Report no.
UC SESM 77-2

STRUCTURAL ENGINEERING AND STRUCTURAL MECHANICS

CAL

COMPUTER ANALYSIS LANGUAGE FOR THE STATIC AND DYNAMIC ANALYSIS OF STRUCTURAL SYSTEMS

by

E.L.WILSON

JANUARY 1977

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

C A L

Computer Analysis Language for the Static and Dynamic Analysis of Structural Systems

by

Edward L. Wilson

University of California
Department of Civil Engineering
Division of Structural Engineering
and Structural Mechanics
Berkeley, California
Report No. UC SESM 77-2

TABLE OF CONTENTS

I. INTRODUCTION

The static and dynamic analysis of structures can be completely automated. During the past ten years several general purpose computer programs have been developed which automatically evaluate the displacements and member forces within the structure. In order to use such a program it is only necessary to idealize the structure by a system of joints (nodal points) interconnected by structural members (elements). The computer input is a simple numerical listing of the geometry of the joints, properties of the members and the location and magnitude of the loads. Therefore, it is now possible for professional engineers to perform complex structural analyses without a complete knowledge of the basic assumptions and theory which have been incorporated into the computer program. This "black box" use of computer programs for structural analysis has caused problems in the interpretation of the results of the analysis. Because of these problems many professional engineers do not have faith in the use of general purpose programs and still use simplified approximate methods which can be checked by hand calculations.

In many Universities the teaching of structural analysis methods to both undergraduate and graduate students does not properly emphasize the direct stiffness method which is the most common computer method implemented. One of the reasons for this practice is because most modern books on matrix analysis of structures do not contain a simple explanation of the direct stiffness method. Also, it is very cumbersome to teach using hand calculation techniques.

In the teaching of structural dynamics, strong background in static analysis of complex structures is not normally required. Therefore,

dynamic structural analysis is often taught using very simple two-dimensional models. Hence, dynamic courses almost never illustrate the dynamic behavior of a significant three-dimensional system.

As a result of the present limitation of our education system, which still emphasize hand methods of structural analysis, professional engineers have not been able to use effectively the completely automated general purpose computer programs for structural analysis. The purpose of this report and the computer program CAL is an attempt to bridge the educational gap between the traditional hand calculation methods and the use of a completely automated program for structural analysis.

The computer program CAL is a significant extension of the program SMIS, Symoblic Matrix Interpretive System, which was developed in 1963. SMIS has been very useful in the teaching of structural analysis. The program was designed to perform the formal matrix transformation techniques of the force and displacement methods for structural analysis. The initial version of the program did not have direct stiffness options or the automated formulation of element stiffness matrices. Therefore, the use of SMIS did not illustrate to the student the methods used in automated computer programs for structural analysis. Also, the dynamic analysis operations were very limited.

CAL is a completely new program which is based on a different internal organization structure. The program is designed to work on several different types of computer including the small 16 bit minicomputer. The program has options which allow the user to debug data without printing previously obtained results. Also, looping operations provide a compact language for the evaluation of numerical algorithms which involve iteration.

In the use of CAL it should be remembered that the basic purpose of the program is educational. The automated structural analysis methods which are used in large general purpose programs have been subdivided into a series of simple operations which are under the control of the user. Also, no attempt has been made to extend the capacity of the program to effectively use low speed storage. In addition, all matrices are stored in rectangular matrix form including zero terms, without taking advantage of symmetry. Since CAL is designed for small problems, these limitations are not serious. For practical problems over 100 degrees of freedom one of the "black box" general purpose programs should be used.

The operation for general matrix inversion has been omitted intentionally from the set of CAL operations. The educational reason for this is that it is a numerically inefficient operation which is rarely required in structural analysis; therefore the student should learn to use the more efficient solve operation. Also, I hope to reverse the trend set in many modern text books which imply that matrix inversion is a necessary operation for the analysis of structural systems when using matrix notation.

II. COMPUTER ANALYSIS LANGUAGE - CAL

A. FORM AND RESTRICTION OF THE LANGUAGE.

CAL is an interpretive language which is designed to manipulate arrays and matrices and to perform several standard structural analysis operations. A CAL program run involves the reading of the input deck once and executing the commands designated by the operation cards as they are encountered. Looping operations allow a sequence of commands to be executed more than once.

The input deck is composed of operation cards and data cards. The data cards directly follow each operation card which requires data (see LOOP operation for exception to this). The operation card contains the name of the operation to be executed, names of arrays associated with the operation and integer constants. Examples of the general form of this card are

```
OP,M1,M2,M3,M4,M5,N1,N2,N3,N4
OP,M1,N1,N2
OP,N1
OP
```

in which OP is the name of the operation to be executed, Mi is the name of an array and Ni is an interger. The names of OP or Mi are one to six alphabetic characters to be selected by the user. These sequence of terms OP, Mi and Ni must be separated by commas and terminated by a blank.

If an operation attempts to load or generate an array which previously existed the program will delete the array before the execution of the operation. A new array need not be the same size of the old array which had the same name.

B. GENERAL MATRIX OPERATIONS

CAL has most of the standard matrix operations plus some special array operations which are useful in structural analysis. The theoretical background for matrix notation is given in Reference [2]. The following is a list of approximately 25 operations which are used for control and general matrix or array manipulation.

START

This operation eliminates all arrays which were previously loaded or generated

---

STOP

This operation causes normal termination of a CAL program

+
LOAD,M1,N1,N2,N3

This operation will load an array of real numbers named M1 which has N1 rows and N2 columns. The terms of the array are punched in row-wise sequence on data cards following this operation. If N3 is zero or blank the cards are punched in a format of (8F10.0).

If N3 is nonzero an additional card containing the format of the data cards must follow this operation and preceed the data cards. If the data is to be 4 numbers per card in field widths of 15 the additional card would contain the following information: (4F15.0).

+
ZERO,M1,N1,N2,N3,N4

A real matrix named M1 with N1 rows and N2 columns will be generated. The terms in this matrix will have the following values.

$$M1(I,I) = N3 \qquad I = 1,\ldots.N1$$
$$M1(I,J) = N4 \qquad J = 1,\ldots.N2$$

Therefore, this operation can be used to form null or unit matrices.

+ indicates the formation of a new matrix - a matrix previously defined with the same name will be deleted.
- indicates modification of an existing matrix.

NO
YES

These operations are used to selectively suppress output from CAL. The NO operation suppresses all printing, except diagnostics, until the operation YES is encountered. Therefore, in subsequent runs of the same CAL program, output which was previously correct need not be reprinted if these cards are inserted in the data deck.

PRINT,M1,N1

This operation will print the real array named M1 in a matrix format of 8 columns per line. If N1 is greater than zero the operation will read and print N1 comment cards which follow the operation card.

LABEL,N1

This operation will read and print N1 comment cards which follow this operation card. Column 1 of each card will be interpreted as a standing carrage control symbol (i.e. 0 for double space and 1 for a skip to the top of the next page).

DUP,M1,M2

This operation will form an array named M2 which is identical to the array named M1.

ADD,M1,M2

This operation will replace matrix M1 with the sum of the matrices M1 and M2.

SUB,M1,M2

This operation will replace matrix M1 with matrix M1 less matrix M2.

MULT,M1,M2,M3

This operation generates a new matrix M3 which is the product of matrices M1 and M2, or M3 = M1 · M2.

TRAN,M1,$\overset{+}{M2}$

This operation generates a new matrix M2 which is the transpose of matrix M1.

SCALE,$\overline{M1}$,M2

This operation replaces each term in the matrix named M1 with the term multiplied by the term M2(1,1) of matrix named M2.

SOLVE,$\overline{M1}$,$\overline{M2}$ OR SOLVE,$\overline{M1}$,N1 OR SOLVE,$\overline{M1}$,$\overline{M2}$,N1

If N1 = 0, this operation solves the set of equations $\underline{A}\ \underline{X} = \underline{B}$. In which M1 is the name of the symmetric $\underline{A}$ matrix and M2 is the name of the $\underline{B}$ matrix. Matrix $\underline{A}$ is triangularized and the results $\underline{X}$ are stored where the matrix $\underline{B}$ was initially stored and is named M2.

If N1 = 1 Matrix $\underline{A}$ is triangularized only.
N1 = 2 For a given $\underline{B}$ matrix and the $\underline{A}$ matrix previously triangularized the $\underline{B}$ matrix is replaced by the results $\underline{X}$.
N1 = 3 Matrix $\underline{A}$ is replaced by it inverse $\underline{A}^{-1}$.
For N = 0 or 1 matrix A is factored into the LDL^T form. The diagonal $\underline{D}$ matrix is stored on the diagonal of $\underline{A}$.

DUPSM,M1,$\overset{+}{M2}$,N1,N2,N3,N4

This operation forms a new submatrix named M2 with N3 rows and N4 colunns from terms within the matrix named M1. The first term of matrix M2, M2(1,1), will be from row N1 and column N2 of matrix M1, or M1(N1,N2).

STOSM,M1,M2,N1,N2

This operation stores a submatrix named M2 within the matrix named M1. The first term of the submatrix M2 will be stored at row N1 and column N2 of matrix M1. The terms within the area of M1 in which M2 is stored will be destroyed.

DUPDG,M1,$\overset{+}{M2}$

This operation forms a new row matrix named M2 from the diagonal terms of matrix M1.

STODG,$\overline{M1}$,M2

This operation stores a row or column matrix named M2 at the diagonal locations of matrix M1.

MAX,M1,+M2

This operation forms a column matrix named M2 in which each row contains the maximum absolute value of the corresponding row in matrix M1. The maximum and its column number is printed for each row.

NORM,M1,+M2,N1

If N1 = 0 a row matrix named M2 is formed in which each column contains the sum of the absolute values of the corresponding column of matrix M1.

If. N1 ≠ 0 a row matrix named M2 is formed in which each column contains the square root of the sum of the squares of the values of the corresponding column of matrix M1.

INVEL,-M1

This operation replaces each term in the matrix named M1 with it's inverse.

SQREL,-M1

This operation replaces each term in the matrix named M1 with the square root of the term.

LOG,-M1

This operation replaces each term in the matrix named M1 with the natural log of the term.

PROD,M1,+M2

This operation forms a 1 x 2 array named M2 which contains the product of all terms in the matrix M1. The product, X, is stored as two numbers of the form

$$X = P \cdot 10^{E}$$

in which M2(1) = P and M2(2) = E.

DELETE,-M1

This operation will cause the elimination from storage of the array named M1.

C. STRUCTURAL ANALYSIS OPERATIONS

The purpose of this series of operations is to form the total stiffness and diagonal mass matrices for systems of three-dimensional beam and truss elements. If the appropriate properties and boundary conditions are given two-dimensional truss and beam systems may also be treated.

After the creation of an array containing the coordinates of the joints of the system, the specification of displacement boundary conditions, the tabulation of material and section properties, the mass and stiffness matrices are formed for each structural member and placed in sequence on low speed storage along with the global equation number which are associated with their stiffness terms. In addition the member force-displacement transformation matrices are formed and stored on a separate low speed storage file along with the appropriate displacement numbers.

The NODES operation is used to specify or generate the geometry of the system. The operation BOUND specifies which joint displacements exists and assigns internal equation numbers to these displacements. Therefore, each joint may have from zero to six displacement degrees of freedom. Tables of material and section properties for the various members are loaded and printed as standard arrays of information.

A special operation ADDSF is used for the direct addition of element stiffnesses to form the total stiffness and diagonal mass matrix of the system. The LOADS operation specifies the concentrated joint loads for all load conditions. After the direct solution for joint displacements due to static or dynamic loads the member forces can be evaluated using the FORCE operation. The DISPL operation is used to print the displacement in joint number order.

