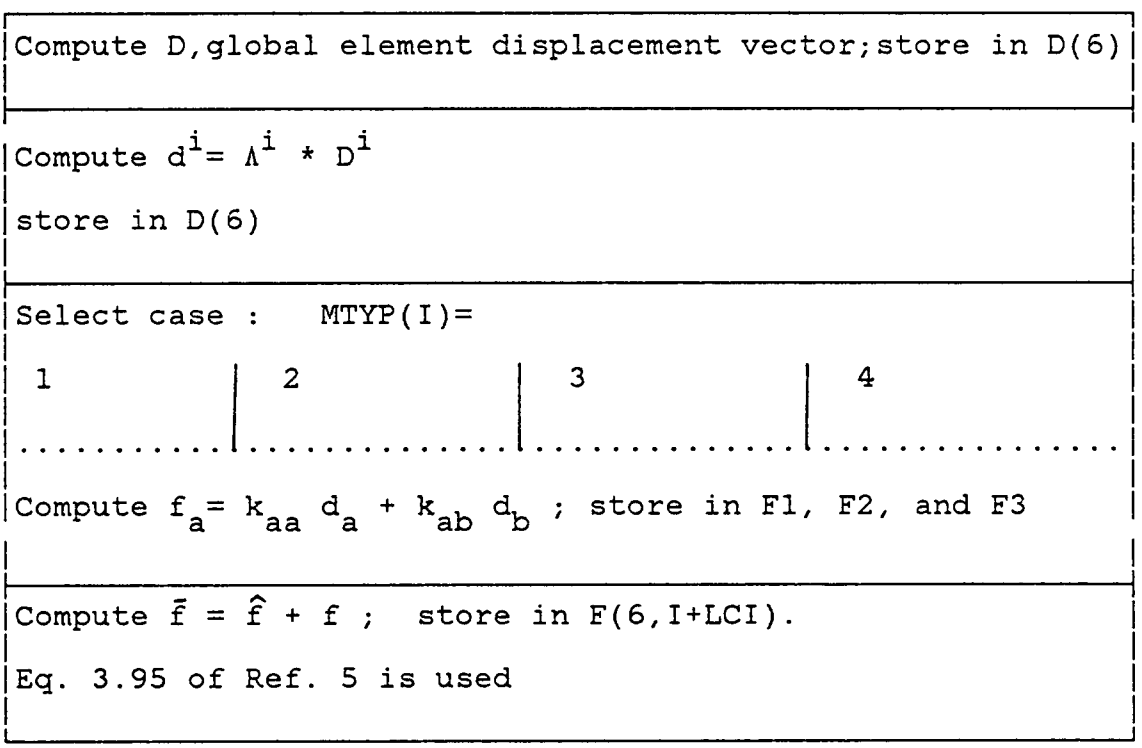
Function: compute the actual local end forces of element I, F(6,I+LCI): determine the global element-end displacements, D(6), from the joint displacement vector Q via MCODE, and transform them into local element-end displacements; compute

the corresponding local forces at the a-end of the specific type of element (MTYP(I)= 1, 2, 3, or 4); use equilibrium to compute the local forces at the b-end and use equation 3.95 of (REF.5) to compute the actual element-end forces.

Input arguments: AREA, C1, C2, F, ELENG, EMOD, I, LCI, LCK, MCODE, MTYP, Q, ZI.

output arguments: D, F.

NS-diagram:



Subroutine JOINTF

Function: Transform the local forces of element I, F(6,I+LCI), to global forces and assign them to the joint force matrix, P, by equations (of Ref.5) 2.29, 2.34, 2.37, and MINC.

Input arguments: C1, C2, F, I, IG, LCI, MINC, P.

Output argument: P

NS-diagram:

```
Statement function for eqs. 2.34, 2.37 of (Ref. 5)
FG( C1I, C2I, FLX, FLY)= C1I*FLX + C2I*FLY
```

```
Read J and K (MINC of element I)
```

```
P(1,J+LCI)= P(1,J+LCI)
                + FG(C1(I), -C2(I), F(1,I+LCI),F (2,I+LCI))
```

```
P(2,J+LCI)= P(2,J+LCI)
                 + FG(C2(I), C1(I), F(1,I+LCI), F(2,I+LCI))
```

```
P(3,J+LCI)= P(3,J+LCI) + F(3,I+LCI)
```

```
P(1,K+LCI)= P(1,K+LCI)
                + FG(C1(I), -C2(I), F(4,I+LCI), F(5,I+LCI))
```

```
P(2,K+LCI)= P(2,K+LCI)
```

+ FG(C2(I), C1(I), F(4,I+LCI), F(5,I+LCI))
P(3,K+LCI)= P(3,K+LCI) + F(6,I+LCI)

Subroutine SECE

Function: For elements of types 2 or 4 store d7 (fig. 3.1b&d) in D(3). Compute section forces and deflections resulting from element nodal forces and displacements.

Input arguments: D, DSEC, ELENG, EMOD, F, I, LCI, MTYP, NSEC, ZI.

Output arguments: AF, BM, SF, TD.

NS-diagram:

EI= EMOD(I)*ZI(I)	
Select type MT= MTYP(I)	
MT= 2	MT= 4
D(3)= 3.*(D(5) - D(2)) /2./ELENG(I) - 0.5*D(6) - ELENG(I)*F(7,I+LCI)	D(3)= (D(5) - D(2))/ELENG(I) + F(7,I+LCI)*ELENG(I) /6./EI

```
|                  /4./EI     |                                          |
|-----------------------------|------------------------------------------|
| F1= F(1,I+LCI),  F2= F(2,I+LCI),  and F3= F(3,I+LCI)                   |
|------------------------------------------------------------------------|
| Do for L=1 to NSEC(I)                                                  |
|    |-------------------------------------------------------------------|
|    | X= DSEC(L,I)                                                      |
|    |-------------------------------------------------------------------|
|    | AF(L,I+LCI)= -F1                                                  |
|    |-------------------------------------------------------------------|
|    | SF(L,I+LCI)= F2                                                   |
|    |-------------------------------------------------------------------|
|    | BM(L,I+LCI)= F2*X - F3                                            |
|    |-------------------------------------------------------------------|
|    | TD(L,I+LCI)= D(2)+ D(3)*X + X**2/(2.*EI)*(F2*X/3.-F3)             |
|    |                                                                   |
|-------------------------------------------------------------------------|
```

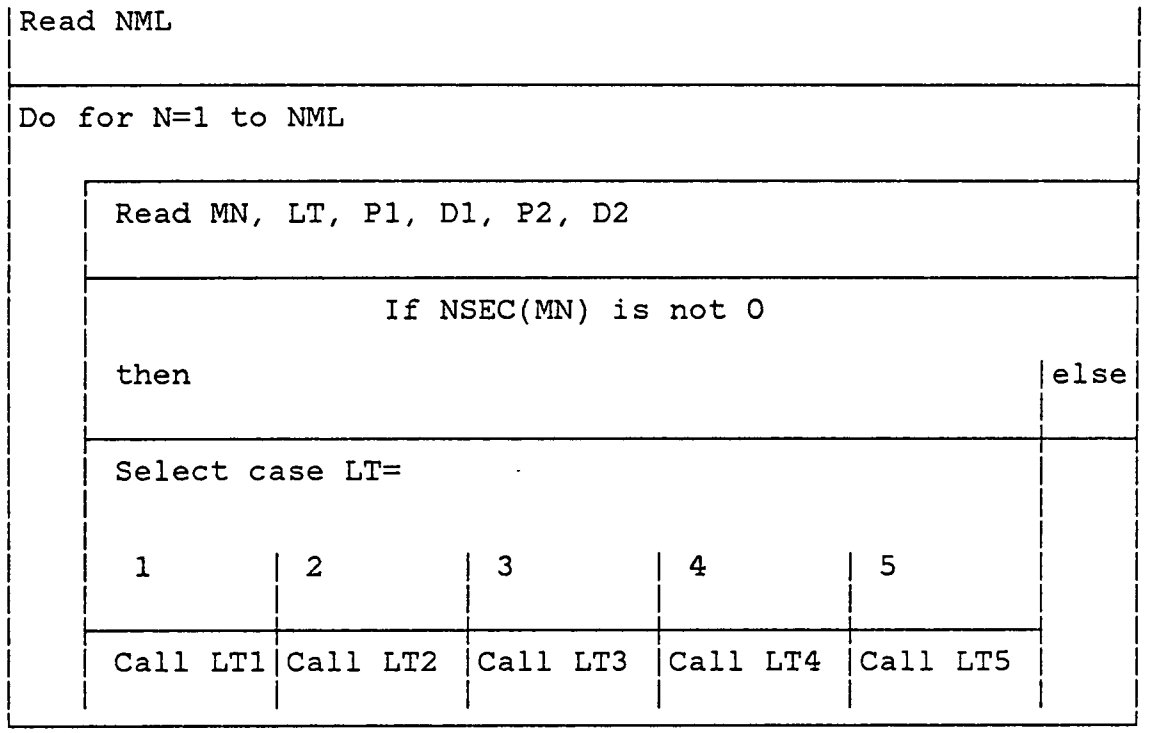
Subroutine SECL

Function: Compute and accumulate the section forces and deflections induced by element loads: Read NML; for each load call LT1, LT2, LT3, LT4, or LT5 if NSEC(MN) is not zero.

Input arguments: AF, BM, DSEC, EMOD, LC, LCI, NSEC, SF, TD, ZI.

Output arguments: AF, BM, SF, TD.

NS-diagram for SECL:



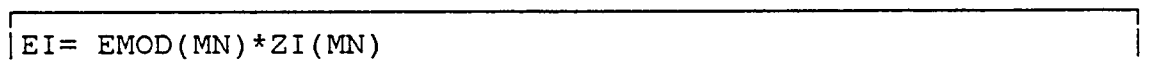
Subroutine LT1

Function: Compute and accumulate section forces and deflections caused by LT= 1.

Input arguments: BM, D1, DSEC, EMOD, LCI, MN, NSEC, P1, SF, TD, ZI.

Output arguments: BM, SF, TD.

NS-diagram:



```
Do for L=1 to NSEC(MN)
    X= DSEC(L,MN)
    If X is > D1
      then                                          | else
    SF(L,MN+LCI)= SF(L,MN+LCI) + P1
    BM(L,MN+LCI)= BM(L,MN+LCI) + P1*(X-D1)
    TD(L,MN+LCI)=TD(L,MN+LCI)+P1*(X-D1)**3/(6.*EI)
```

Subroutine LT2

Function: Compute and accumulate section forces and deflections caused by LT= 2.

Input arguments: BM, DSEC, EMOD, LCI, MN, NSEC, P1, SF, TD, ZI.

Output arguments: BM, SF, TD.