+
NODES,M1,N1

The cards following this operation provide information for the creation of a N1 x 3 array which will contain the coordinates for all the joints in the system. Where

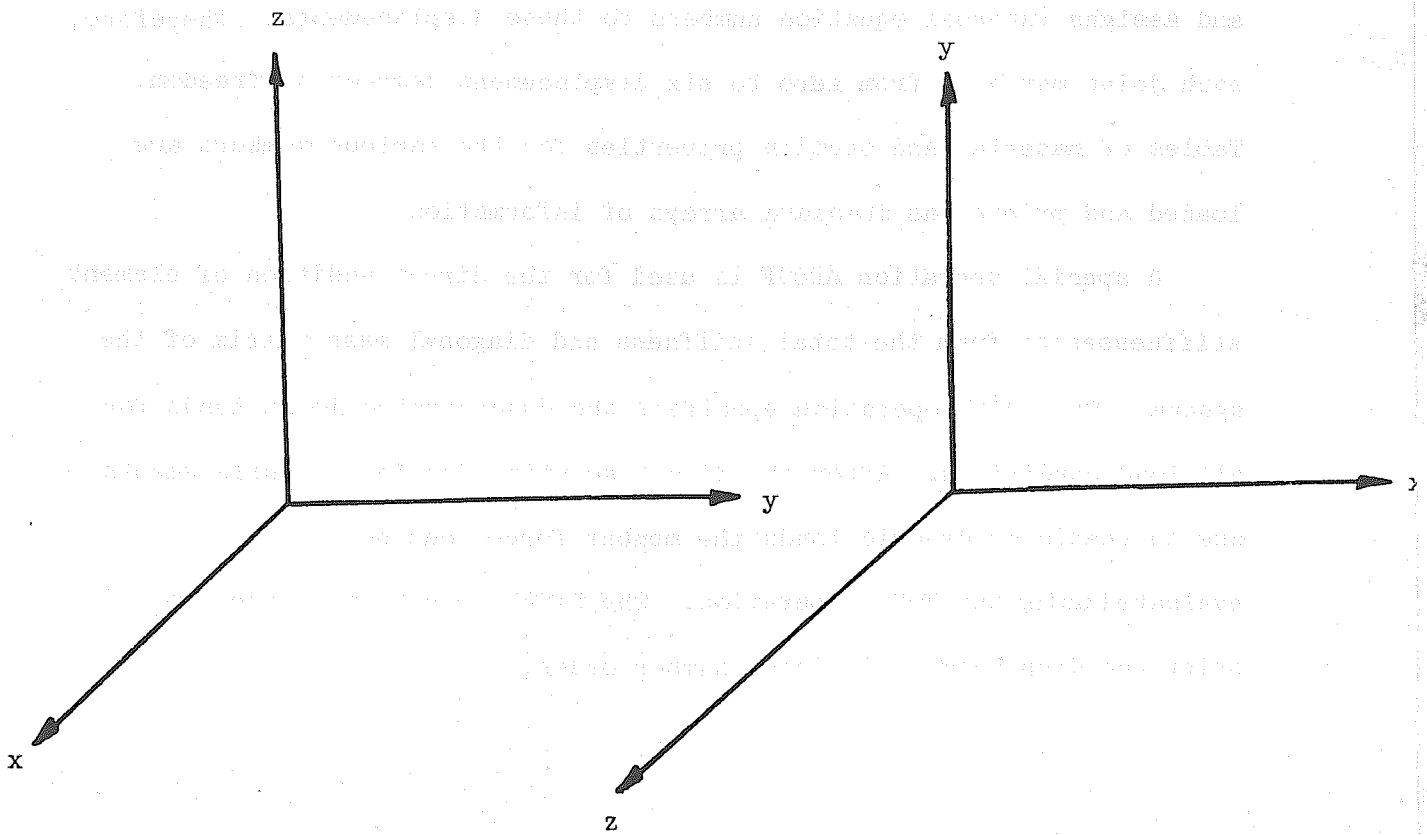
M1 = Name of new coordinate array to be loaded.

N1 = Number of joints (or nodes) in the system.

The following sequence of cards punched in a (2I, 3F10.0) format must follow this operation.

Columns	1 - 5	Node number selected by user
	6 - 10	blank
	11 - 20	X-ordinate
	21 - 30	Y-ordinate
	31 - 40	Z-ordinate

Node cards may be supplied in any order; however, all nodes must be defined. If nodes are defined more than once the last definition will be used. This sequence of data must be terminated with a blank card.



POSITIVE COORDINATE SYSTEMS

BOUND,M1

This operation specifies the displacements which are nonzero for the structural sytems of joints specified by the NODES operation. Where

M1 = Name of boundary condition code array to be generated.

This operation is followed by a series of cards containing the following information and punched in a (8I5) format.

Columns	1 - 5	Node number for the first node in a series of nodes with identical displacement specification.
	6 - 10	Node number for the last node in the series.
	11 - 15	X-translation
	16 - 20	Y-translation
	21 - 25	Z-translation
	26 - 30	X-rotation
	31 - 35	Y-rotation
	36 - 40	Z-rotation

EQ. zero for zero or undefined displacements

EQ. 1 for nonzero displacements to be evaluated by other operations

	41 - 45	Node number increment used to generate boundary conditions for additional nodes.

If a node boundary condition is not specified all displacements at that node are assumed zero. Cards may be supplied in any order. If node boundary conditions are specified more than once, the last definition is used. This sequence of data must be terminated by a blank card.

The selection by the user of which nodes have nonzero displacements requires an understanding of the direct stiffness procedure. Displacement degrees of freedom which have no stiffness associated with the displacement must be considered to be undefined since it is not possible to develop an equilibrium equation for that direction. The total number of nonzero displacements specified will be the size of the total stiffness matrix to be defined by the ADDSF operation.

+

BEAM, M1,M2,M3,M4

This operation calculates the element stiffness, mass and force-displacement transformation matrices for 3D beam members. These array are stored in sequence on low speed storage to be used by other operations where:

M1 is the name of the beam element group
M2 is the name of the coordinate array
M3 is the name of the boundary condition array
M4 is the name of an array which contains beam properties and has been loaded by the standard matrix LOAD operation.

One card for each beam in this group of beam elements must follow this operation. The beam cards are punched in the following FORMAT (5I5).

Columns	1 - 5	Beam identification number
	6 - 10	Node number I
	11 - 15	Node number J
	16 - 20	Node number K
	21 - 25	Beam property number NP

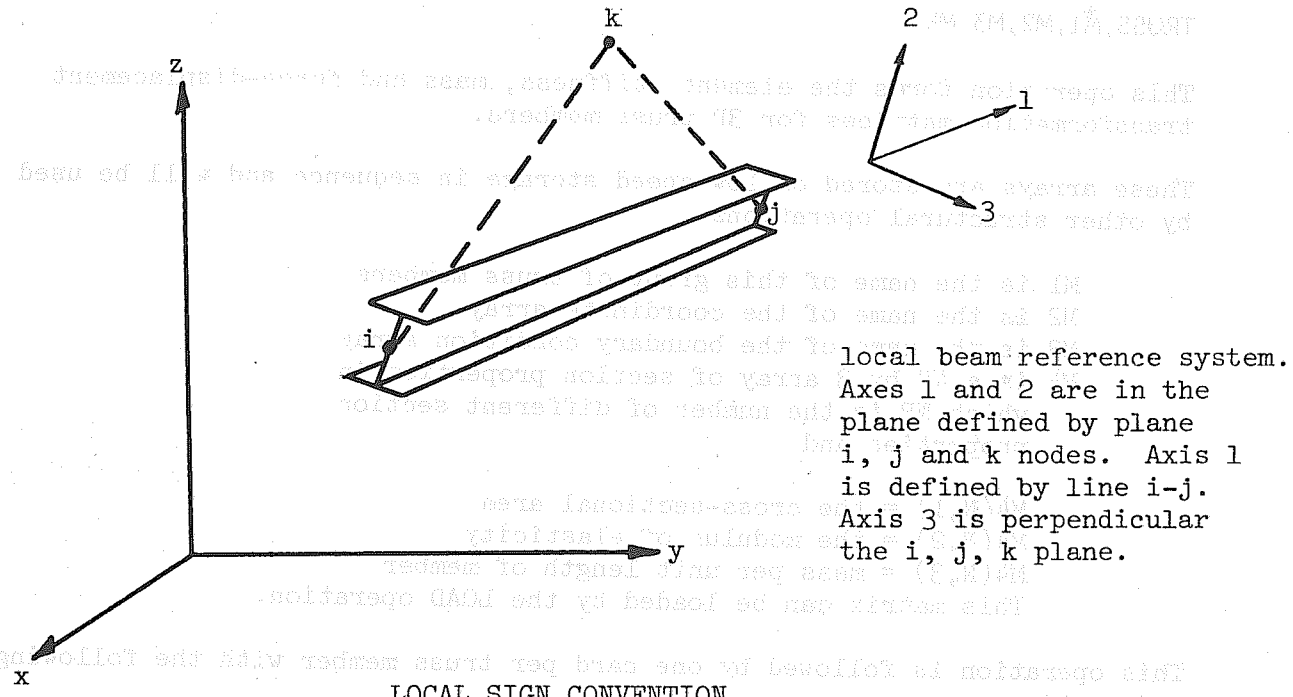
This sequence of cards must be terminated with a blank card.

The material and geometric properties for each element are given in the M4 array in the following order:

M4(NP,1) = Axial area of member, A.
M4(NP,2) = Torsional moment of Inertia, J
M4(NP,3) = Moment of Inertia about axis 2, I_{22}
M4(NP,4) = Moment of Inertia about axis 3, I_{33}
M4(NP,5) = Modulus of Elasticity, E.
M4(NP,6) = Shear Modulus, G.
M4(NP,7) = Mass per unit length of beam

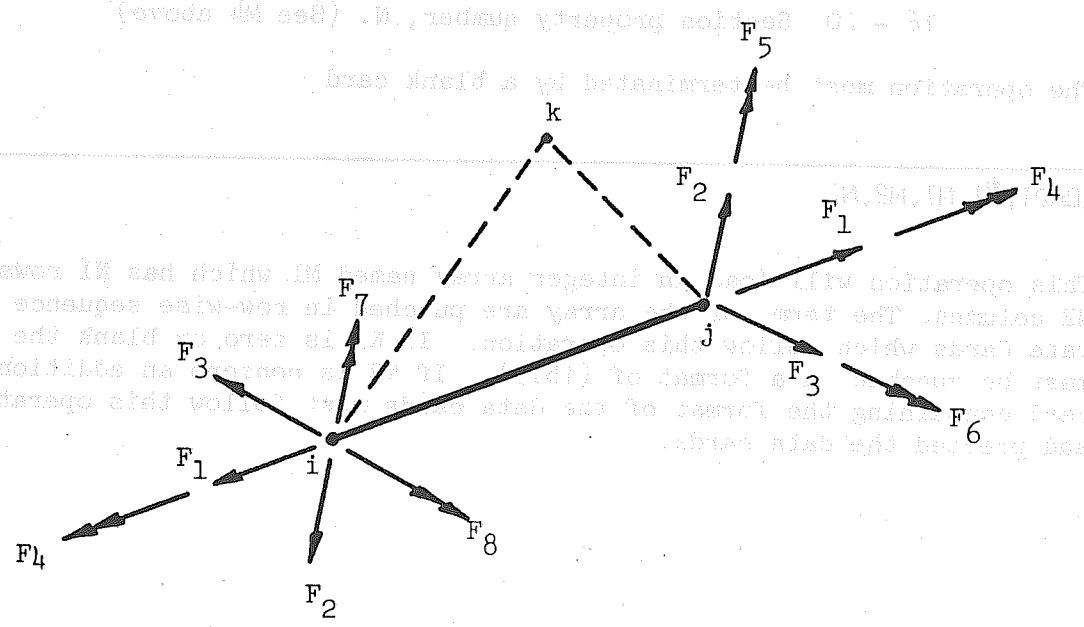
where NP is the specific material property number specified in columns 21-25 of the beam card.

The local sign convention for the beam element is given in the figure shown on page 13.



local beam reference system. Axes 1 and 2 are in the plane defined by plane i, j and k nodes. Axis 1 is defined by line i-j. Axis 3 is perpendicular the i, j, k plane.

LOCAL SIGN CONVENTION



DEFINITION OF POSITIVE BEAM FORCES

TRUSS,*M1,M2,M3,M4

This operation forms the element stiffness, mass and force-displacement transformation matrices for 3D truss members.

These arrays are stored on low speed storage in sequence and will be used by other structural operations.

M1 is the name of this group of truss members
M2 is the name of the coordinate array
M3 is the name of the boundary condition array
M4 is a NP by 3 array of section properties in which NP is the number of different section properties and

M4(N,1) = the cross-sectional area
M4(N,2) = the modulus of elasticity
M4(N,3) = mass per unit length of member
This matrix can be loaded by the LOAD operation.

This operation is followed by one card per truss member with the following information:

Columns	1 - 5	Truss member identification number
	6 - 10	Joint Number I
	11 - 15	Joint Number J
	16 - 20	Section property number, N. (See M4 above)

The operation must be terminated by a blank card

LOADI,*M1,N1,N2,N3

This operation will load an integer array named M1 which has N1 rows and N2 columns. The terms of the array are punched in row-wise sequence on Data Cards which follow this operation. If N3 is zero or blank the data must be punched in a format of (16I5). If N3 is nonzero an additional card containing the format of the data cards must follow this operation and preceed the data cards.

+
PLANE,M1,M2,M3,M4,N1,N2

This operation calculates the element stiffness, mass and stress-displacement transformation matrices for 3 to 8 node isoparametric elements. (Y-Z plane only). These arrays are stored in sequence as a group on low speed storage to be used later by other operations (i.e. ADDSF and FORCE). The arguments are defined as

M1 is the user defined name of the element group
M2 is the name of the joint coordinate array
M3 is the name of the boundary condition array
M4 is the name of the array which contains the material properties of the elements (one row per different material) where

M4(NP,1) = Modulus of Elasticity, E
M4(NP,2) = Poissons Ratio, ν
M4(NP,3) = Thickness of the element (for plane problems)
M4(NP,4) = Mass density of element

in which NP is the material identification number.

N1 and N2 are the number of integration points in the r and s directions.

One card for each 3 to 8 node element in the group must follow the operation card. The card is punched in a (10I5, 6F5.0) format and contains the following information:

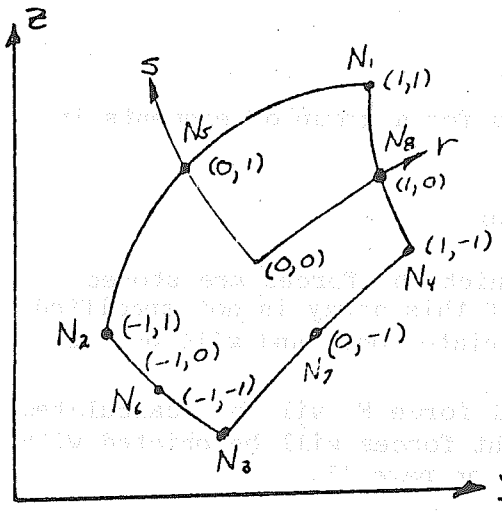
Columns:

1 - 5	element identification number	
6 - 10	Node Number	N_1
11 - 15	" "	N_2
16 - 20	" "	N_3
21 - 25	" "	N_4
26 - 30	" "	N_5
31 - 35	" "	N_6
36 - 40	" "	N_7
41 - 45	" "	N_8

Optional midside nodes - must be within center half of side.