NS-diagram:

```
EI= EMOD(MN)*ZI(MN)
```

```
Do for L=1 to NSEC(MN)
    X= DSEC(L,MN)
    SF(L,MN+LCI)= SF(L,MN+LCI) + P1*X
    BM(L,MN+LCI)= BM(L,MN+LCI) + P1*X**2/2.
    TD(L,MN+LCI)= TD(L,MN+LCI) + P1*X**4/(24.*EI)
```

Subroutine LT3

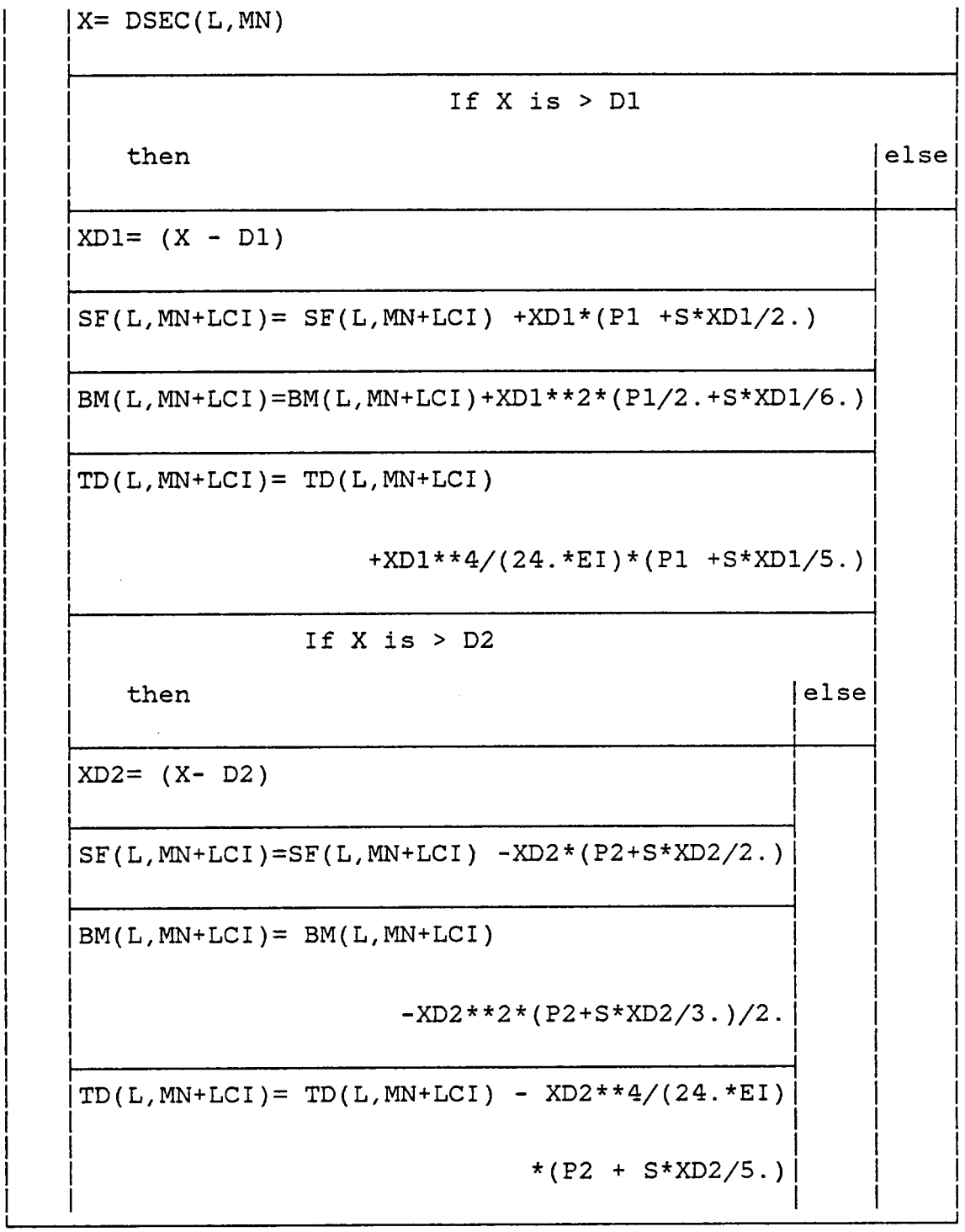
Function: Compute and accumulate section forces and deflections caused by LT= 3.

Input arguments: BM, D1, D2, DSEC, EMOD, LCI, MN, NSEC, P1, P2, SF, TD, ZI.

Output arguments: BM, SF, TD.

NS-diagram for LT3:

```
EI= EMOD(MN)*ZI(MN)
S= (P2 - P1)/(D2 - D1)
Do for L=1 to NSEC(MN)
```


X= DSEC(L,MN)
If X is > D1
then
else
XD1= (X - D1)
SF(L,MN+LCI)= SF(L,MN+LCI) +XD1*(P1 +S*XD1/2.)
BM(L,MN+LCI)=BM(L,MN+LCI)+XD1**2*(P1/2.+S*XD1/6.)
TD(L,MN+LCI)= TD(L,MN+LCI)
+XD1**4/(24.*EI)*(P1 +S*XD1/5.)
If X is > D2
then
else
XD2= (X- D2)
SF(L,MN+LCI)=SF(L,MN+LCI) -XD2*(P2+S*XD2/2.)
BM(L,MN+LCI)= BM(L,MN+LCI)
-XD2**2*(P2+S*XD2/3.)/2.
TD(L,MN+LCI)= TD(L,MN+LCI) - XD2**4/(24.*EI)
*(P2 + S*XD2/5.)

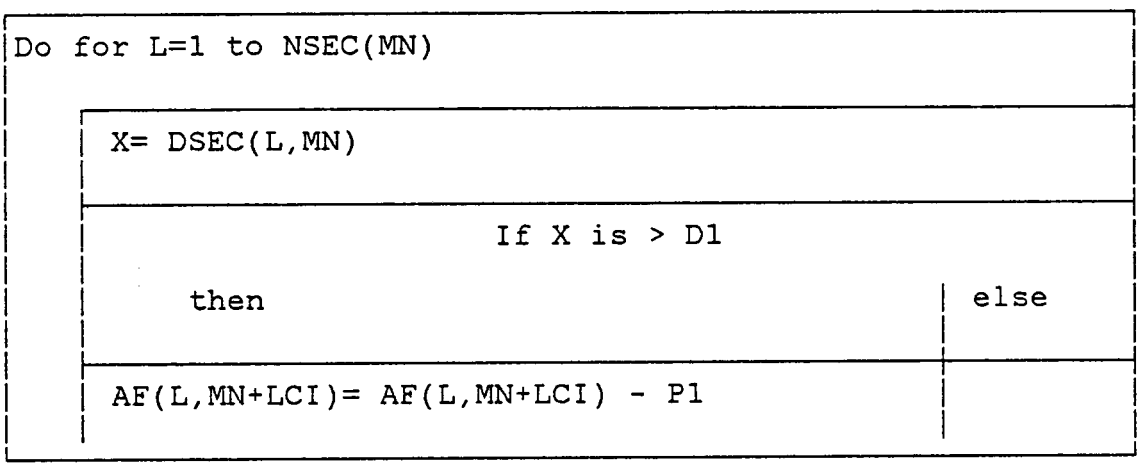
Subroutine LT4

Function: Compute and accumulate axial section forces caused by LT= 4.

Input arguments: AF, D1, DSEC, LCI, MN, NSEC, P1.

Output argument: AF.

NS-diagram for LT4:



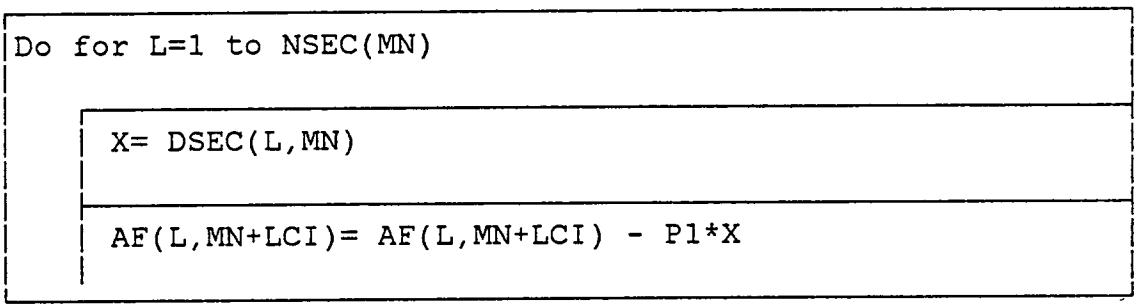
Subroutine LT5

Function: Compute and accumulate axial section forces caused by LT= 5.

Input arguments: AF, DSEC, LCI, MN, NSEC, P1.

Output argument: AF

NS-Diagram for LT5:



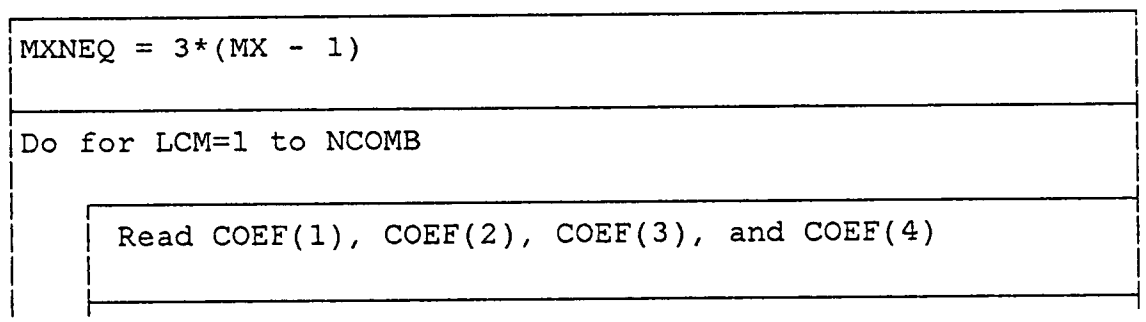
Subroutine LCOMB:

Function: LCOMB computes the responses (F, P, AF, BM, TD, SF) due to load combinations.

Input arguments: AF, AREA, BM, DSEC, F, IP, IPRT, JCODE, LCI, MX, NCOMB, NE, NEQ, NJ, NLC, NSEC, P, Q, SF, TD, ZS.

Output arguments: DESAF, DESBM, FS, ILC, LDES, PS, QS, SEC.

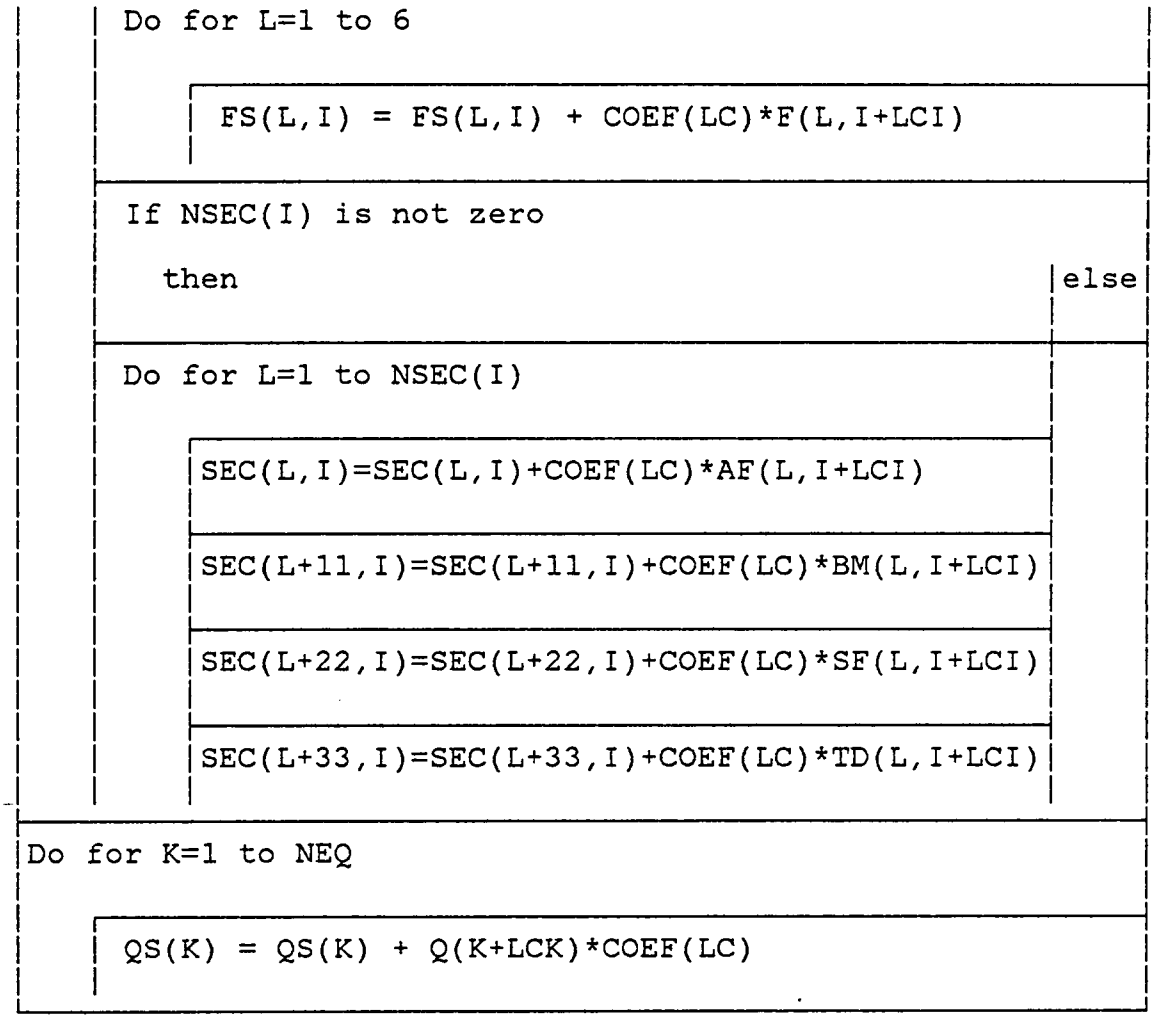
NS-diagram:



```
Initialize the following arrays to zero:
PS(3,NJ), FS(6,NE), SEC(44,NE), and QS(NEQ)

Do for LC=1 to NLC

    LCI = MX*(LC - 1)  and  LCK = MXNEQ*(LC - 1)

    If integer(COEF(LC)) is not zero
      then                                          |else|

    B L O C K   1

Call MAXDEF

Call MAXSTR
```

Block 1 computes the contribution of the current loading condition, LC, to the responses due to the load combination, LCM.

```
Do for J=1 to NJ

    PS(1,J) = PS(1,J) + COEF(LC)*P(1,J+LCI)
    PS(2,J) = PS(2,J) + COEF(LC)*P(2,J+LCI)
    PS(3,J) = PS(3,J) + COEF(LC)*P(3,J+LCI)

Do for I=1 to NE
```



Subroutine MAXDEF

Function: MAXDEF finds the maximum joint displacements , XMAX and YMAX, of load combination LCM in the X and Y directions respectively. JXMAX and JYMAX denote the joints corresponding to XMAX and YMAX respectively. XDES and YDES designate the maximum displacements in the X and Y directions

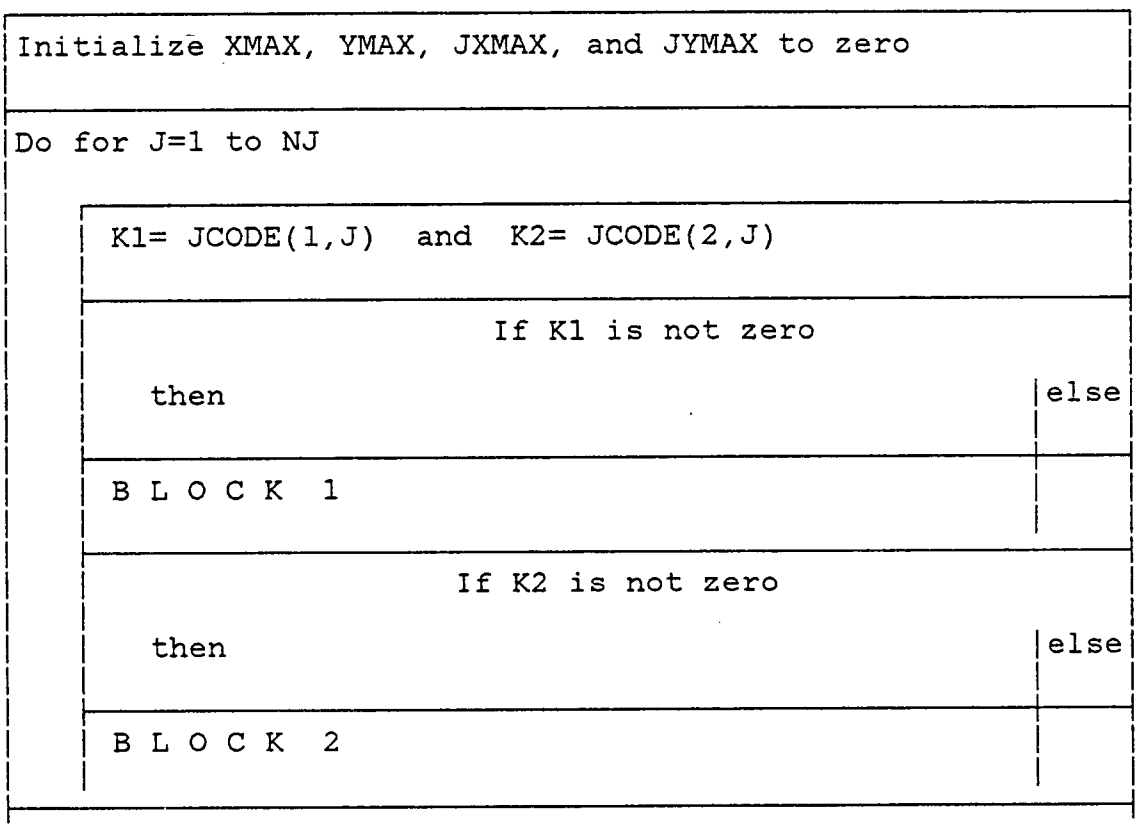
of all load combinations. JXDES and JYDES are the joints corresponding to XDES and YDES.

The joint dispalcements in the X and Y directions are identified using the JCODE of the structure. Then, the maximum in each direction is found as well as the load combination causing it.

Input arguments: IPRT, JCODE, LCM, NJ, QS.

Output arguments: JXDES, JYDES, LXDES, LYDES, XDES, YDES.

NS-diagram:



B L O C K 3

Block 1 is:

XD = QS(K1)	
If ABS(XD) > ABS(XMAX)	
then	else
XMAX = XD	
JXMAX = J	

Block 2 is:

YD = QS(K2)	
If ABS(YD) > ABS(YMAX)	
then	else
YMAX = YD	
JYMAX = J	

Block 3 is:

If LCM = 1	
then	else
XDES = XMAX	If ABS(XMAX) > ABS(XDES)

JXDES = JXMAX	then	else
LXDES = LCM YDES = YMAX	XDES = XMAX JXDES = JXMAX	
JYDES = JYMAX	LXDES = LCM	
LYDES = LCM	If ABS(YMAX) > ABS(YDES)	
	then	else
	YDES = YMAX JYDES = JYMAX LYDES = LCM	