46 - 50	Material Identification (row number in array M4)	
51 - 55 and 56 - 60	Natural Coordinates of Stress Output	r_1 and s_1
61 - 65 and 66 - 70	" " " " "	r_2 and s_2
71 - 75 and 76 - 80	" " " " "	r_3 and s_3

The local numbering system for the element is shown below. Stresses will be printed by the FORCE operation at three points, where



$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \end{bmatrix} = \begin{bmatrix} \tau_{xx}^{(1)} \\ \tau_{yy}^{(1)} \\ \tau_{xy}^{(1)} \\ \tau_{xx}^{(2)} \\ \tau_{yy}^{(2)} \\ \tau_{xy}^{(2)} \\ \tau_{xx}^{(3)} \\ \tau_{yy}^{(3)} \\ \tau_{xy} \end{bmatrix}$$

+
LOADS,M1,M2,N1

This operation forms a load matrix named M1 of N1 columns (N1 load conditions) where M2 is the name of the boundary condition array generated by the operation BOUND. This operation is followed by a series of cards--one for each loaded joint for each load condition. These cards are punched in the following format (2I5,F10.0).

Column	1 - 5	- Joint Number
	6 - 10	- Load Condition Number
	11 - 20	- Load in X-direction
	21 - 30	- Load in Y-direction
	31 - 40	- Load in Z-direction
	41 - 50	- Moment about X-axis
	51 - 60	- Moment about Y-axis
	61 - 70	- Moment about Z-axis

These series of cards must be terminated by a blank card.

+ + +
ADDSF,M1 OR ADDSF, M1,M2

This operation forms the total stiffness matrix named M1 and a lumped mass matrix named M2 for the structural system from the element stiffness and mass matrices which are stored on low speed storage. These matrices can be printed with the PRINT operation. If M2 is not specified the row mass matrix M2 will not be formed.

DISPL,M1,M2

This operation prints the displacements named M1 in joint sequence order. Where M2 is the name of the boundary condition array.

+
FORCE,M1,M2 OR FORCE,M1,M2,M3

This operation calculates the member forces for a group of elements in which

M1 = the name of the element group
M2 = the displacement matrix
M3 = the name of the matrix in which the forces are stored in the order calculated. If this array is not specified the element forces will be printed only and will not be retained in storage.

For the TRUSS element only the member axial force F will be calculated for each member. For the BEAM element eight forces will be printed with reference to the positive definition shown on page 13.

D. OPERATIONS FOR THE DYNAMIC ANALYSIS OF STRUCTURAL SYSTEMS

The following operations were designed to evaluate the dynamic response of structures subjected to arbitrary time-dependent loads. If these operations are used in connection with the standard matrix operations and the structural analysis operations a dynamic analysis is a relatively simple procedure. The user has the option of using the mode superposition method or a direct step-by-step integration of the dynamic equations of motion. The user may examine the spectra of both input loading and calculated displacements. In addition, the contributions of the individual modes may be evaluated and compared.

The most common and convenient form for time-dependent data to be specified is as straight line segments between given time points. Therefore, an operation which generates values at equal intervals is necessary. Another common characteristic of time-varying loads on structures is that it is normally possible to represent the loads at all points on the structure by the product of two matrices--a column matrix indicating the spacial distribution of loads times a row matrix which indicates the values at a function of various times. If a more complicated loading is required it is possible to perform more analyses, each within the restrictions of the program, and then add the results of each analysis.

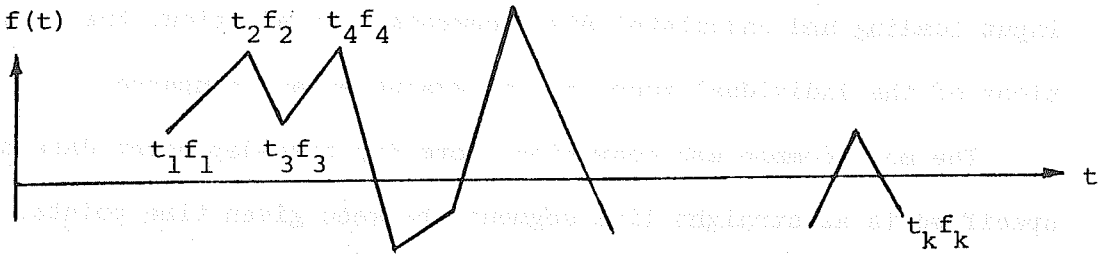
In addition to several other matrix operations within CAL the following operations have been added for the major purpose of performing dynamic analysis.

FUNG,M1,M2,M3,N1,N2

This operation generates a matrix named M2 which contains values, at equal intervals, of the function specified in the array named M1. The array M1 must be a 2 by k array of the form

$$M1 = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \cdots\cdots\cdots\cdots t_k \\ f_1 & f_2 & f_3 & f_4 & f_5 & f_6 \cdots\cdots\cdots\cdots f_k \end{bmatrix}$$

which numerically represents a function of the form shown below



The time interval Δt is specified in the 1 by 1 matrix named M3. N1 specifies the total number of values to be generated which is the number of columns in M2. If N2 = 0 the array M2 will be a 1 x N1 row matrix in which the first value will be f_1. If N2 $\neq$ 0 the array M2 will be a 2 x N1 matrix of the following form

$$M2 = \begin{bmatrix} t_1 & t_1 + \Delta t & t + 2\Delta t \cdots\cdots\cdots\cdots \\ f_1 & f(t_1+\Delta t) & f(t_1 + 2\Delta t) \cdots\cdots\cdots \end{bmatrix}$$

STEP,$\bar{M}1$,M2,M3,$\bar{M}4$,$\overset{+}{M}5$,M6,M7,M8,N1,N2

This operation calculates the dynamic response of a structural system using direct step-by-step integration of the following linear matrix equations of motion:

$$\underline{M}\ddot{\underline{U}} + \underline{C}\dot{\underline{U}} + \underline{K}\underline{U} = \underline{R}(t) = \underline{PF}(t)$$

where

M1 is the name of the N by N stiffness matrix $\underline{K}$.
M2 is the name of the N by N mass matrix $\underline{M}$.
M3 is the name of the N by N damping matrix $\underline{C}$.
M4 is the name of the N by 3 initial condition matrix $\underline{U}$ in which:
$U_o(I,1)$ is a vector of displacements $\underline{U}_o$.
$U_o(I,2)$ is a vector of velocities $\dot{\underline{U}}_o$.
$U_o(I,3)$ is a vector of accelerations $\ddot{\underline{U}}_o$.
M5 is the name of the N by N2 matrix of calculated displacements in which column "i" represents the displacements at time $i \cdot N1 \cdot \Delta t$.
M6 is the name of the N by 1 load distribution matrix $\underline{P}$.
M7 is the name of the 1 by k row matrix representing the load multipliers at equal time increments $\underline{F}$, where $k = N2/N1$.
M8 is the name of the 1 by 1 matrix containing Δt.
N1 is the output interval for the displacements.
N2 is the total number of displacement vectors to be calculated. Therefore, the total time for which results will be calculated. will be $N1 \cdot N2 \cdot \Delta t$.

This operation must be followed with one data card containing the following information:

Column 1 - 10 δ

11 - 20 α

21 - 30 θ

Different values of δ, α and θ will allow the user to select different methods of step-by-step integration. The following table lists some possibilities:

	δ	α	θ
Newmarks Average Acceleration	1/2	1/4	1.0
Linear Acceleration	1/2	1/6	1.0
Wilson's θ Method (low damping)	1/2	1/6	1.42
Wilson's θ Method (high damping)	1/2	1/6	2.00

The method of integration used is given on pages 32 to 36.

```
      -  +  -
EIGEN,M1,M2,M3,N1
```

This operation solves the following eigenvalue problem;

$$K \phi = M \phi \lambda$$

In which the N by N, symmetric, positive-definite matrix K is named M1. The matrix M is a diagonal matrix of positive terms designated by M3. The matrix M3 must be a row or column matrix containing only the diagonal terms of M.

The eigenvalues λ are stored in matrix M3. The eigenvalues are ordered in numerically increasing order and the eigenvectors ϕ are stored in the corresponding columns of the matrix M2.

The number N1 specifies the approximate number of significant figures of the eigenvalues. If N1 is zero or blank 4 figure accuracy will be used.

The program reduces the problem to standard eigenvalue form by the following transformation

$$K^* = m^T K\ m$$

where

$$I = m^T M\ m$$

in which

$$m_i = \frac{1}{\sqrt{M_{ii}}}$$

The calculated mode shapes ϕ are normallized as follows:

$$\phi^T M \phi = I$$

$$\phi^T K \phi = \lambda$$

The program uses the standard Jacobi diagonalization method to solve for all eigenvalues and eigenvectors. See page 445, reference [2], for a complete discussion of the Jacobi method.

DYNAM,M1,M2,M3,M4,M5,M6,N1

This operation evaluates the following set of uncoupled second order differential equations associated with the mode superposition method for the dynamic analysis of a structural system.

$$\ddot{X}_i + 2\lambda_i\omega_i X_i + \omega_i^2 X_i = P_i(t) \quad i = 1 \text{ to } N \text{ modes}$$

M1 is the name of a row or column matrix which contain the N ω_i terms (frequencies in rad/sec). M2 is the name of a row or column matrix which contain the N λ_i terms (ratio of modal damping to critical damping).

The generalized time-varing forces $P_i(t)$ are not specified directly but are evaluated from more fundamental information. The forces for all modes are evaluated at specific times by the program from the following matrix equation:

$$\underline{P} = \underline{p}\,\underline{f} = M3 \cdot M4$$

In which $\underline{p}$ is a specified N by 1 vector named M3, and $\underline{f}$ is a 1 by N1 row matrix which will be generated from the 2 by k array named M4. (The array M4 is the same form as the input array described under the operation FUNG; therefore, it is not necessary to use FUNG before the DYNAM operation.)

M5 is the name of the N x N1 array which contain the generalized displacement $X_i(t)$.

M6 is the name of the 1 x 1 array which contains the time increment associated with the generalized displacements.

N1 is the number of displacements to be generated.

The method of integration used is exact for straight line segments and is summarized on page 52.

PLOT,M1,N1

This operation will prepare a printer plot of selective rows of the matrix named M1. N1 is the number of rows of M1 which will be plotted by this operation. This operation is followed by N1 cards--one for each row plotted with the following information:

Columns	1	Plot symbol--any key punch symbol.
	2 - 5	Row number to be plotted.

The program automatically searches the information to be plotted for the maximum and minimum values. The difference in this numbers divided by 120 spaces is selected as the plot scale.

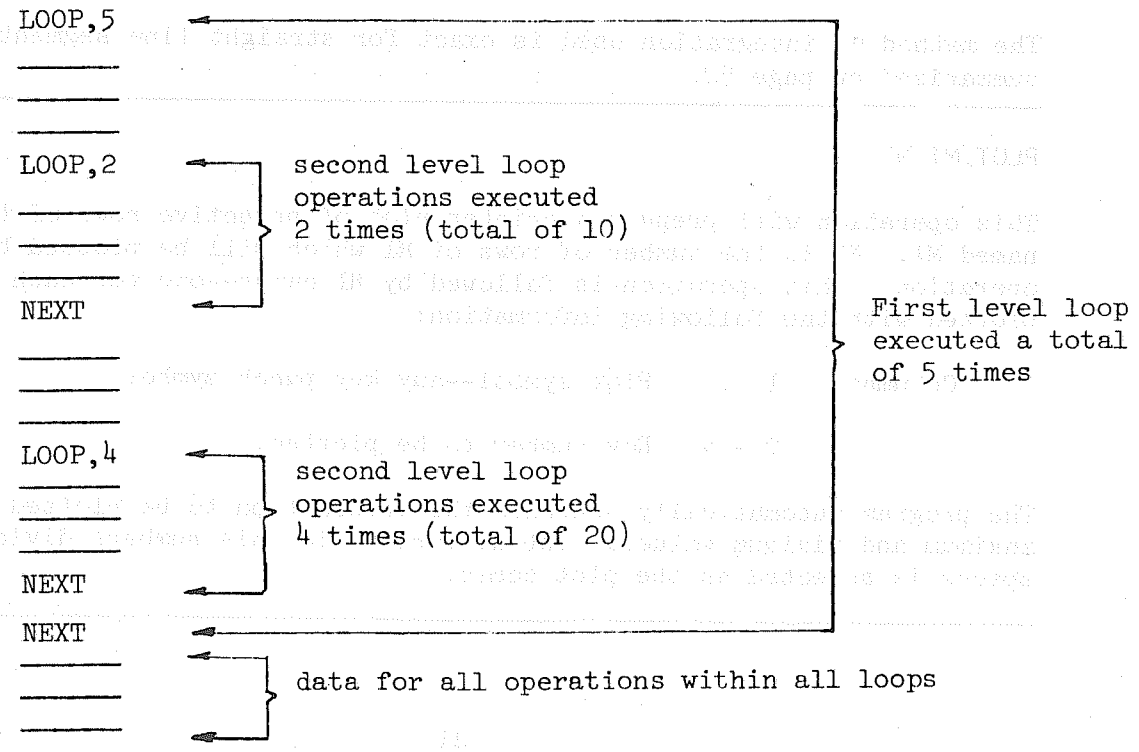
E. LOOPING OPERATIONS WITHIN CAL

LOOP,N1 NEXT,M1 and SKIP,M1

CAL has a five level looping ability. The first operation is LOOP and the last operation is NEXT. Operations within CAL are normally executed as they are encountered and if the operation requires data the data cards follow the operation card. In the case of looping, however, all operation cards from the first LOOP card to the last NEXT card are stored within the computer before they are executed; therefore, if operations within the loops require data, the data card be supplied in the order required after the last NEXT operation. The general form of the operation is

LOOP,N1

Where N1 is the number of times the loop is to be executed. Associated with each LOOP operation there must be a corresponding NEXT operation which signifies the end of the loop and the return of the control to the beginning of the loop. The following is a possible series of looping operations.