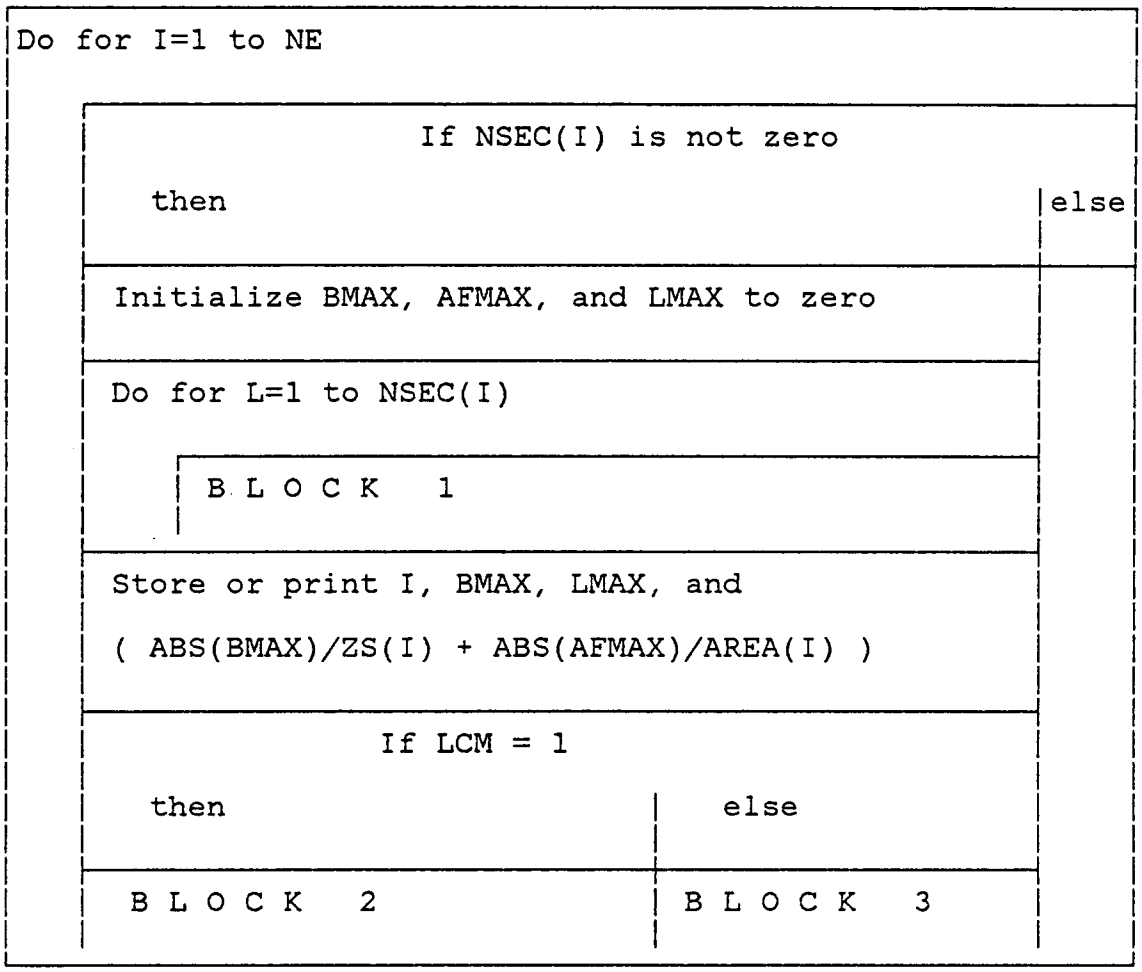
Subroutine MAXSTR

Function: MAXSTR finds the maximum bending moment, BMAX, and its location, LMAX, due to load combination LCM. This is done for each element. Moreover, this subroutine finds the maximum of BM, denoted by DESBM(I), and its location LDES(I), and identifies ILC(I), the load combination causing it. The corresponding AF, denoted by DESAF(I) is also identified so that the total stress can be calculated.

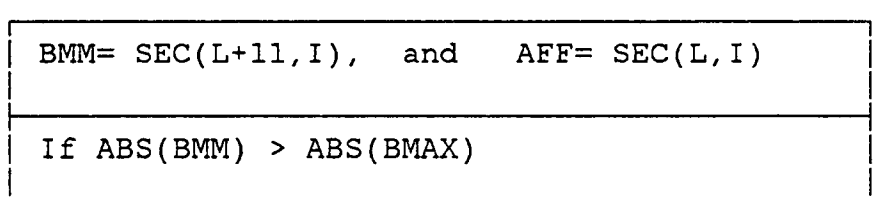
Input arguments: AREA, IPRT, LCM, NE, NSEC, SEC, ZS.

Output arguments: DESAF, DESBM, ILC, LDES.

NS-diagram:



Block 1 is:



then	else
BMAX= BMM, AFMAX= AFF, and LMAX= L	

Block 2 is :

DESBM(I)= BMAX
LDES(I)= LMAX
ILC(I)= LCM
DESAF(I)= AFMAX

Block 3 is :

If ABS(BMAX) > ABS(DESBM(I)) then	else
DESBM(I)= BMAX	
LDES(I)= LMAX	
ILC(I)= LCM	
DESAF(I)= AFMAX	

Chapter V

TESTING

5.1 INTRODUCTION

There is no doubt that every programmer likes to make sure that his computer program is correct. After working for several months on a computer program, discovering bugs can be really frustrating. It may be even more frustrating if these bugs are discovered by the user who has been assured of the correctness of the program! So, the following questions may arise: Why doesn't a programmer follow a systematic procedure in developing a computer program that makes him highly confident in his work? And how can the correctness of a computer program be assured?

5.2 STRUCTURED PROGRAMMING

The systematic procedure suggested for developing computer programs is structured programming. Structured programming makes the computer program simpler. When the program is modified, it will be easier to understand. When the program is checked, it will be quicker to identify any errors it might have. There are two main steps towards making a computer program simpler (Ref.5): First, by using control structures. Second, by modularizing the program.

There are three basic control structures(Ref. 5):

1. Sequence structure
2. Alternate structure
3. Loop structure

A look at the N-S diagrams of chapter 4 will show how these basic control structures are used.

Modularization means dividing the program into modules. Each module should be independent and has a specific well-defined function. A tree chart, such as that shown in chapter 4, is a direct product of modularization. There are two main methods of modularization in structured programming (Ref. 5): the top-down approach and the bottom-up approach. The top-down approach is used by the author to develop the computer program of this thesis.

5.3 PROGRAM CORRECTNESS

There are two main approaches in which programmers can convince themselves that there are no errors in their programs (Ref. 10):

"1. Exercising the program with TEST DATA and inspecting the output for deviations from the EXPECTED VALUES.
2. Demonstrating, by more or less rigorous mathematical PROOFS, that the program is consistent with a

SPECIFICATION of its intended actions.

The first method identified above is usually known as TESTING (or testing and debugging if error location and correction are also involved). The second method is variously described as VERIFICATION, correctness proving, proofs of correctness, etc. However, the term PROVING CORRECTNESS is occasionally used to denote the general principle of assuring that there are no errors, by whatever means." The main concern here is practicality. That is, a clear procedure of what to do in order to test a program. Accordingly, for the program developed in this thesis, the first method of proving program correctness, namely TESTING, is adopted. Testing is discussed in the following sections.

5.4 TESTING

"The PRIMARY GOAL of testing," according to reference 1, "is BUG PREVENTION. A prevented bug is more desirable than a detected and corrected bug; because there is no code to correct if the bug is prevented, no retesting to confirm that the correction was valid, no one is embarrassed, no memory is consumed, and prevented bugs cannot adversely affect a schedule." Beizer (Ref. 1) continues to say that "more than the act of testing, the act of DESIGNING TESTS is one of the most effective bug preventers known... Discover and eliminate bugs at EVERY STAGE in the creation of software."

"The SECONDARY GOAL," still according to Beizer (Ref. 1), "is clearly identifying that there is a bug. A test design must document expectations, test procedure in detail, and the results of the actual tests, all of which are subject to error." Some programmers, usually beginners, think that the secondary goal mentioned above by Beizer is the only goal of testing! Thus, their testing process does not start until the program is completely coded! Worse than that, some of these programmers do not have any testing plan before they start testing.

The question remains: Is there a procedure that, if followed, achieves the two goals of testing 100%? The answer to this question is NO! For, according to reference 10, " there is, as yet, no general theory of testing and no agreement as to what constitutes an adequate test criterion. However, several definitions have been put forward for various TESTING STRATEGIES." There is a well-known maxim (Ref. 10) "that program testing can be used to discover the presence of errors, but not their absence." There are two different economic concepts of testing (Ref. 15): "one concept deals with the economics of removing defects at a single point in

time, and the second concept is the option of multiple testing at different points in the life cycle in order to remove more defects at the cost-effectiveness point of testing." Perry (Ref. 15) continues to say that "if during the requirements phase you could remove 80% of the defects and then during design up to 80% of the remaining defects left from requirements plus those introduced during design, and during coding another 80% of the defects undetected during requirements and design, plus 80% of the defects introduced through coding, etc. only a minimal number of defects would remain for detection" during the running of test problems. Experience shows that the second concept is more sound economically. Therefore, Beizer (Ref. 1) concludes, "our objective must shift from an ABSOLUTE PROOF to a SUITABLY CONVINCING DEMONSTRATION... This implies a statistical measure of software RELIABILITY... Our goal, then, should be to provide sufficient testing to assure that the probability of failure due to hibernating bugs is sufficiently low to be acceptable. Sufficient implies judgement." So, how to go about testing and what is to be tested? The following sections attempt to provide an answer to this question.

5.4.1 When to Test

Testing should be performed in every phase of program development. Perry (Ref. 15) divides the life cycle of a program into six phases:

1. REQUIREMENTS: also called specifications. These define the problem. They state what is expected from the program. "Specifications must be tested for clarity and completeness," (Ref. 7). "Vague or untestable requirements will leave the validity of the delivered product in doubt. Late discovery of requirements inadequacy can be very costly," (Ref. 15). One approach for analyzing specifications is to use CAUSE-AND-EFFECT logic (Ref. 7).

2. DESIGN: this consists of two steps. First step is preliminary design where N-S diagrams contain only general information. Second step is detailed design where N-S diagrams are more detailed. In the design phase, testing is done by continuous checking: does the N-S diagram yield what is expected? Are there any steps missing? Is the logic simple and correct? etc. Testing techniques are described in section 5.4.2.

3. CODING: "Experimental" testing occurs here. Coding

introduces syntactic errors in addition to any logical errors remaining from the design phase. Top-down design and incremental testing, described by Holzer (Ref. 5), are best applied during this phase.

4. TEST PHASE: "During the test process, careful control and management of test information is critical. Test sets, test results, and test reports should be catalogued and stored," (Ref. 15). Testing techniques are described in section 5.4.2.

5. INSTALLATION: this is the phase of placing tested programs into production.

6. MAINTENANCE: "As the system is used, it is MODIFIED either to correct errors or to augment the original system... The OBJECTIVE of testing during maintenance is to provide the highest degree of assurance that the installed change will function properly and that unaffected areas of the application will not cause unanticipated problems due to a by-product of the change," (Ref.15). In this phase the importance of documented test data is mostly recognized.

Before going into the methods of testing, the following points should be emphasized:

1. It is highly recommended, if not required, that the person who is testing the program be different from the person who is developing it. Also, there should be at least two persons on the testing team.

2. Tests must reproducible (Ref. 1) "so that they can serve a diagnostic purpose if they reveal a bug. An undocumented test cannot be reproducible."

3. The actual test data is prepared BEFORE module coding begins (Ref. 7). "An unexpected test result is as often due to a TEST BUG as it is to a real bug,"(Ref. 1).

4. "The number of sets of test data has to be kept to a manageable number," (Ref. 7).

5. The results of each test should be documented for future reference.

5.4.2 Testing Techniques and Tools

How does one go about testing a computer program? "Perhaps," as reference 10 cites, "the most intuitive (and seemingly plausible) answer to this question is to consider the program as a black box and test it for all possible input cases to see if it will produce the correct output! Unfortunately, ...it is in general impractical for us to do this simply because of the number of possible input cases involved." In fact, for most programs, there is an infinite number of possible input cases (Ref. 10); one cannot try them all! There are two main approaches to test a program: structural testing and functional testing. In functional testing, tests are based on the function(s) of the program as a whole. "A complete functional test would consist of subjecting the program to all possible input streams; practically impossible for most programs," (Ref. 1). In structural testing, tests are based on the structure of the program (Ref. 1). Structural testing means that "one should design a sufficient number of tests to assure that every path through the routine is exercised at least once. (Right off that's impossible because some loops might never terminate). Total path testing is not generally practical," (Ref. 1). One cannot draw a clear distinct line between functional testing and structural testing. In fact, as re-

ference 10 mentions, the internal structure of the program should be "exploited to reduce the number of necessary cases and for all other cases (the vast majority), one tries to convince oneself by REASONING... In this respect, structured programming is of special significance in view of the special emphasis on control structures. With structured programming, the number of test cases can be reduced." Before describing in detail two testing techniques and tools, the following guidelines summarize the testing strategy of our program:

1. Perform the MINIMAL TEST on the program; that is, plan sets of test data the execution of which assures that each statement in each module has been executed at least once.

2. Plan test data sets to make sure that each BRANCH in the N-S diagrams is traversed at least once.

3. Since the program has more than one LOGICAL PATH, do the following:

    a) Identify all the possible logical paths

    b) Test each MAJOR path on its own

    c) Test the module interface at each of the decision point in isolation.

4. Do sets of tests at the "bounds" or limits. For

example, if MX=30 try NJ=30; etc. Remember to test BOTH the minimum and maximum bounds or limitations imposed by the program.

5.4.2.1 Path Testing

"Path testing based on structure is a powerful tool at the unit level. With suitable interpretation, it can be used for functional tests at the system level," (Ref. 1). A COMPLETE path testing is attained by (Ref. 1):

1. traversing every path from entry to exit
2. executing every statement at least once
3. traversing every branch at least once

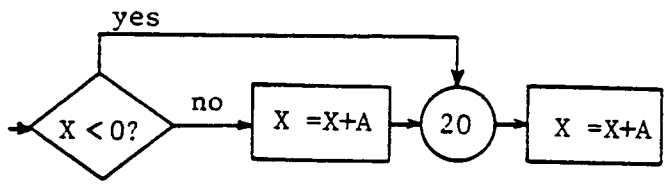
According to Beizer (Ref. 1), "if prescription 1 is followed, then prescriptions 2 and 3 are automatically followed. However, prescription 1 is impractical for most routines. It can be performed only for routines that have no loops, in which case it is equivalent to a combination of prescriptions 2 and 3." It is important to realize that prescriptions 2 and 3 are not the same, a point illustrated by the following examples given by Beizer(Ref. 1):

Figure 5.1(a) shows a "correct version of a routine. For X negative, the output is X+A, while for X greater than or equal to zero, the output is X+2A. Following prescription 2

and executing every instruction, but not every branch, would not reveal the bug" in the incorrect version shown in figure 5.1(b). Beizer continues,"A negative value produces the correct answer. Every instruction can be executed, but if the test cases do not force each branch to be taken in all directions, the bug can remain hidden."

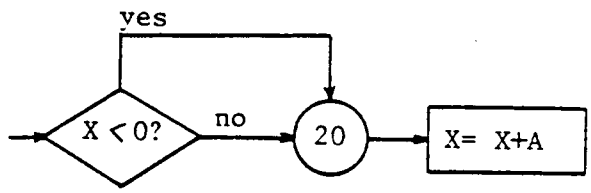
The next example is shown in figure 5.1(c). This example (Ref. 1),"uses a test based on executing each branch in all directions but does not force the execution of all instructions. The hidden loop around label 10 is not revealed by tests based on prescription 3 alone, because no test forces the execution of statement 10 and the following GOTO statement. Furthermore, label 10 is not (or will not be) flagged by the compiler as an unreferenced label, and the subsequent GOTO does not refer to an undefined label." The OBJECTIVE of path testing is to execute a sufficient number of tests to assure that both prescriptions 2 and 3 have been covered (Ref. 1).

```
   IF(X.LT.0) GO TO 20
   X=X+A
20 X=X+A
```

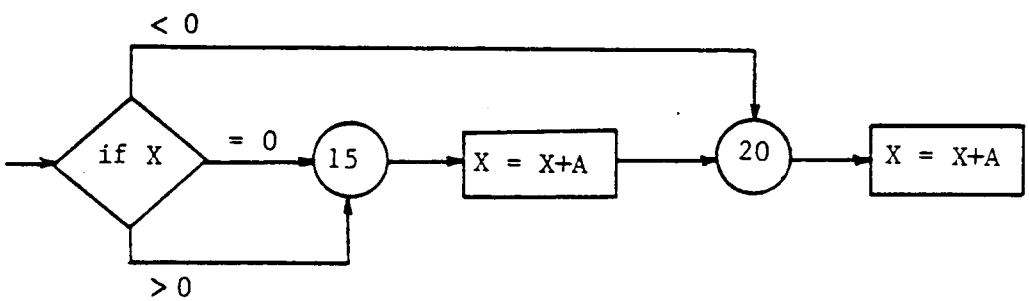


(a) Correct Version of Routine

```
   IF(X.LT.0) GO TO 20
20 X=X+A
```



(b) An Incorrect Version (tested by prescription 2 alone)




```
   IF (X) 20,15,15
10 X=X+A
   GO TO 10
15 X=X+A
20 X=X+A
```

(c) An Incorrect Version (tested by prescription 3 alone)

Figure 5.1 Example (Ref. 1)

How to select paths? Beizer (Ref. 1) answers that question,"Select paths as (small) deviations from the NORMAL paths, starting with the simplest, most familiar, most direct paths from the entry to the exit. More paths should be added to provide more coverage." For loops, extreme cases should be checked (Ref. 1):

1. no looping (bypassing the loop)
2. one path through the loop
3. two paths through the loop
4. one less than maximum
5. the maximum
6. forbidden cases

As pointed out by Beizer (Ref. 1) himself, "...all that is a lot of work...," but, "the act of careful, complete, systematic test design will catch as many bugs as the act of testing... The test design process, at all levels, is at least as effective at catching bugs as is running the test designed by that process! Bugs caught during test design cost less to fix than bugs caught during testing.."

5.4.2.2 Argument Matrix

An argument matrix is a useful testing tool used in program design phase and in subsequent phases of of program de-

velopment. The argument matrix is described by Hughes(Ref. 7) to be a tool that assists in the testing of intermodule interfaces. "In this matrix," as portrayed by (Ref. 7), "all module names appear across the top. Include all DATA ITEMS that you think belong there. Unused data items can be easily deleted later. Each data item that is passed to or from a module should be identified with either an I, an M, or a U, where

I indicates that this data item is INITIALLY provided by this module (Data is either provided by an input operation or is established internally through initialization)

M indicates that the data item could be or is MODIFIED by this module

U indicates that the data item is USED by this module."

It is suggested that an additional identification letter be used, P. P identifies items that are merely PASSED through the module from a higher-level module to a lower-level module, without being U or M. P is added to the argument matrix during the phase of program coding when items to be passed are recognized. The argument matrix is constantly modified (items added or deleted, data items original names changed to symbolic computer names, etc.) as the program development process proceeds. Mainly, there are two "edi-

tions." The preliminary argument matrix (or first edition) is constructed from the N-S diagrams developed during preliminary program design (see section 5.4.1). In this matrix, data items may be repesented by their usual literature symbols (no symbolic computer names). For example, the modulus of elasticity is represented by E, the moment of inertia by I, the area by A, etc. One of the advantages of this is that it tells how many times a data item is used, and if a group of data items is always needed as a group and not as individual data items. This may save some symbolic (computer) names and arrays. For example, instead of storing E, A, and L in arrays EMOD(), AREA(), and ELENG(), it might be concluded that only the quantity EA/L need be stored; one variable can be used for that purpose. Thus, two variable names are saved! The second edition of the argument matrix is constructed during the detailed design phase (see section 5.4.1). Here, symbolic (computer) names are chosen for each data item. For example, the modulus of elasticity E may be represented by EMOD, I by ZI, A by AREA, etc.

What is the main purpose of the argument matrix? The answer is found in reference 7 : "By examining these entries in the matrix, you may discover omissions or inconsistences. For example, if the initialization of a data item has been overlooked, this omission should be detected here. If a

data item is passed to a module but never used in that module," and it is not a P, "the inconsistency should be noticed. This makes it very helpful in planning module testing and stub requirements." In addition to that, the argument matrix, along with the argument flow diagram discussed in section 5.4.2.3), can give the programmer a feeling of confidence in his work. This is very important, especially in large programs where the programmer might feel that he lost control! At a glance, the argument matrix can tell, for example, the "life story" of any data item. A look at module entries can show all data items processed by the module. The argument matrices for the PROCESSOR and the POSTPROCESSOR are shown in figures 5.2 and 5.3.

5.4.2.3 Argument Flow Diagram

The argument flow diagram is nothing but the tree chart (see chapter 4) to which variables (or arguments) passed to and received from modules are augmented. The arguments are written,in an alphabetic order, on a line that links two modules. Arguments can be added and deleted as the process of program development dictates. The argument flow diagram is constructed just after the second edition of the argument matrix (see section 5.4.2.2) is written, and during top-down code-testing.

Module arguments can be classified into input and output arguments. An input argument is a variable passed from a higher-level module to the lower-level module under concern. An output argument is a variable passed from a lower-level module (under concern) to a higher-level module. A variable can be, at the same time, an input and output argument if it is received by the module, modified, and then passed back to the higher-level module. It should be remembered that the argument list of a module is used for module interfacing. Thus, a variable is classified only if it is part of the interface. An item produced by a module and not passed up to a higher-level module is NOT an output argument!

The main benefit from drawing an argument flow diagram is that, at a glance, errors in module interfacing can be readily detected. Modifications to the diagram (deleting, adding, etc.) can be easily made. As an example, the argument flow diagram for the POSTPROCESSOR is shown in figure 5.4 .