It is apparent which LOOP and NEXT cards are associated if there are an equal number of each. After all loops are executed the computer storage required for these operations is automatically released by the program. The operation NEXT,M1 will cause the loop to terminate if the first term in matrix named M1 is negative.

SKIP,M1,N1

This operation will cause the skip of the next N1 operations if the first term in the matrix named M1 is negative. The operation cannot be used to skip in or out of a specific loop; therefore, it can be used only to skip operations within the same level of looping.

F. USER DEFINED OPERATIONS

USERA and USERB

These names which are reserved for operations to be defined and programmed by the user. In order to program these operations it is necessary to understand the internal organization of CAL.

III. NUMERICAL METHODS FOR DYNAMIC ANALYSIS

A. INTRODUCTION

Some of the numerical techniques used in the program CAL are presented here; however; several additional numerical methods for dynamic analysis are given in order to expose the CAL user to more advanced methods of dynamic analysis.

The value of the results of a dynamic analysis depends on the approximations involved in the establishment of mathematical models for the structure and foundation and in the selection of the various dynamic load conditions. In general, the establishment of the models and the interpretation of the results are the most critical phases of a dynamic analysis. This assumes that the particular method of dynamic analysis used does not introduce additional errors in the solution of the model for the specified loads. It is important that the computer cost for any one analysis is not large in order that inexpensive reanalysis is possible and the results of the analysis can be used by the engineer to influence the basic design of the structure. Also, an economical computer analysis will allow some of the basic assumptions used in selecting the model and loads to be varied and the sensitivity of the results evaluated.

The effectiveness of a numerical method for dynamic analysis depends primarily on two factors--the minimization of computer storage and of computer execution time. The purpose of this section is to present a summary of various numerical methods which are used in the dynamic analysis of complex structures and to comment with respect to their computer implementation and efficiency.

It should be noted that all structures, regardless of their simplicity, have an infinite number of degrees of freedom when subjected to dynamic loading. One of the main objectives of selecting a mathematical model is to reduce the infinite degree of freedom system to a model with a limited degree of freedom which will capture the significant physical behavior of the system. Therefore, a considerable insight into the expected dynamic behavior of the system must be present if a realistic mathematical model is to be established. The model must be capable of representing both the significant wave propagation and structural vibration behavior.

The force equilibrium of a complex structure modeled as a system with a finite number of degrees of freedom may be written as

$$F_i + F_d + F_s = F \qquad (1)$$

in which all forces are a function of time and defined as

F_i = The inertia force

F_d = The internal and external damping forces

F_s = The forces carried by the structural members

F = The external applied forces

Equation (1) holds for both linear and nonlinear systems. However, the appropriate numerical method for solution will depend on the degree of nonlinearity which is present and if linearization is possible.

B. LINEAR DYNAMIC ANALYSIS

For linear systems with viscous damping equation (1) can be written in the form

$$M\ddot{U}(t) + C\dot{U}(t) + KU(t) = F(t) \tag{2}$$

in which M is the mass matrix (lumped or consistent), C is the damping matrix (normally not given in the form) and K is the stiffness matrix for the system of structural elements. The time dependent vector U(t), $\dot{U}(t)$ and $\ddot{U}(t)$ are the displacements, velocities and accelerations, respectively.

The time dependent forces F(t) may be due to moving equipment, blast, wind, wave or seismic forces. Depending on the type of structure the calculation of these forces can be complicated and will not be discussed here. However; in case of earthquake motion in three-dimensions the loading is of the form:

$$F(t) = - M_x\ddot{U}_{xg} - M_y\ddot{U}_{yg} - M_z\ddot{U}_{zg} \tag{3}$$

where $\ddot{U}_{ig}$ is the ground acceleration in direction i and M_i is a column matrix which represents the sum of all columns in the mass matrix M associated with displacements in the i-direction [1]. This definition of the seismic loading is valid only if the vector is defined as the displacement relative to the displacement at the base of the structure.

In many cases wave and seismic loads in each direction are specified in terms of a spectrum of maximum response values. In the case of three-dimensional behavior, however, great care must be taken in the application of the technique since the basic input in the various directions must be statistically independent if the results are to be combined in a probabilistic manner.

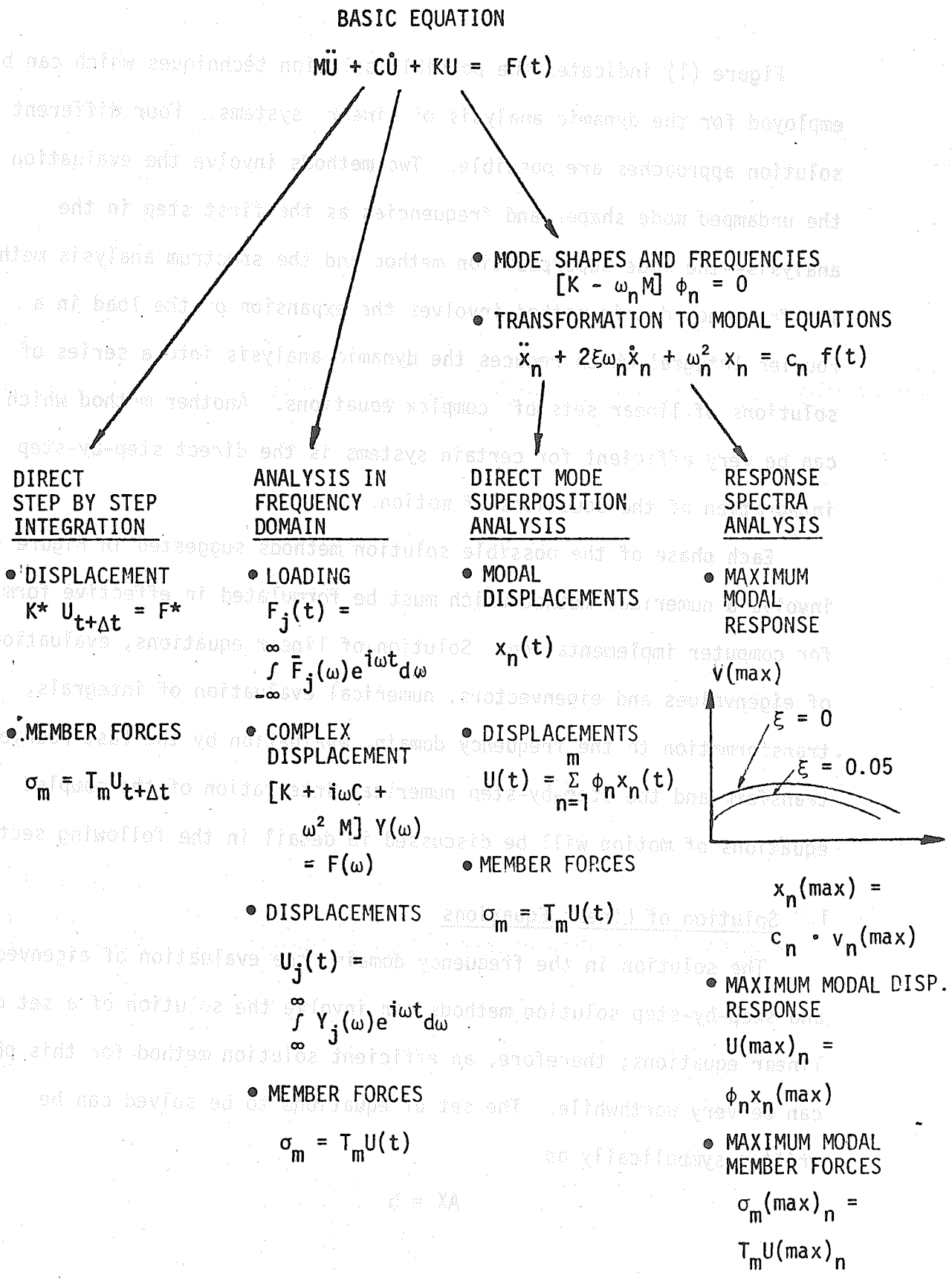


Figure 1. METHODS FOR LINEAR DYNAMIC ANALYSIS

Figure (1) indicates the possible solution techniques which can be employed for the dynamic analysis of linear systems. Four different solution approaches are possible. Two methods involve the evaluation of the undamped mode shapes and frequencies as the first step in the analysis--the mode superposition method and the spectrum analysis method. The frequency domain method involves the expansion of the load in a Fourier Integral which reduces the dynamic analysis into a series of solutions of linear sets of complex equations. Another method which can be very efficient for certain systems is the direct step-by-step integration of the equations of motion.

Each phase of the possible solution methods suggested in Figure (1) involve a numerical method which must be formulated in effective form for computer implementation. Solution of linear equations, evaluation of eigenvalues and eigenvectors, numerical evaluation of integrals, transformation to the frequency domain, evaluation by the fast Fourier transform and the step-by-step numerical integration of the coupled equations of motion will be discussed in detail in the following sections.

1. Solution of Linear Equations

The solution in the frequency domain, the evaluation of eigenvectors and step-by-step solution methods can involve the solution of a set of linear equations; therefore, an efficient solution method for this phase can be very worthwhile. The set of equations to be solved can be written symbolically as

$$AX = b \tag{4}$$

where A is an N x N symmetrical matrix, X is a vector of unknowns corresponding with the specified vector b. One of the most important aspects in the computer solution of a large set of equations is the method used to store the terms in the A matrix. One method which has been found to be very effective is the active column storage technique in which only the non-zero terms in the reduced matrix are stored. If a basic elimination method is used the only storage required will be from the first non-zero terms in each column down to the diagonal term in that column. Therefore, the matrix with terms initially located as indicated in Figure 2a can be stored as a one-dimensional array as shown in Figure 2b along with an integer pointer array indicating the location of each diagonal term. Figure 2c indicates how the storage technique is extended to approximately equal size blocks which can be stored on low speed storage. The solution technique requires two blocks in high speed core storage at any particular time. It has been demonstrated that most of the popular methods for the solution of equations are very similar, with respect to the number of numerical operations, and they can be considered to be variations of the Gauss elimination method [2].

The basic factorization algorithm for the solution of equation stored in active column form may be summarized by the following three steps:

a. <u>Triangularization of A</u> (j = 2 to N)

$$A = LU \quad \text{or} \quad A = LDL^T \tag{5}$$

in which the j^{th} column of upper triangular matrix U is evaluated from

$$U_{ij} = A_{ij} - \sum_{k = km}^{j-1} L_{ik} U_{kj} \qquad i = fj \text{ to } j \tag{6}$$

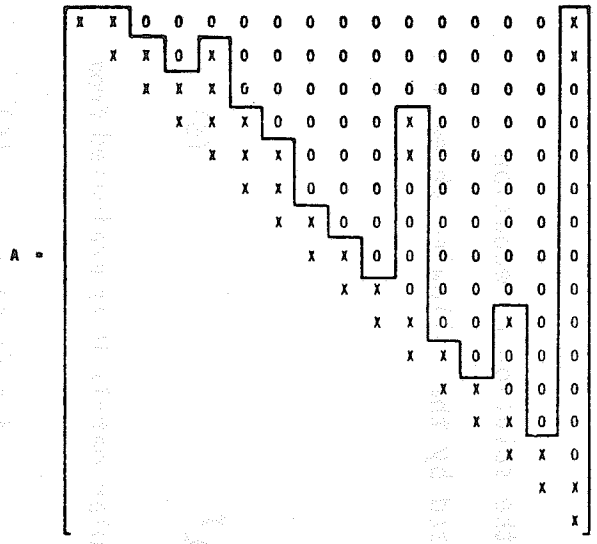


Fig. 2a Elements in Original Matrix

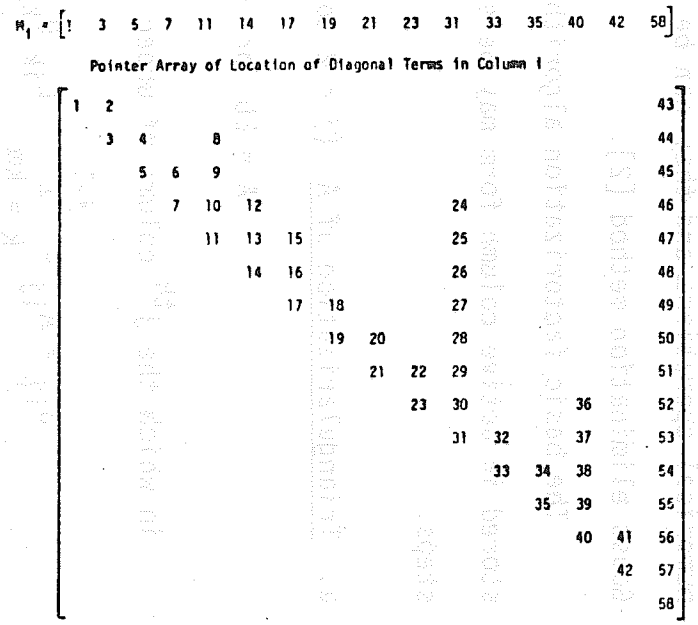
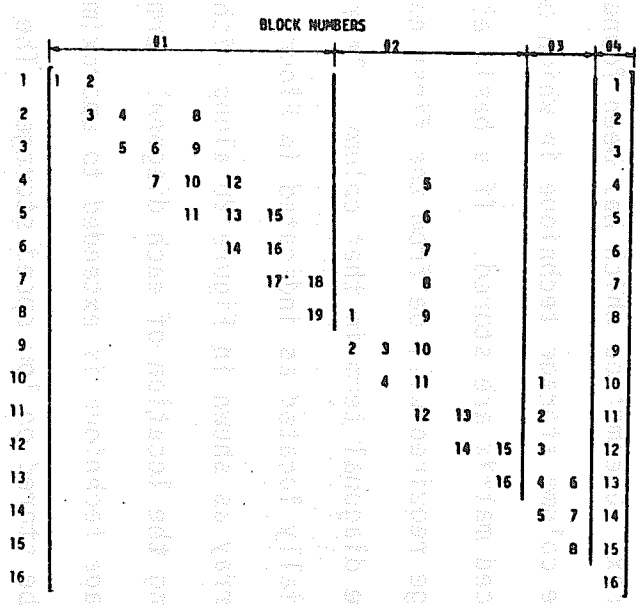