	PROCES	DATA	STRUCT	CODES	MBAND	LOAD	JLOAD	MLOAD	FEFORC	ASSEMF	SYSTEM	STIFF	ELEM1	ELEM2	ELEM3	ELEM4	ASSEMS	MODIFY	JDISP	JSPRG	SOLVE	SPBFA	SPBSL
AREA													I-U	I-U	I-U	I-U							
C1( ) &C2		I								U			U	U	U	U							
COEF																				I-U			
D1 & D2								I-U															
DISP																			I-U				
ELENG( )		I						U	U				U	U	U	U							
EMOD													I-U	I-U	I-U	I-U							
F(,)						I		U-M	U-M	U													
F2,F3 F5, & F6								I	U														
FORCE							I-U																
G( )													I	I	I	I	U						
INDEX(,)													I	I	I	I	U						
JCODE(,)			I	U			U												U	U			
JDIR							I-U												I-U	I-U			
JNUM							I-U												I-U	I-U			
LC	I	U				U	U	U			U										U		
LT								I-U															
MBD			I									U					U		U	U		U	U
MCODE(,)				I	U					U							U						
MN								I-U	U														
MTYP( )		I							U			U											
MXNEQ	I											U					U		U	U		U	U
N												I	U	U	U	U	U						
NA( )						I		U-M		U													
NE	I	U		U	U	U				U		U											
NEQ		I				U						U							U			U	U
NJ	I		U																				
NJD																		I-U	U				
NJL						I-U	U																
NJS																		I-U		U			
NLC	I-U																						
NML						I-U		U															
P1 & P2								I-U															
PFAC																			I-U				
PNUM																			I-U-M				
Q( )						I	M			U-M									M				U-M
SS(,)												I					U-M		U-M	U-M		U	U
ZI													I-U	I-U	I-U								

Figure 5.2 Argument Matrix for PROCESSOR

	POST	LCOND	ELEMF	JOINTF	SECE	SECL	LT1	LT2	LT3	LT4	LT5	LCOMB	MAXDEF	MAXSTR
AF(,)					I					U-M	U-M	U		
AFMAX														I-U-M
AREA( )	I		U											U
BM(,)					I		U-M	U-M	U-M			U		
BMAX														I-U-M
C1( )	I		U	U										
C2( )	I		U	U										
COEF( )												I-U		
D( )			I-U-M		U-M									
D1						I	U		U	U				
D2						I			U					
DESAF( )														I-M
DESBM( )														I-U-M
DSEC(,)	I				U		U	U	U	U	U			
ELENG( )	I		U		U									
EMOD( )	I		U		U		U	U	U					
F(,)	I		U-M	U	U							U		
FS(,)												I-U-M		
I		I	U	U	U									
ILC( )														I-M
IR		I-U-M												
JCODE(,)	I												U	
JXDES													I-M	
JXMAX													I-U-M	
JYDES													I-M	
JYMAX													I-U-M	
LC	I					U						I		
LCI	I	U	U	U	U	U						U		
LCK	I	U	U	U	U	U						U		
LCM												I	U	U
LDES( )														I-M
LMAX														I-M-U
LT						I-U								
MCODE(,)	I		U											
MINC(,)	I			U										
MN						I-U	U	U	U	U	U			
MTYP( )	I		U		U									

Figure 5.3 Argument Matrix for POSTPROCESSOR(continued next page)

MX	I-U													
NCOMB	I-U											U		
NE	I-U	U										U		U
NEQ	I-U											U		
NJ	I											U	U	
NLC	I-U											U		
NML						I-U								
NSEC( )	I	U			U	U	U	U	U	U	U	U		U
P(,)	I			U-M								U		
P1						I	U	U	U	U	U			
P2						I			U					
PS(,)												I-U-M		
Q( )	I		U									U		
QS( )												I-U-M	U	
SEC(,)												I-U-M		U
SF(,)					I		U-M	U-M	U-M			U		
TD(,)					I		U-M	U-M	U-M			U		
XDES													I-U-M	
XMAX													I-U-M	
YDES													I-U-M	
YMAX													I-U-M	
ZI( )	I		U		U		U	U	U					
ZS( )	I													U

... Continuation of Figure 5.3

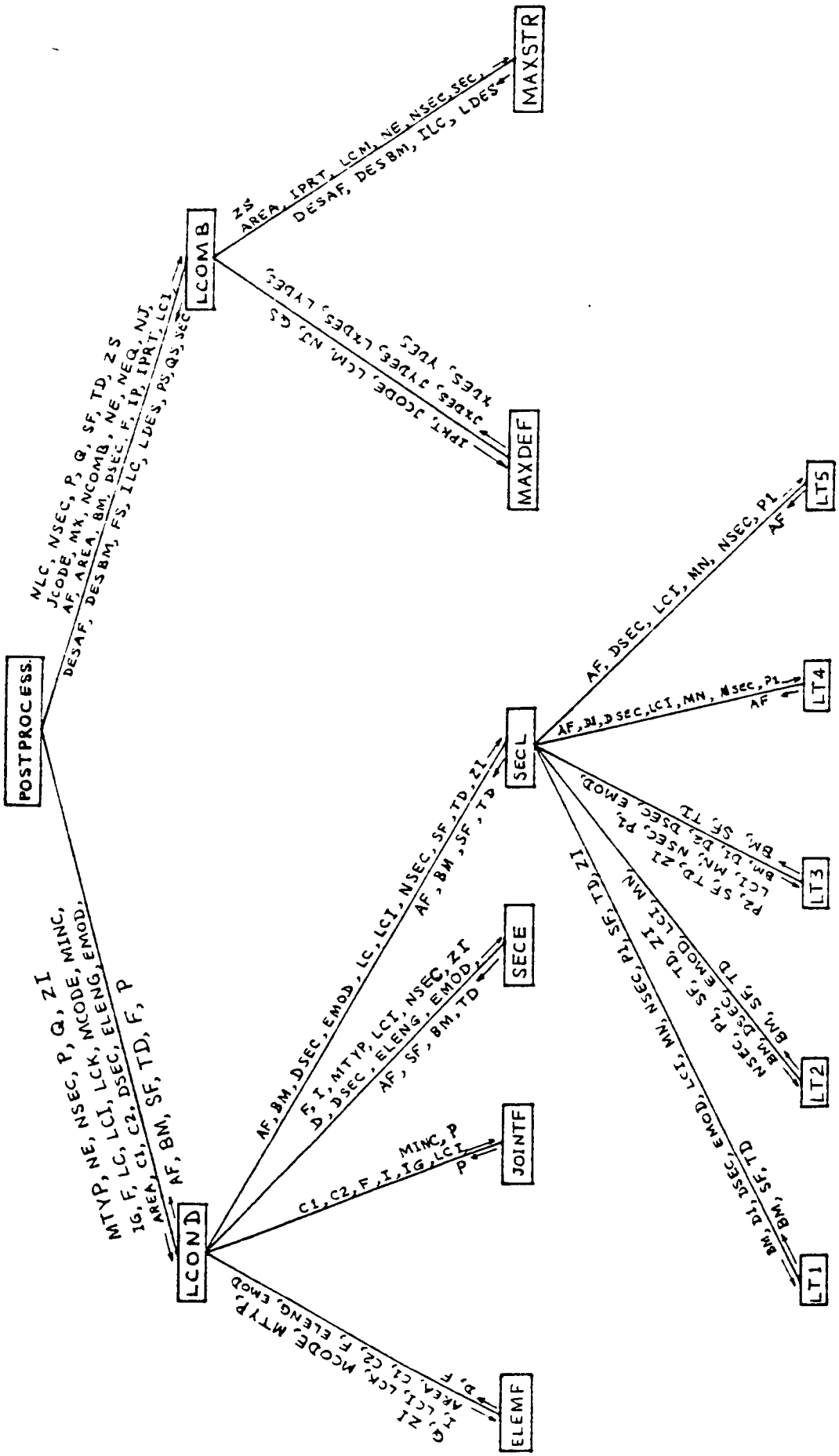


Figure 5.4 Argument Flow Diagram For the Postprocessor

REFERENCES

1. Beizer, B., Software Testing Techniques Van Nostrand Reinhold Company, 1983.

2. Bathe, K. J., Finite Element Procedures in Engineering Analysis Prentice-Hall, Englewood Cliffs, N. J., 1982.

3. Huber, H., An Interactive Preprocessor for a Plane Frame Analysis Program on the IBM PC, Thesis, Virginia Tech, December 1984.

4. Hariri, M. R., Post Processor for Plane Frame Analysis Project and Report, Virginia Tech, March 1984.

5. Holzer, S. M., Computer Analysis of Structures Elsevier, to appear in 1985.

6. Hughes, C. E., Pfleeger, C. P., and Rose, L. L., Advanced Programming Techniques A Second Course in Programming Using Fortran, John Wiley & Sons, 1978.

7. Hughes, J. K., and J. I. Michtom, A Structured Approach to Programming Prentice-Hall, Englewood Cliffs, N. J., 1977.

8. ICES STRUDL-II, The Structural Design Language, Engineering User's Manual Vol. I, Frame Analysis, 1st edition, Department of Civil Engineering, M.I.T., 1968.

9. Infotech, Software Testing, State of the Art Report Infotech International, England, 1979.

10. Infotech, Structured Programming, State of the Art Report Infotech International, England, 1976.

11. Merchant, M. J., FORTRAN 77: Language and Style Wadsworth Publishing Company, Belmont, California, 1981. 12. Moore, J. B., WATFIV: Fortran Programming with the WATFIV Compiler Reston Publishing Company, Reston, Virginia, 1978.

13. Nassi, I., and B. Schneiderman, "Flow Chart Techniques for Structured Programming," ACM SIGPLAN Notices Vol 8, No. 8, August 1973.

14. Norris, C. H., Wilbur, J. B., and Utku, S., Elementary Structural Analysis 3rd edition, McGraw-Hill, 1976.

15. Perry, W. E., A Structured Approach to Systems Testing Prentice-Hall, Englewood Cliffs, N. J., 1983.

Appendix A

DRAWING N-S DIAGRAMS USING SCRIPT

SCRIPT provides a tool to draw the N-S diagrams. This tool is the BOX control word. The BOX control word (REF: Document Composition Facility- User's Guide, IBM) has the following form:

```
.BX option1 option2
```

Option1 can be: NEW, SET, OFF, or CAN.

Option2 can be: d1, d2, ..., dn; also a slash (/) separating these serve as a specifier (see below).

The following describes each one of these options:

d1, ..., dn: denote the positions from left margin at which vertical lines are placed. For vertical lines to be printed continuously, the printer option of eight lines per inch, 8 lpi, should be used. Otherwise, if 6 lpi is used, the vertical lines will be dashed. At the same time, PL (page length) should be set to 88.

/ (slash): If a slash separates di and dj, then column positions di and dj will not be connected by a horizontal line. This enables the drawing of adjacent separate boxes.

NEW: allows boxes to be drawn inside boxes. It initiates, or nests, a new box inside the current box.

SET: is just like NEW option except that no initial, or top, horizontal line connecting the column positions is drawn.

OFF: draws the final, or bottom, horizontal line of current box; that is, it causes current box to be off.

CAN: causes the current box to end, or be cancelled; it is just like OFF except that no bottom horizontal line is drawn. The result is a box without a box bottom.

.BX alone will draw horizontal line joining the positions of the columns of the current box.