Fig. 2b Storage Sequence for Incore Active Column Equation Solver

$$AX = b \qquad A = LU = LDL^T$$



K(1)	K(2)	K(3)	K(4)	M(J)							
1	1	1	8	1	3	5	7	11	14	17	19
2	1	9	13	2	4	12	14	16			
3	2	14	15	5	8						
4	1	16	16	16							

Fig. 2c Example of Matrix Stored in a Maximum of 20 Blocks.

and the j^{th} row of lower triangular matrix is given by

$$L_{ji} = U_{ij}/D_{ii} \qquad i = fj \text{ to } j-1 \qquad (7)$$

where D is a diagonal matrix, fj is the first non-zero term in column j and fi is the first non-zero term in column i. The symbol km represents maximum of fj and fi.

The diagonal terms U_{ii} and D_{ii} are identical since L_{ii} is normalized as 1.0. It is most convenient to evaluate U_{ij} within a computer program column-wise with i = fj to j. After each column is complete L_{ji} is evaluated row-wise with i = fj to j-1 and stored in transposed form as L^*_{ij} where U_{ij} was previously located. The diagonal term U_{ii} or D_{ii} remain at the same location as A_{ii}.

b. Forward Reduction

Equation (4) can be written as $Lz = b$, where $z = DL^T$. Therefore

$$z_i = b_i - \sum_{k=fi}^{i-1} L^*_{ki}\, z_k \qquad i = 1 \text{ to } N \qquad (8)$$

If y is defined as $y = D^{-1} z$ then $y = L^T x$; or

$$y_i = z_i/D_{ii} \qquad i = 1 \text{ to } N \qquad (9)$$

c. Backsubstitution

From $y = L^T x$

$$x_i = y_i - \sum_{k=i+1}^{N} L^*_{ik}\, y_k \qquad i = N \text{ to } 1 \qquad (10)$$

It is important to note that all zero operations are skipped by this technique and that the number of operations or the required storage is not a function of the band width. Also, the triangularization, operations on the A matrix, is independent of the forward or backsubstitution operations;

therefore, this operation need be done only once. The forward and back-substitution phases for each load condition normally involve a small number of operations compared to triangularization. Recognition of this can greatly minimize the numerical effort required in some eigenvalue methods and in the direct step-by-step integration of the equations of motion.

2. Step-by-Step Integration

The direct integration of the linear dynamic equations of motion is a simple approach which can have considerable advantages for some problems. The basic equation is satisfied at discrete points in time, 0, Δt, $2\Delta t$, $3\Delta t$, ... t, $t+\Delta t$, ... T. The solution starts from a point in time where the displacements, velocities and accelerations are known. Based on an assumption on the behavior of the system during the next small increment of time the displacements, velocities and accelerations at the next point in time can be evaluated. Many different methods have been developed for this purpose, References [2] to [12]. However, only two techniques will be summarized in this paper.

a. The Central Difference Method

In the case of a diagonal mass and damping matrix and for systems where the shortest period is not too small the central difference method has proven to be most effective. At time t the equation to be satisfied is

$$M\ddot{U}_t + C\dot{U}_t + KU_t = F_t \qquad (11)$$

The following standard finite difference relationships are used:

$$\ddot{U}_t = \frac{1}{\Delta t^2}\left(U_{t-\Delta t} - 2U_t + U_{t+\Delta t}\right) \qquad (12)$$

$$\dot{U}_t = \frac{1}{2\Delta t}\left(U_{t+\Delta t} - U_{t-\Delta t}\right) \tag{13}$$

Equation (12) and (13) can be substituted into equation (11) to form a set of linear equations of the form

$$M^* U_{t+\Delta t} = F^* \tag{14}$$

where

$$M^* = \frac{1}{\Delta t^2} M + \frac{1}{2\Delta t} C \tag{15}$$

$$F^* = F_t - KU_t + \frac{1}{\Delta t^2} M\left(2U_t - U_{t-\Delta t}\right) + \frac{1}{2\Delta t} C\, U_{t-\Delta t} \tag{16}$$

If M and C are diagonal one notes that the solution of equation (14) is trivial; also, computer storage will be minimized. Another very important technique which can be used to further minimize the number of numerical operations and computer storage requirements is not to form the complete stiffness K. The structural forces KU_t can be evaluated element by element, or

$$F_s = KU_t = \sum_m K_m U_t$$

in which K_m is the stiffness matrix for element m. If elements have identical stiffness matrices a further reduction in computer storage can be realized.

The solution $U_{t+\Delta t}$ is based on using the equilibrium at time t; therefore, this approach is called an "explicit integration method." One of the most significant disadvantages of the central difference method is that it is only conditionally stable [2]. In order for the method to produce finite results the time step Δt must be less than T_n/π. Where T_n is the shortest period in the discrete model. For many structures this requires time steps so small that the method may be impractical.

b. The Newmark-Wilson Method

One of the most flexible step-by-step integration methods has been presented by Newmark [8]. This method is based on the following expressions for the velocity and displacement at the end of the time interval:

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta t\,(1 - \delta)\,\ddot{U}_t + \Delta t\,\delta\,\ddot{U}_{t+\Delta t} \tag{17}$$

$$U_{t+\Delta t} = U_t + \Delta t\,\dot{U}_t + \Delta t^2\,(\frac{1}{2} - \alpha)\,\ddot{U}_t + \Delta t^2\,\alpha\ddot{U}_{t+\Delta t} \tag{18}$$

where α and δ are selected to produce the desired accuracy and stability. If $\delta = 1/2$ and $\alpha = 1/6$ the well known linear acceleration is produced, which is also a conditionally stable method. One of the most widely used methods is the constant-average-acceleration method ($\delta = 1/2$ and $\alpha = 1/4$) which is an unconditionally stable method without numerical damping.

This method is called an "implicit integration method" since it satisfies the equilibrium equations of motion at time $t+\Delta t$, or

$$M\ddot{U}_{t+\Delta t} + C\dot{U}_{t+\Delta t} + KU_{t+\Delta t} = F_{t+\Delta t} \tag{19}$$

This equation can be solved by iteration; however, equations (17), (18) and (19) can be combined into a step-by-step algorithm which involves the solution of a set of equations at each time step of the form

$$K^*U_{t+\Delta t} = F^* \tag{20}$$

Since K^* is not a function of time it can be triangularized once at the beginning of the calculations. The computer solution time for this type of algorithm is basically proportional to the number of time steps required.

The Wilson θ method is a technique which can be used to modify the basic Newmark method in order to increase the stability limits and to

TABLE 1. The Newmark-Wilson Algorithm for Linear Step-by-Step Integration

A. INITIAL CALCULATIONS:

1. Form stiffness matrix K, mass matrix M and damping matrix C.

2. Initialize U_0, $\dot{U}_0$, and $\ddot{U}_0$.

3. Specify algorithm parameters α, δ and θ

$$\delta \geq 0.50 \quad ; \quad \alpha \geq 0.25(0.5 + \delta)^2 \quad ; \quad \theta \geq 1.0$$

4. Calculate integration constants:

$$\tau = \theta\Delta t \qquad a_3 = \frac{1}{2\alpha} - 1 \qquad a_7 = \Delta t\delta$$

$$a_0 = \frac{1}{\alpha\tau^2} \qquad a_4 = \frac{\delta}{\alpha} - 1 \qquad a_8 = \Delta t^2\left(\frac{1}{2} - \alpha\right)$$

$$a_1 = \frac{\delta}{\alpha\tau} \qquad a_5 = \frac{\tau}{2}\left(\delta/\alpha - 2\right) \qquad a_9 = \alpha\Delta t^2$$

$$a_2 = \frac{1}{\alpha\tau} \qquad a_6 = \Delta t(1 - \delta)$$

5. Form effective stiffness matrix: $K^* = K + a_0M + a_1C$

6. Triangularize K^*: $K^* = LDL^T$

B. FOR EACH TIME STEP:

1. Calculate effective load vector at time $t + \tau$:

$$F^* = F_{t+\tau} + M(a_0U_t + a_2\dot{U}_t + a_3\ddot{U}_t) + C(a_1U_t + a_4\dot{U}_t + a_5\ddot{U}_t)$$

2. Solve for displacements at time $t + \tau$:

$$LDL^T U_{t+\tau} = F^*$$

3. Calculate Accelerations, Velocities and Displacement at $t + \Delta t$:

$$\ddot{U}_{t+\tau} = a_0(U_{t+\tau} - U_t) - a_2\dot{U}_t - a_3\ddot{U}_t$$

$$\ddot{U}_{t+\Delta t} = \ddot{U}_t + \frac{1}{\theta}(\ddot{U}_{t+\tau} - \ddot{U}_t)$$

$$\dot{U}_{t+\Delta t} = \dot{U}_t + a_6\ddot{U}_t + a_7\ddot{U}_{t+\Delta t}$$

$$U_{t+\Delta t} = U_t + \Delta t\dot{U}_t + a_8\ddot{U}_t + a_9\ddot{U}_{t+\Delta t}$$

add numerical damping [13]. The θ method was first applied to the linear acceleration method in order to improve stability and has been used to damp out high frequency occillations which often develop in linear and nonlinear step-by-step integration. The technique involves using the Newmark method to find the solution at $t + \theta\Delta t$, then, based on linear acceleration, calculating the results at $t+\Delta t$ for use as initial conditions for the next time step. The Newmark-Wilson algorithm is summarized in Table 1. With $\theta = 1$ the approach is the standard Newmark method. An unconditionally stable method with large damping in the higher modes is produced with $\delta = 1/2$, $\alpha = 1/6$ and $\theta = 1.4$ [12].

3. Frequency Domain Approach

An alternative to the direct integration of the coupled linear equations of motion is to use a formal mathematical transformation to eliminate the time function from the equations before solution progresses [1]. The basic approach involves the expansion of the time-dependent loads in terms of a series of harmonic functions. One can use the standard Fourier Series in which the loads F(t) are expanded in a series of the form

$$F(t) = \sum_{n=0}^{\infty} A_n \cos \frac{n\pi}{d} t + \sum_{n=0}^{\infty} B_n \sin \frac{n\pi}{d} t \qquad (21)$$

in which d is the duration of the loading. The Fourier Coefficients can be evaluated and exact solutions found for the harmonic functions $A_n \cos \frac{n\pi}{d} t$ and $B_n \sin \frac{n\pi}{d} t$. It is assumed that the loading can be approximated by a finite number of terms. Therefore, the total solution in time is a summation of the exact solution for each harmonic function. For systems without damping this straight-forward Fourier Series approach is numerically very effective since the response of an undamped structure to a harmonic sin or cos function loading is also sin or cos displacement solution.

An alternate method of eliminating the time variable from the dynamic equilibrium equations is to express the loads as an infinite integral, or

$$F(t) = \int_0^\infty (A(\omega) \cos \omega t + B(\omega) \sin \omega t)\, d\omega \tag{22}$$

The functions $A(\omega)$ and $B(\omega)$ in the Fourier integral are given by

$$A(\omega) = \frac{2}{\pi} \int_0^d F(t) \cos \omega t\, dt \tag{23}$$

$$B(\omega) = \frac{2}{\pi} \int_0^d F(t) \sin \omega t\, dt \tag{24}$$

Also, the Fourier integral can be written in complex form as

$$F(t) = \int_{-\infty}^{\infty} \bar{F}(\omega)\, e^{i\omega t}\, d\omega \tag{25}$$

in which

$$\bar{F}(\omega) = \frac{1}{2} [A(\omega) - iB(\omega)] \tag{26}$$

$$\bar{F}(-\omega) = \frac{1}{2} [A(\omega) + iB(\omega)] \tag{27}$$

since

$$e^{i\omega t} + e^{-i\omega t} = 2 \cos \omega t$$

$$e^{i\omega t} - e^{-i\omega t} = 2i \sin \omega t$$

The general equilibrium equations can now be written as

$$M\ddot{U} + C\dot{U} + KU = \int_{-\infty}^{\infty} \bar{F}(\omega)\, e^{i\omega t}\, d\omega \tag{28}$$

The solution is assumed to be of the form

$$U(t) = \int_{-\infty}^{\infty} Y(\omega)\, e^{i\omega t}\, d\omega \tag{29}$$

therefore

$$\dot{U}(t) = \int_{-\infty}^{\infty} i\omega\, Y(\omega)\, e^{i\omega t}\, d\omega \tag{30}$$

$$\ddot{U}(t) = \int_{-\infty}^{\infty} - \omega^2 Y(\omega)\, e^{i\omega t} \qquad (31)$$

Hence, the following complex set of equations must be solved for various values of ω:

$$(K + i\omega C - \omega^2 M)\, Y(\omega) = \bar{F}(\omega) \qquad (32)$$

If $\bar{F}(\omega)$ requires a large number of points to define the complete function, it is apparent that a large number of solutions of complex equations will be necessary. This large numerical effort can be minimized by solving for the basic eigenvalues of the system and transforming the equations to a smaller system expressed in model coordinants, Eq. 67. The evaluation of the complex loads $\bar{F}(\omega)$ is not a major computational problem as compared to the multiple solution of a large system of complex equations. Furthermore, the Fast Fourier Transform algorithm can minimize both the evaluation of the Fourier transforms and the recovery of the displacements, equation (29).

The major advantage of the Frequency Domain approach is in its application to substructure analysis. Structure-foundation or structure-fluid systems are considered by the development of the frequency-dependent matrices for the separate systems. Ritz techniques can also be used to effectively reduce the size of the system.

4. Numerical Evaluation of Mode Shapes & Frequencies

For large structural systems one of the most time consuming phases of a dynamic analysis may be the evaluations of eigenvalues and eigenvectors of the N x N matrix equation

$$MU + \bar{\omega}^2 KU = 0 \qquad (33)$$

The undamped free vibration of the structural model has a solution form of

$$U(t) = \sum_{n=1}^{N} e^{i\bar{\omega}t}\phi_n$$

Therefore, the resulting eigenvalue problem must be solved

$$(K - \bar{\omega}_n^2 M)\,\phi_n = 0 \tag{34}$$

a. Static Condensation

One technique which is often used to reduce the size of the system before the evaluation of eigenvalues is to eliminate the massless degrees of freedom from the system. For this case equation (34) is rewritten as

$$\begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} \phi_a \\ \phi_b \end{bmatrix} = \bar{\omega}^2 \begin{bmatrix} 0 & 0 \\ 0 & M_b \end{bmatrix} \begin{bmatrix} \phi_a \\ \phi_b \end{bmatrix} \tag{35}$$

The first submatrix equation is

$$K_{aa}\,\phi_a + K_{ab}\,\phi_b = 0 \tag{36}$$

Therefore, the massless degrees of freedom are related to the degrees of freedom at the mass points by

$$\phi_a = T\,\phi_b \tag{37}$$

where

$$T = -K_{aa}^{-1}\,K_{ab} \tag{38}$$

The resulting eigenvalue problem is of the form

$$K_{bb}^{*}\,\phi_b = \bar{\omega}^2\,M_b\,\phi_b \tag{39}$$

in which

$$K_{bb}^{*} = K_{bb} + K_{ba}\,T \tag{40}$$

Within a computer program, however, these submatrix operations are not necessary since it is more efficient to perform the "static condensation" directly on the massless degrees of freedom similar to the Gauss elimination procedure [14] without requiring submatrix storage. One important disadvantage of the static condensation approach is that the matrix K^*_{bb} tends to fill as more massless degrees of freedom are eliminated. Therefore, the reduction in size of the system may not be economical from a computational viewpoint. In addition, if the mass is physically lumped at the time of creating the mathematical model additional errors may be introduced.

b. Reduction of Size of System by Ritz Functions

A more general approach to the reduction of the size of the eigenvalue problem (equation (34)) is the application of the Ritz method. This technique is not restricted to a particular mass distribution--a full mass matrix does not increase the computational effort significantly. However, for systems with a limited number of masses the method can be identical to the static condensation method [15].

The method can be very accurate if some physical insight into the behavior of the structure is known. Static load patterns are selected and corresponding displacement vectors are calculated. Or

$$KR = P \tag{41}$$

The true displacements of the system are approximated by a linear combination of the discrete Ritz Vectors R. Or, the true eigenvectors are approximated by

$$\phi = RX = R_1X_1 + R_2X_2 + R_3X_3 + \ldots . R_LX_L \tag{42}$$

where L is the number of load patterns and is smaller than the size of the system N. The load patterns can be very simple; however, they must

be linearly independent. In order to produce the lower frequencies the load pattern should activate the large masses and areas of maximum flexibility. If a single unit load is used as a load pattern the method mathematically lumps the consistent mass of the system at that degree of freedom.

The reduced eigenvalue problem is produced by the substitution of equation (42) into equation (34) and premultiplication by R^T. Or,

$$K^*X - MX\Omega = 0 \tag{43}$$

where

$$K^* = R^TKR = R^TP \tag{44}$$

$$M^* = R^TMR \tag{45}$$

$$\Omega = \text{diag}\ (\omega_i^2) \tag{46}$$

Both K* and M* are full matrices because the Ritz Vectors are not orthogonal. Since all the eigenvalues and eigenvectors are required and the system is relatively small, less than 100, the Jacobi method is one of the most effective for this type of eigenvalue problem [2].

One advantage of the Ritz method for the reduction of the number of degrees of freedom for a very large structure is that it involves only a solution of a set of linear equations which may have a large number of zero terms in the stiffness matrix, equation (41). Therefore, a large number of structural elements can be used to model the basic structural behavior. The use of Ritz functions can be considered as a formal mathematical method of evaluating an approximate generalized mass matrix for the purpose of dynamic analysis.

Figure 3 illustrates the selection of static load patterns for a simple tower type structure. This structure has 6 unconstrained joints or 36 degrees of freedom. It is of interest to point out that for the

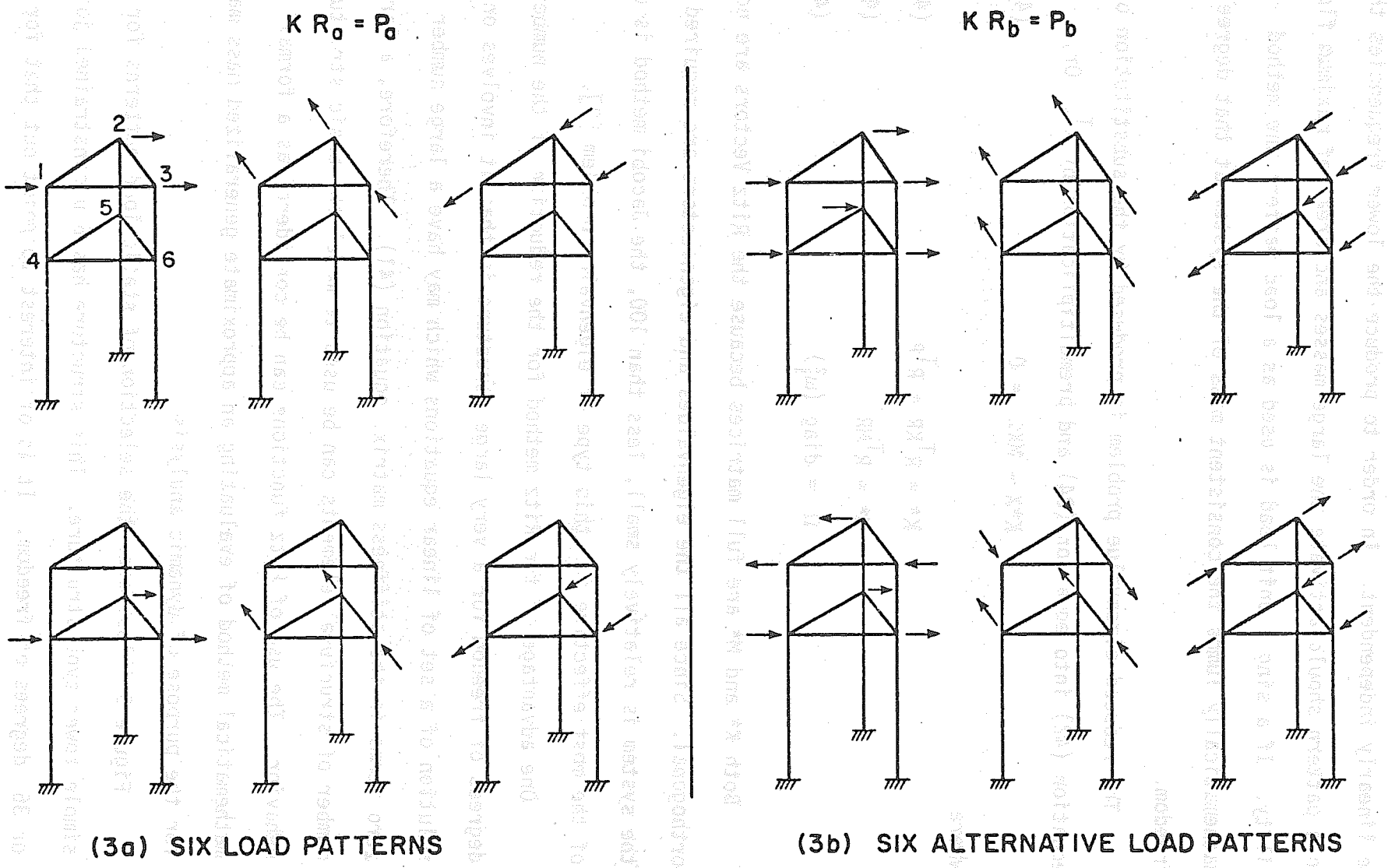


Figure 3. EXAMPLE OF SELECTION OF DIFFERENT LOAD PATTERNS FOR TOWER STRUCTURE

lateral load pattern shown the resulting displacement is complex and involves movements in all directions in addition to torsional behavior.

For a structure of this type one would expect the lateral behavior to be expressed by the first six mode shapes. Because of the geometric arrangement of the members, joints 1, 2 and 3 and joints 4, 5 and 6 will act as separate units with very little relative movement between joints. Therefore, the six possible load patterns which will capture this fundamental behavior are easily established and are shown in Figure 3a. Of course, the first six vibrational mode shapes for this structure will be composed of a linear combination of the resulting displacement patterns. Figure 3b illustrates six other load patterns which could have been used to produce identical results.

The physical modes which have been neglected by this approach are the breathing modes between joints 1, 2 and 3 and joints 4, 5 and 6. Also, the vertical vibrational modes have been omitted; but axial deformations are included in the six lateral modes.

c. Subspace Iteration

The subspace iteration method for the determination of eigenvalues and eigenvectors of very large structural systems is a significant extension of the Ritz reduction approach [16], [17] and [18]. It is the only modern computer method for very large systems (over 5000 degrees of freedom) which will converge to the exact eigenvalues.

Table 2 summarizes the subspace iteration method. The computer implementation of the method and a FORTRAN listing is given in reference [2]. From Table 2 one notes that the initial calculations are identical to the Ritz method in which the first set of load patterns must be specified. After one Ritz solution is found and the first approximation

of the first L eigenvalues of the system is calculated as

$$\phi^{(1)} = R_1 X_1 \tag{47}$$

an improved set of load patterns can be calculated from

$$P_2 = M \phi^{(1)}$$

It is assumed that P_2 is an improved estimation of the inertia forces associated with the basic mode shapes. From Table 2 it is clear that this iteration technique can be carried out to any desired degree of accuracy, assuming the method converges. The convergence of the method for various conditions is given in reference [2].

Some additional comments associated with the advantages of the subspace iteration method with respect to numerical effort are:

a) The total stiffness matrix for the system need be triangularized only once.

b) Since $K^{(k)}$ and $M^{(k)}$ tend to become diagonal as the iteration progresses the Jacobi method is very effective for this type of eigenvalue problem.

c) The size of the subspace L should be approximately 30% more vectors more than the number of accurate eigenvalues required.

d) In order to insure participation of all modes one additional random vector should be added to the load pattern set before each iteration.

e) Since the approach is basically a power method the lowest eigenvalues converge faster and are more accurate.

f) For most structures a high degree of accuracy is not required for the highest modes, because of their low participation in the dynamic response. Therefore, the subspace iteration produces practical results with respect to required accuracy.

TABLE 2. SUMMARY OF THE SUBSPACE ITERATION ALGORITHM FOR SOLUTION OF LARGE EIGENVALUE PROBLEM $K\phi_n = \omega_n^2 M\phi_n$

A. INITIAL CALCULATION:

1. Form Stiffness Matrix K and Mass Matrix M.

2. Triangularize K:

$$K = L\,D\,L^T \qquad (N \times N)$$

3. Specify Initial Load Patterns:

$$P_1 = N \times L \text{ matrix} \qquad \text{where} \qquad L \ll N$$

B. FOR EACH ITERATION, k = 1, 2, 3, ... :

1. Solve for Ritz Vectors R_k: $(N \times L)$

$$L\,D\,L^T\,R_k = P_k$$

2. Calculate Generalized Stiffness in Subspace:

$$K^{(k)} = R_k^T\,K\,R_k = R_k^T\,P_k \qquad (L \times L)$$

3. Calculate Generalized Mass in Subspace:

$$M^{(k)} = R_k^T\,M\,R_k \qquad (L \times L)$$

4. Solve Eigenvalue Problem in Subspace:

$$K^{(k)}X_k = M^{(k)}X_k\Omega^{(k)} \qquad (L \times L)$$

5. Calculate Improved Approximate Eigenvectors:

$$\phi^{(k)} = R_k\,X_k$$

6. Check for Convergence:

$$\Omega^{(k)} \to \text{diag.}\ (\omega_n^2) \quad \text{and} \quad \phi^{(k)} \to \phi \text{ as } k \to \infty$$

stop if converged - perform Sturm sequence check

7. Calculate Improved Load Patterns for Next Iteration:

$$P_{k+1} = M\phi^{(k)} \qquad (N \times L)$$

8. Return to Step B-1.

d. Additional Numerical Techniques for Eigenvalue Problems

In a general computer program for the calculation of eigenvalues and eigenvectors several different numerical techniques may be useful in the solution strategy in order to improve the convergence and to minimize numerical effort.