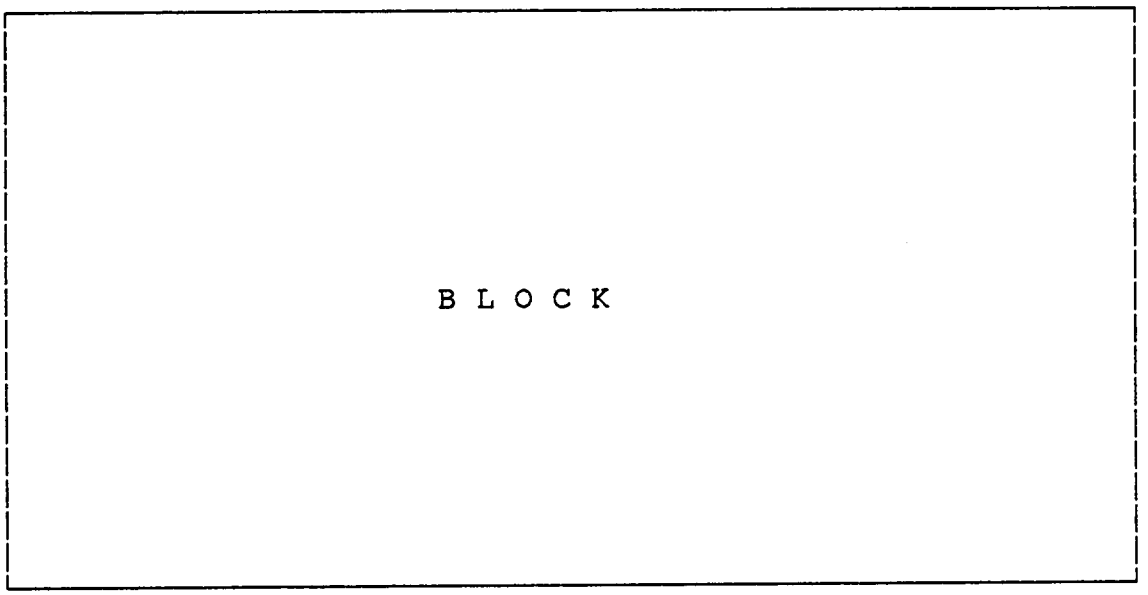
Application to N-S diagrams:

Example 1:

The SEQUENCE structure can be drawn by the following:

```
.BX 1 60
.SP 4
                    B L O C K
.SP 4
.BX OFF
```

The result is:



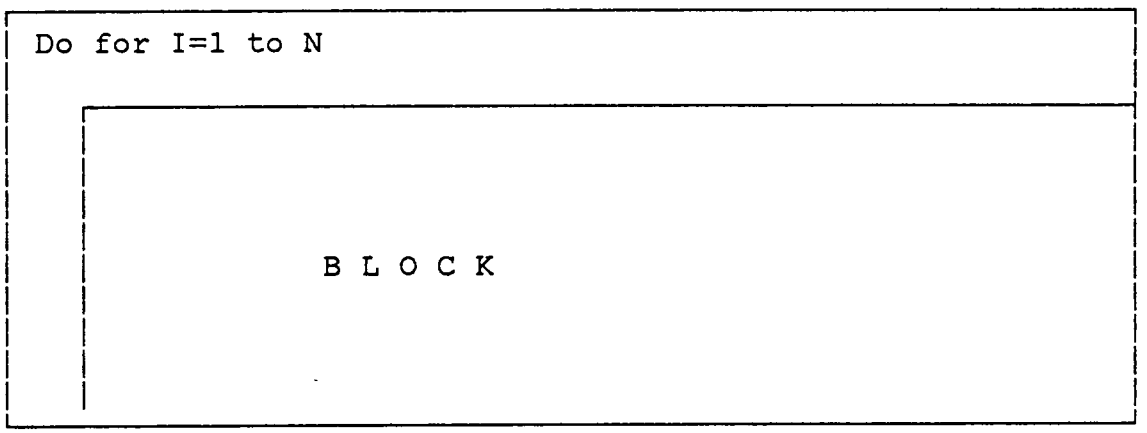
Example 2:

The LOOP structure can be drawn by the following:

```
.BX 1 60
  Do for I=1 to N
.BX NEW 5 60
.SP 3
```

```
B L O C K
.SP 3
.BX CAN
.BX OFF
```

The result is:



Example 3:

The following N-S diagram for the IF-THEN-ELSE structure is suggested as an alternative to the standard N-S diagram.

The IF-THEN-ELSE structure can be drawn by the following:

```
.BX 1 60
  IF condition
.BX SET 1 40 60
    then                                    else
.BX
```

```
.SP 4
   B  L  O  C  K        1                    B L O C K      2
.sp 4
.BX CAN
.BX OFF
```

The result is:

IF condition then	else
B L O C K 1	B L O C K 2

Example 4:

The CASE structure can be drawn by the following:

```
.BX 1 60
```

```
  Select case
.BX SET 1 20 40 60
        1                   2                   3
.BX
.SP 1
   B L O C K  1      B L O C K  2      B L O C K  3
.SP 1
.BX CAN
.BX OFF
```

The result is:

Select case		
1	2	3
B L O C K 1	B L O C K. 2	B L O C K 3

APPENDIX B

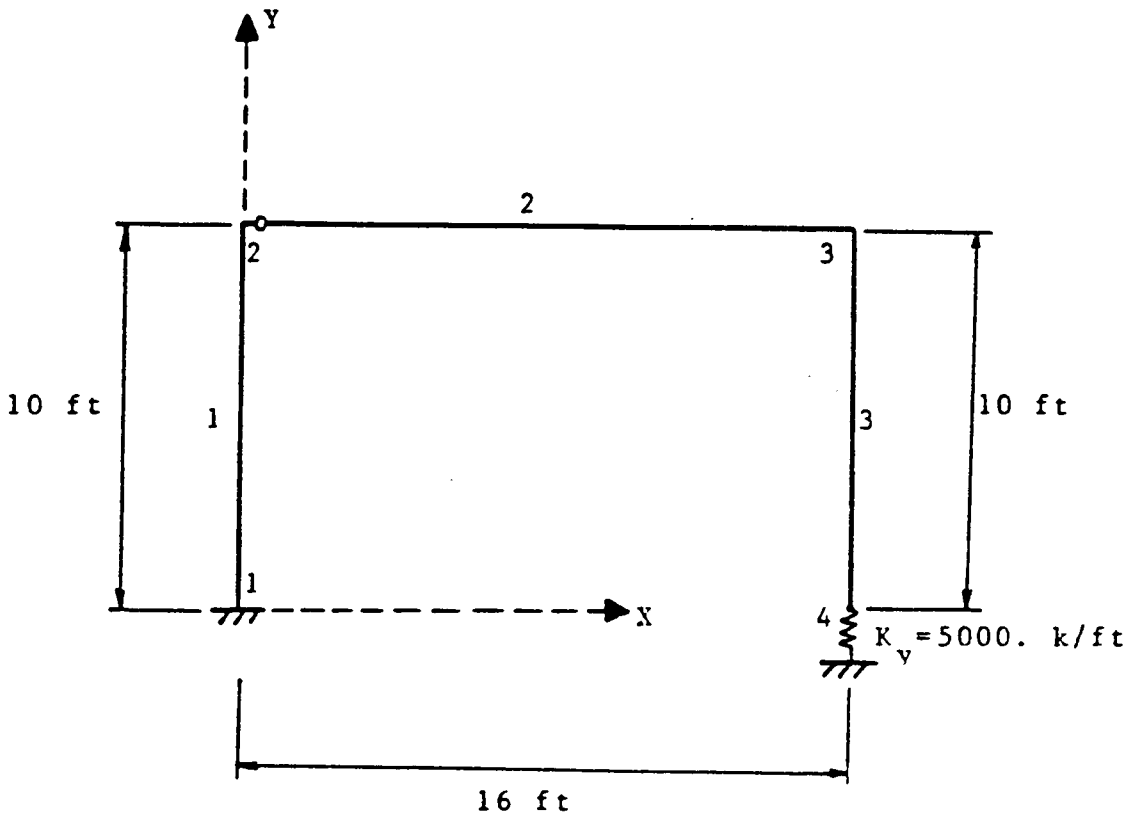
DEMONSTRATIVE EXAMPLE

The structure shown in figure B.1 is subjected to two loading conditions. The linear structure consists of prismatic elements and is subjected to the loading conditions shown in figure B.2 . At the a-end (joint 2) of element 2 there is a hinge. At joint 4 there is a linear spring with stiffness of 5000 k/ft.

The dead load of the structure is not included in the analysis. Each element of the structure is divided so that five equally spaced sections are obtained. There are two load combinations: The load factors for load combination 1 are 2.67 and 1.70 . The load factors for load combination 2 are 0.00 and 0.90 .

REQUIRED:

1) Joint displacements, element-end forces, joint forces, and section responses for each loading condition.
2) Joint displacements, element-end forces, joint forces, section responses, maximum joint deflections, and maximum moments and stresses for each load combination.
3) Design values of joint deflections, and design moments and stresses.



For all elements:

Area = 75. in^2

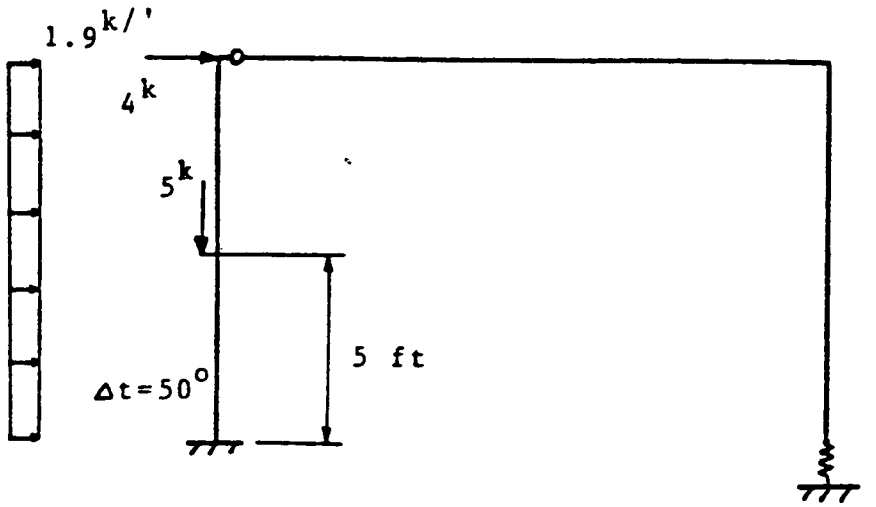
I = 1500. in^4

S = 200. in^3

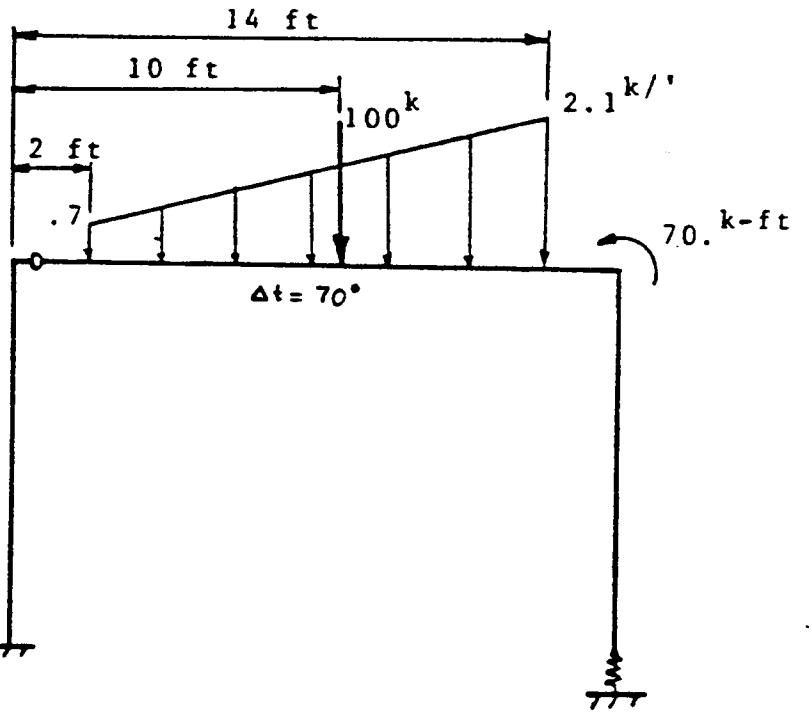
E = 30000. ksi

Coeff. of thermal expansion = 0.0000065

Figure B.1



LOADING CONDITION 1



LOADING CONDITION 2

Figure B.2 Loading Conditions

LOADING CONDITION 1 :