Inverse Iteration

If only one load pattern is used in the iteration procedure given in Table 2 the subspace iteration approach is the standard inverse iteration method and will converge to the lowest eigenvalue. For this case the eigenvalue problem in the subspace is trivial. Or

$$\lambda_1 = \omega_1^2 \simeq K^{(k)}/M^{(k)} \tag{49}$$

which is better known as the Rayleigh quotient [6]. After the first eigenpair λ_1 and ϕ_1 are determined inverse iteration can be used to calculate accurately additional eigenpairs if the techniques of shifting, determinant search, deflation and Sturm sequence checking are introduced.

Determinant Search

This numerical technique can be explained by considering the following equation:

$$[K - \lambda M] X = 0 \tag{50}$$

For any numerical value of λ_k the following triangularization is possible

$$[K - \lambda_k M] = L_k D_k L_k^T$$

The determinant of this matrix for λ_k is

$$\det (K - \lambda_k M) = D_{11} D_{22} D_{33} \ldots D_{NN}$$

where N is the order of the matrix.

If the numerical value of the determinant is evaluated for a large number of different values of λ a function can be generated of the form shown in Figure (4). This function, $p(\lambda)$, which in this case is evaluated numerically, is a plot of the characteristic polynomial. λ_1, λ_2, λ_3, λ_4, ... λ_N, are the eigenvalues of the system and correspond to zero values of the det $(K - \lambda M)$.

Deflation

A numerical plot of a polynomial with the first root suppressed, or deflated, can be computed and would have the approximate shape as shown in Figure 5. The only reason for numerically evaluating $p_1(\mu_k)$ and $p_1(\mu_{k+1})$ is to obtain a better estimation for λ_s from the extrapolation equation

$$\lambda_s = \mu_k + \frac{\mu_k - \mu_{k+1}}{p_1(\mu_{k+1}) - p_1(\mu_k)} \; p_1(\mu_k) \tag{53}$$

It is this type of strategy which is used to obtain a value of λ_s which is close to a desired root. For this case two triangularizations are required in order to evaluate λ_s. Therefore, this technique is effective for matrices with small band widths which can be triangularized with a minimum of numerical effort [19].

Shifting

Inverse iteration converges to the numerically smallest eigenvalue. In order for the method to be used for other eigenvalues the following change of variable can be introduced

$$\lambda = \lambda_s + \rho \tag{54}$$

Therefore, equation (50) can be written as

$$[K_s - \rho M] X = 0 \tag{55}$$

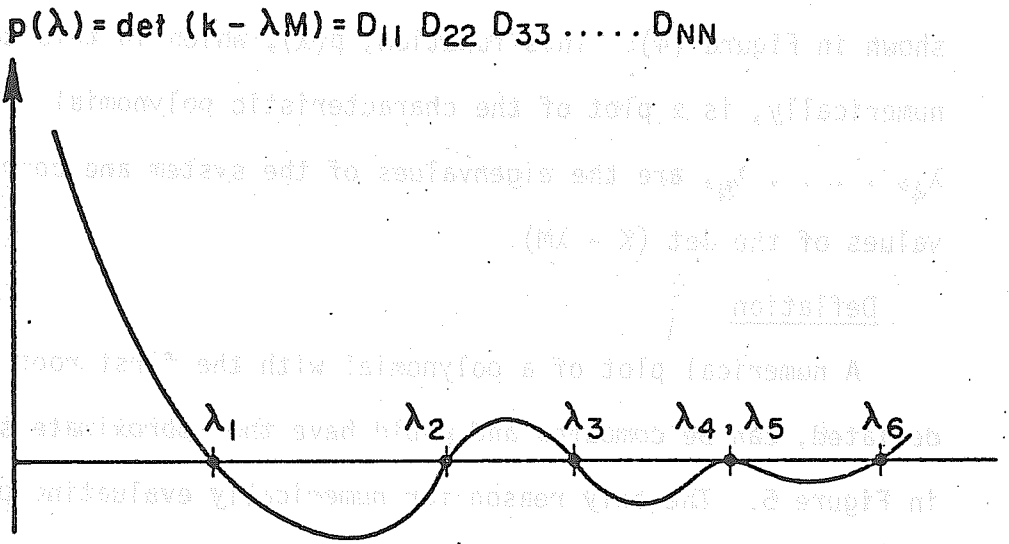


Figure 4. Characteristic Polynomial [K - λM]

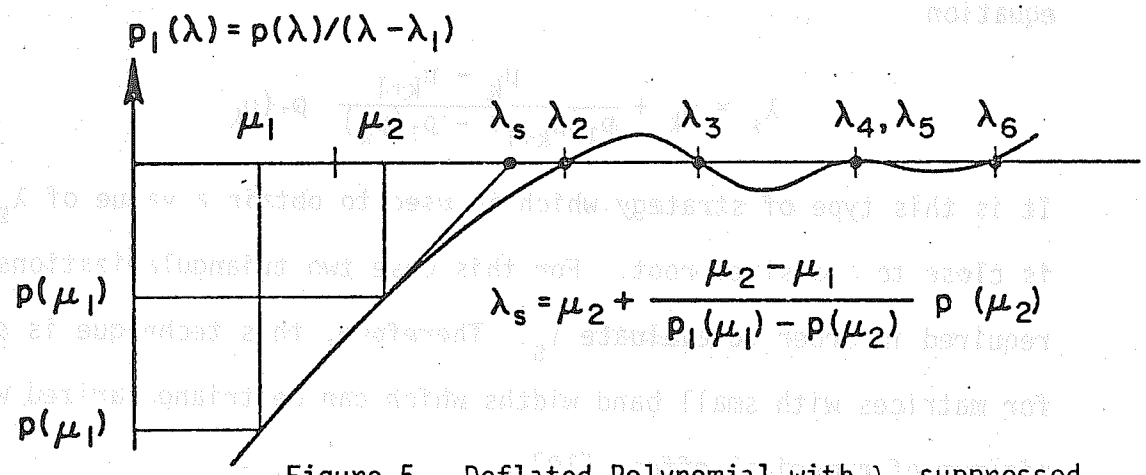


Figure 5. Deflated Polynomial with λ_1 suppressed

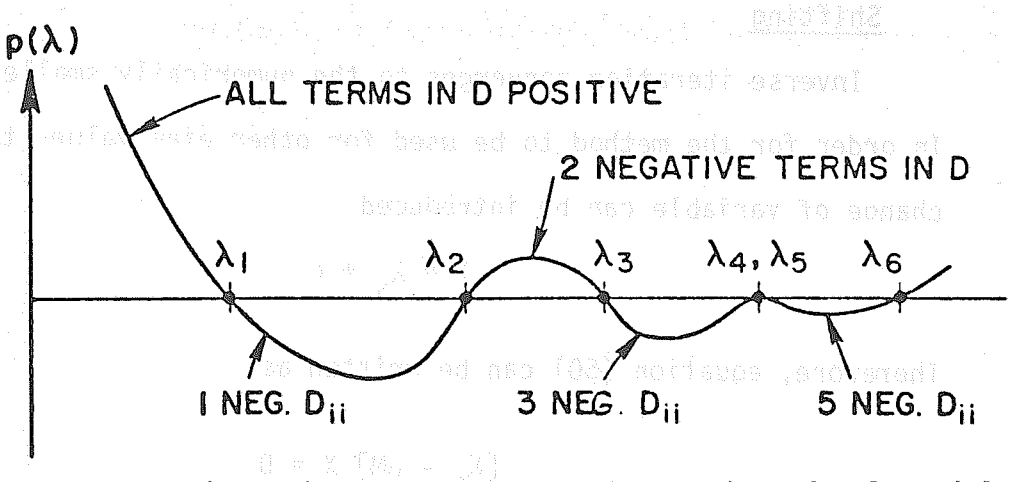


Figure 6. Sturm Sequence Properties of Polynomial

where

$$K_s = K - \lambda_s M \tag{56}$$

If λ_s is closer to λ_2 than to λ_1 the inverse iteration method when applied to equation (55) will converge to λ_2. This follows from the standard power method proof [1] [2].

Sturm Sequence Check

One of the potentially serious problems which can develop in the numerical evaluation of frequencies and mode shapes in a practical dynamic analysis is if important frequencies are neglected. The Sturm Sequence Check is a method which allows the engineer to verify the results of an eigenvalue problem. One can prove the Sturm Sequence Theorem as presented here from a direct examination of the complete family of deflated polynomials, $p(\lambda)$, $p_1(\lambda)$, $p_2(\lambda)$, ... One notes that they are all derived from the basic sequence for a given value of λ, or

$$\det (K - \lambda M) = D_{11}\ D_{22}\ D_{33}\ .\ .\ .\ .\ D_{NN} \tag{57}$$

For a given value of λ the basic properties of this sequence of numbers are illustrated in Figure 6. It is apparent that the properties of this sequence of numbers can be used as a powerful technique in the numerical strategy of evaluating eigenvalues. If lowest eigenvalues are evaluated by any method the Sturm sequence can be calculated for

$$\lambda = 1.001\ \lambda_n \tag{58}$$

If all eigenvalues have been calculated the Sturm sequence should have n negative terms [2]. This assumes a desired accuracy of .001.

Another important application of the Sturm Sequence Technique is to evaluate the number of frequencies in a certain frequency range--say

between λ_L and λ_H. Therefore, LDL^T triangularizations of $[K - \lambda_H M]$ and $[K - \lambda_L M]$ will indicate the number of eigenvalues below each value. The difference will be the number of values within the range. In order to calculate the eigenpairs in the range one can shift into the range and use inverse iteration or subspace iteration to evaluate only the values of interest.

5. Transformation To Uncoupled Modal Equations

The basic numerical properties of the undamped free vibration mode shapes are

$$M^* = \phi^T M \phi = I \tag{59}$$

$$K^* = \phi^T K \phi = \text{diag}\ (\bar{\omega}_n^2) \tag{60}$$

In which the mode shapes have been normalized so the generalized mass is one. Or

$$\phi_n^T M \phi_n = 1 \tag{61}$$

The following transformation, change of variable, is introduced into the basic equilibrium equations.

$$U(t) = \sum_{n=1}^{M} \phi_n X_n(t) = \phi X(t) \tag{62}$$

Therefore, the velocities and accelerations are

$$\dot{U}(t) = \phi \dot{X}(t) \tag{63}$$

$$\ddot{U}(t) = \phi \ddot{X}(t) \tag{64}$$

If equations (62), (63) and (64) are substituted into equation (2) and the resulting matrix equation premultiplied by ϕ^T we obtain

$$M^* \ddot{X}(t) + C^* \dot{X}(t) + K^* X(t) = P(t) \tag{65}$$

The matrices M* and K* are diagonal; however, C* is not diagonal unless an assumption is made on the basic form of viscous damping which exists in the structure. Since damping is normally small and is difficult to physically model and identify the following assumption is normally made:

$$C^* \simeq \text{diag}\ (2\xi_n \bar{\omega}_n) \tag{66}$$

where ξ_n is the ratio of damping in mode n to the critical damping for the mode. With this assumption of uncoupled modal damping the typical modal equation can be written as

$$\ddot{X}_n(t) + 2\xi_n\ \bar{\omega}_n\ X_n(t) + \bar{\omega}_n^2\ X_n(t) = p_n(t) = c_n\ f(t) \tag{67}$$

After the modal equations are evaluated the time dependent displacements are calculated from equation (62).

The same technique can be used to uncouple equation (32) in order to avoid the solution of a large number of complex linear equations. For this case the following transformation is introduced

$$Y(\omega) = \phi Z(\omega) \tag{68}$$

The substitution of equation (68) into equation (32) yields a typical modal equation in the frequency domain.

$$[\bar{\omega}_n^2 + 2i\omega\ \bar{\omega}_n \xi_n - \omega^2]\ Z_n(\omega) = \phi_n\ \bar{F}(\omega) \tag{69}$$

For structures which are formulated in the frequency domain and their behavior can be represented by a limited number of natural frequencies this is numerically the most efficient approach.