JOINT DISPLACEMENTS (GLOBAL):

JOINT	X DISP.	Y DISP.	Z ROTATION
1	.0000000	.0000000	.0000000
2	.0541633	.0389268	-.0005504
3	.0535751	-.0027681	-.0005253
4	.0000000	-.0027079	.0000000

ELEMENT END FORCES (LOCAL):

ELEMENT	AT END	Fx (axial)	Fy (shear)	Fz (moment z)
1	A	3.872	16.107	792.781
1	B	1.128	2.893	.000
2	A	6.894	-1.128	.000
2	B	-6.894	1.128	-216.636
3	A	1.128	6.893	216.635
3	B	-1.128	-6.893	610.584

JOINT FORCES (GLOBAL):

JOINT	X FORCE	Y FORCE	Z MOMENT
1	-16.107	3.872	792.781
2	4.000	.000	.000
3	.000	.000	.000
4	-6.893	1.128	610.584

SECTION RESPONSES (ALL ARE LOCAL):

ELEMENT 1 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-3.872	16.107	-792.781	.0000000
30.00000	-3.872	11.357	-383.83[illegible]	-.0064[illegible]
60.00000	-3.872	6.607	-111.391	-.02072[illegible]
90.00000	1.128	1.857	15.555	-.0274815
120.00000	1.128	-2.893	.000	-.0541634

ELEMENT 2 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-6.894	-1.128	.000	.0389268
48.00000	-6.894	-1.128	-54.159	.0354754
96.00000	-6.894	-1.128	-108.318	.0291711
144.00000	-6.894	-1.128	-162.477	.0177605
192.00000	-6.894	-1.128	-216.636	-.0027681

ELEMENT 3 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-1.128	6.893	-216.635	.0535751
30.00000	-1.128	6.893	-9.830	.0363401
60.00000	-1.128	6.893	196.974	.0189086
90.00000	-1.128	6.893	403.779	.0054165
120.00000	-1.128	6.893	610.584	.0000000

LOADING CONDITION 2 :

JOINT DISPLACEMENTS (GLOBAL):

JOINT	X DISP.	Y DISP.	Z ROTATION
1	.0000000	.0000000	.0000000
2	-.2153985	-.0020266	.0026925
3	-.1294745	-.1933257	.0030554
4	.0000000	-.1891230	.0000000

ELEMENT END FORCES (LOCAL):

ELEMENT	AT END	Fx (axial)	Fy (shear)	Fz (moment z)
1	A	37.999	-16.828	-2019.361
1	B	-37.999	16.828	.000
2	A	16.828	37.999	.000
2	B	-16.828	78.801	-1315.457
3	A	78.801	16.828	2155.456
3	B	-78.801	-16.828	-136.095

JOINT FORCES (GLOBAL):

JOINT	X FORCE	Y FORCE	Z MOMENT
1	16.828	37.999	-2019.361
2	.000	.000	.000
3	.000	.001	839.999
4	-16.828	78.801	-136.095

SECTION RESPONSES (ALL ARE LOCAL):

ELEMENT 1 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-37.999	-16.828	2019.361	.0000000
30.00000	-37.999	-16.828	1514.521	.0185[illegible]
60.00000	-37.999	-16.828	1009.68[illegible]	[illegible]
90.00000	-37.999	-16.828	504.84[illegible]	[illegible]
120.00000	-37.999	-16.828	.000	[illegible]

ELEMENT 2 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-16.828	37.999	.000	-.[illegible]
48.00000	-16.828	36.365	1815.269	-.[illegible]
96.00000	-16.828	31.655	3446.271	-.[illegible]
144.00000	-16.828	-74.833	2415.477	-.[illegible]
192.00000	-16.828	-78.801	-1315.455	-.[illegible]

ELEMENT 3 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-78.801	16.828	-2155.456	-.1294745
30.00000	-78.801	16.828	-1650.616	-.0576842
60.00000	-78.801	16.828	-1145.776	-.0189062
90.00000	-78.801	16.828	-640.936	-.0030438
120.00000	-78.801	16.828	-136.095	.0000000

****

LOAD COMBINATION 1 :

JOINT DISPLACEMENTS (GLOBAL):

JOINT	X DISP.	Y DISP.	Z ROTATION
1	.0000000	.0000000	.0000000
2	-.2215613	.1004895	.0021077
3	-.0770612	-.3360446	.0037917
4	.0000000	-.3287393	.0000000

ELEMENT END FORCES (LOCAL):

ELEMENT	AT END	Fx (axial)	Fy (shear)	Fz (moment z)
1	A	74.935	14.397	-1316.188
1	B	-61.585	36.333	-.001
2	A	47.014	61.585	.000
2	B	-47.014	136.975	-2814.695
3	A	136.973	47.013	4242.691
3	B	-136.973	-47.013	1298.897

JOINT FORCES (GLOBAL):

JOINT	X FORCE	Y FORCE	Z MOMENT
1	-14.397	74.935	-1316.188
2	10.680	.000	-.001
3	.000	.001	1427.997
4	-47.013	136.973	1298.897

SECTION RESPONSES (ALL ARE LOCAL):

ELEMENT 1 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-74.935	14.397	1316.188	.0000000
30.00000	-74.935	1.714	1557.853	.0142845
60.00000	-74.935	-10.968	1415.044	.0569819
90.00000	-61.585	-22.651	895.759	.1216461
120.00000	-61.585	-36.333	-.001	.2215613

ELEMENT 2 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION

.00000	-47.014	61.585	.000	.1004895
48.00000	-47.014	58.808	2924.353	-.3015075
96.00000	-47.014	50.675	5569.453	-.5550489
144.00000	-47.014	-130.232	3677.592	-.5340881
192.00000	-47.014	-136.975	-2814.695	-.3360448

ELEMENT 3 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	Y DEFLECTION
.00000	-136.973	47.013	-4242.691	-.0770612
30.00000	-136.973	47.013	-2832.294	-.0010350
60.00000	-136.973	47.013	-1421.897	.0183453
90.00000	-136.973	47.013	-11.500	.0092877
120.00000	-136.973	47.013	1398.897	.0000000

MAXIMUM JOINT DEFLECTIONS:

X DISP.	ITS JOINT	Y DISP.	ITS JOINT
-.2215613	2	-.3360446	3

MAXIMUM MOMENTS & STRESSES:

ELEMENT	Fz (moment z)	STRESS	SECTION NO.
1	1557.853	8.788	2
2	5569.453	28.474	3
3	-4242.691	23.040	1

LOAD COMBINATION 2 :

JOINT DISPLACEMENTS (GLOBAL):

JOINT	X DISP.	Y DISP.	Z ROTATION
1	.0000000	.0000000	.0000000
2	-.1938587	-.0018239	.0024232
3	-.1165271	-.1739931	.0027495
4	.0000000	-.1702107	.0000000

ELEMENT END FORCES (LOCAL):

ELEMENT	AT END	Fx (axial)	Fy (shear)	Fz (moment z)
1	A	34.199	-15.145	-1817.425
1	B	-34.199	15.145	.000
2	A	15.145	34.199	.000
2	B	-15.145	70.921	-1180.912
3	A	70.920	15.145	1936.91[illegible]
3	B	-70.920	-15.145	-122.48[illegible]

JOINT FORCES (GLOBAL):

JOINT	X FORCE	Y FORCE	Z MOMENT
1	15.145	34.199	-1817.425
2	.000	.000	.000

3	.000	.001	755.999
4	-15.145	70.920	-122.486

SECTION RESPONSES (ALL ARE LOCAL):

ELEMENT 1 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-34.199	-15.145	1817.425	.0000000
30.00000	-34.199	-15.145	1363.069	.0166597
60.00000	-34.199	-15.145	908.712	.0605808
90.00000	-34.199	-15.145	454.356	.1226702
120.00000	-34.199	-15.145	.000	.1938587

ELEMENT 2 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-15.145	34.199	.000	-.0018279
48.00000	-15.145	22.729	1624.742	-.2097106
96.00000	-15.145	28.529	3101.644	-.2350876
144.00000	-15.145	-67.351	2176.626	-.3072927
192.00000	-15.145	-70.921	-1182.912	-.1739931

ELEMENT 3 :

SEC. AT DIST.	Fx (axial)	Fy (shear)	Fz (mom. z)	y DEFLECTION
.00000	-70.920	15.145	-1939.910	-.1165271
30.00000	-70.920	15.145	-1485.554	-.0519158
60.00000	-70.920	15.145	-1031.198	-.0170156
90.00000	-70.920	15.145	-576.842	-.0027394
120.00000	-70.920	15.145	-122.486	.0000000

MAXIMUM JOINT DEFLECTIONS:

X DISP.	ITS JOINT	Y DISP.	ITS JOINT
-.1938587	2	-.1739931	3

MAXIMUM MOMENTS & STRESSES:

ELEMENT	Fz (moment z)	STRESS	SECTION NO.
1	1817.425	9.543	1
2	3101.644	15.710	2
3	-1939.910	10.645	1

DESIGN VALUES:

JOINT DISPLACEMENT:

X-DISP. = -.2215617 AT JOINT 2 LOAD COMBINATION 1

Y-DISP. = -.2364446 AT JOINT 2 DUE TO LOAD COMBI 1

DESIGN MOMENTS AND STRESSES:

ELEMENT STRESS Fz (mom. z AT SECTION LOAD COMBINATION

1	9.543	1817.425	1	2
2	28.474	5569.453	3	1
3	23.040	-4242.691	1	1

The vita has been removed from the scanned document