6. Solution of Modal Equations

The solution to the single-degree of freedom modal equation given by equation (67) can be accomplished by one of several methods. For certain loading which can be expressed as an analytic function exact mathematical solutions are possible [1].

a. Direct Step-by-Step Solution

The most direct approach to the solution of this second order ordinary differential equation is to use a numerical finite difference method. The same techniques are possible which were used for the coupled equation. The step-by-step solution method given in Table 1 is numerically very efficient when applied to equation (67).

b. Duhamel Integral

In the case of arbitrary loading it is very common to express the solution in the form of the Duhamel Integral [1]. The Duhamel Integral is then numerically integrated. Since this numerical integration approach involves many numerical evaluations of trigonometric and exponential functions, which require series expansions within a digital computer, the method cannot be considered a good numerical method. Also, for high frequencies a very small integration interval is required for accuracy.

c. Transformation To The Frequency Domain

The single degree of freedom modal equations can be transformed into the frequency domain. The Fourier integrals and transforms can be numerically evaluated by the Fast Fourier transform technique. This, of course, is identical to the approach suggested by equation (69). This method introduces the same types of errors which are normally associated with the approximation of a function by a Fourier series or integrals.

d. Piecewise Exact Method

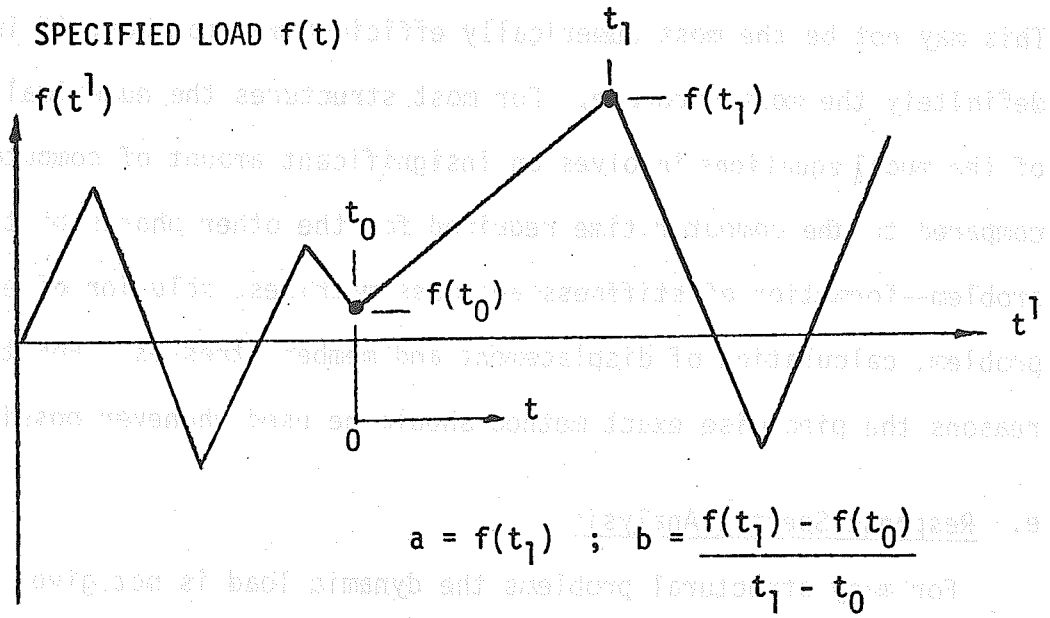
Many types of loading can be represented by a series of straight lines between unequal time intervals. Most earthquake ground acceleration data is in this form. An exact mathematical solution is possible for a

Table 3. PIECEWISE EXACT SOLUTION METHOD

BASIC EQUATION:

$$\ddot{X} + 2\xi\omega\dot{X} + \omega^2 X = f(t) = a + bt$$

SPECIFIED LOAD f(t)



$$a = f(t_1) \quad ; \quad b = \frac{f(t_1) - f(t_0)}{t_1 - t_0}$$

EXACT DISPLACEMENT SOLUTION:

$$X(t) = A_0 + A_1\, t + A_2\, e^{-\xi\omega t} \cos \omega^* t + A_3\, e^{-\xi\omega t} \sin \omega^* t$$

where $A_0 = \frac{a}{\omega^2} - 2\,\frac{\xi b}{\omega^3}$

$A_1 = \frac{b}{\omega^2}$

$A_2 = X(t_o) - A_0$

$A_3 = \frac{1}{\omega^*}(\dot{X}(t_0) + \xi\,\omega\,A_2 - A_1)$

EXACT SOLUTION FOR VELOCITY:

$$\dot{X}(t) = A_1 + (\omega^* A_3 - \xi\omega A_2)\, e^{-\xi\omega t} \cos \omega^* t$$

$$- (\omega^* A_2 + \xi\omega A_3)\, e^{-\xi\omega t} \sin \omega^* t$$

straight line loading subjected to displacement and velocity initial conditions. Therefore, exact numerical values at any convenient time interval can be calculated. Table 3 summarizes the necessary equations for this approach for the numerical evaluation of the modal equations. This may not be the most numerically efficient method; but, it is definitely the most accurate. For most structures the numerical solution of the modal equations involves an insignificant amount of computer time compared to the computer time required for the other phases of the problem--formation of stiffness and mass matrices, solution of eigenvalue problem, calculation of displacement and member stresses. For these reasons the piecewise exact method should be used whenever possible.

e. Response Spectra Analysis

For many structural problems the dynamic load is not given in terms of a time dependent function; but, the load is specified as a response spectra. By definition a response spectra is a plot of maximum values of displacement response v(max) obtained from the solution of the following equation for various values of ω.

$$\ddot{v}(t) + 2\omega\xi\dot{v}(t) + \omega^2 v(t) = f(t) \tag{70}$$

A typical plot of the maximum, v(max), for specified ω and damping ratios ξ is shown in Figure 1.

The typical modal equation is of the form

$$\ddot{X}_n + 2\bar{\omega}_n\xi_n\dot{X}_n + \bar{\omega}_n^2 X_n = \phi_{11} F(t) = c_n f(t) \tag{71}$$

Therefore, the maximum modal response can be calculated from

$$X_n(\max) = c_n v_n(\max) \tag{72}$$

where $v_n(\max)$ is the value obtained from Figure 1 for values of ω_n and ξ_n.

After the maximum response in each modal equation is evaluated the maximum displacement in each modeshape for the complete structure is given by

$$U(max)_n = \phi_n X_n(max) \tag{73}$$

For any particular degree of freedom a probabilistic value of displacement may be calculated from

$$u = \sqrt{u_1^2 + u_2^2 + U_3^2 + \ldots . u_m^2} \tag{74}$$

Other methods exist for adding maximum modal response; however, the square root of the sum of the squares of the maximum modal values is one of the most common.

In order to evaluate member stresses it is first necessary to evaluate the member stresses due to each of the maximum modal responses.. Or

$$\sigma(max) = T\ U(max)_n \tag{75}$$

where T is a stress-displacement transformation matrix for the member. The probabilistic member stress is estimated by

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \ldots . \sigma_M^2} \tag{76}$$

Note that the probabilistic member stress cannot be calculated from the probabilistic displacements.

NONLINEAR ANALYSIS

For complex structures several different types of nonlinear behavior for both static and dynamic loads are possible. Nonlinear behavior implies that the displacements and stresses produced by the different load conditions cannot be directly added; or, the basic principle of superposition does not hold. Because there are so many different types of nonlinearities there is not one general method which can be applied to all problems.

Large structures for which their weight is significant may have a dead load stress distribution which is highly dependent on the method of construction and installation sequence. The correct theoretical method of evaluating these stresses is to perform a complete analysis at each stage of construction with the internal stresses from the analysis of the previous stage used as initial conditions for the new analysis. This analysis technique can be used for the evaluation of installation stresses also. For subsequent analysis of the structure in which additional nonlinear stresses are developed due to static or dynamic loads it may be extremely important to start the analysis with an accurate estimation of the initial stress conditions.

Perhaps the most common type of nonlinear behavior is due to nonlinear materials. Most structural design requires that the structural materials remain in the elastic range during the design loads. However, foundation stresses in soil may be nonlinear under low stress levels. Under dynamic loads soil nonlinearities are often approximated by an effective damping factor to be used in a linear dynamic analysis. Also, during earthquake or other load conditions some nonlinear material behavior can be tolerated without the collapse of the structure.

One of the most unlikely types of nonlinear behaviors to expect in a practical structure is the existence of large strains which require an alternate definition of stress. This type of nonlinearity exists in only rubber like materials.

For tall structures or structures supported by cables large displacements may exist under design loading. For this case it is extremely important that the static or dynamic equilibrium equations are satisfied in the deformed geometry.

For fixed offshore structures where the velocities of the structure are comparable to the water partical velocities, the wave forces are nonlinear and may be expressed as

$$F(t) = F(U(t), \dot{U}(t)) \tag{77}$$

This type of nonlinear behavior can be considered by the linear step-by-step methods. Based on the previous increment the velocity of the structure can be predicted from

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta t\, \ddot{U}_t \tag{78}$$

and a good estimate of the nonlinear drag forces can be estimated. Since the properties of the structures do not change the effective stiffness matrix need not be modified or triangularized at each time increment. Therefore, the method given in Table 1 can be used for this form of nonlinearity.

A general formulation for the nonlinear analysis of a structural system can be developed if equation (1) is rewritten at time

$$(F_t^i + \Delta F_t^i) + (F_t^d + \Delta F_t^d) + (F_t^s + \Delta F_t^s) = F_{t+\Delta t} \tag{79}$$

in which the force changes are given by

$$\Delta F_t^i = M_t\, \Delta\ddot{U}_t \quad , \quad \Delta F_t^d = C_t\, \Delta\dot{U}_t \quad , \quad \Delta F_t^s = K_t U_t \tag{80}$$

where M_t, C_t and K_t are the approximate mass, damping and stiffness matrices at time t. Therefore, equation (79) can be rewritten as

$$M_t \ddot{\Delta U} + C_t \dot{\Delta U}_t + K_t \Delta U_t = F^* \tag{81}$$

where

$$F^* = F_{t+\Delta t} - F_t^i - F_t^d - F_t^s \tag{82}$$

A direct step-by-step method, as presented for linear systems, can be used for the evaluation of $\ddot{\Delta U}_t$, $\dot{\Delta U}_t$ and ΔU_t. Because the matrices M_t, C_t and K_t can only be approximated over the time interval it is recommended that the forces F_t^i, F_t^d and F_t^s be recomputed at the end of the time increment. For example, the structural forces F_t^s which are consistent with U_t should be evaluated as follows:

1. From the displacement U_t calculate in member m the strain ε_m.
2. From the specified nonlinear stress strain behavior calculate the stress τ_m.
3. Using virtual work calculate the structural forces acting on element

$$F_m = \int A_m^T \tau_m \, dV_m$$

where A_m is the strain-displacement transformation matrix for member m.

4. Calculate the total structural forces at time t from

$$F_t^s = \Sigma F_m$$

For structures with slight nonlinearities this type of analysis may be accurate. However, for very nonlinear behavior it may be necessary to iterate within a time step in order to minimize the accumulation of errors. For a more complete discussion of dynamic nonlinear analysis methods the reader is referred to references [12], [21], [22], [23], [24].

FINAL REMARKS

Several different numerical methods for the dynamic analysis of linear structural systems have been presented. Many of these methods have been incorporated into general purpose programs and have been successfully used in the solution of complex structural systems [24], [25], [26]. General purpose programs for nonlinear analysis have been developed based on the techniques presented [22], [24].

The computer program CAL has a limited number of operations at the present time. It's present purpose is to solve small linear systems which are associated with courses on the static and dynamic analysis of structures. It is possible to expand CAL into a general purpose program capable of effectively solving very large problems. Such an extension will depend on the rate of development and economy of minicomputer systems.

REFERENCES

1. R. W. Clough and J. Penzien, Dynamics of Structures, McGraw-Hill Book Company, New York, NY, 1975.

2. K. J. Bathe and E. L. Wilson Numerical Methods in Finite Element Analysis, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.

3. J. M. Biggs, Introduction to Structural Dynamics, McGraw-Hill Book Company, New York, NY, 1964.

4. W. C. Hurty and M. F. Rubinstein, Dynamics of Structures, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1964.

5. L. Collatz, The Numerical Treatment of Differential Equations, Springer-Verlag, New York, NY, 1966, p. 116.

6. S. H. Crandall, Engineering Analysis, McGraw-Hill Book Company, New York, NY, 1956.

7. C. E. Forberg, Introduction to Numerical Analysis, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1969.

8. N. M. Newmark, "A Method of Computation for Structural Dynamics," ASCE, Journal of Engineering Mechanics Division, Vol. 85, 1959, p. 67-94.

9. J. C. Houbolt, "A Recurrence Matrix Solution for the Dynamic Response Elastic Aircraft," Journal of Aeronautical Science, Vol. 17, 1959, pp. 540-550.

10. E. L. Wilson, "A Computer Program for the Dynamic Stress Analysis of Underground Structures," Report UC SESM 68-1, Department of Civil Engineering, University of California, Berkeley, 1968.

11. E. L. Wilson, "Elastic Dynamic Response of Axisymmetric Structures," Report UC SESM 69-2, Department of Civil Engineering, University of California, Berkeley, 1969.

12. E. L. Wilson, I. Farhoomand, and K. J. Bathe, "Nonlinear Dynamic Analysis of Complex Structures," International Journal of Earthquake Engineering and Structural Dynamics, Vol. 1, 1973, pp. 241-252.

13. K. J. Bathe and E. L. Wilson, "Stability and Accuracy Analysis of Direct Integration Methods," International Journal of Earthquake Engineering and Structural Dynamics, Vol. 1, 1973, pp. 283-291.

14. E. L. Wilson, "The Static Condensation Algorithm," International Journal of Numerical Methods in Engineering, Vol. 8, 1974, pp. 199-203.

15. R. P. Shubinski, E. L. Wilson, and L. G. Selna, "Dynamic Response of Deepwater Structures," Civil Engineering in the Oceans, ASCE Conference, San Francisco, 1966.

16. K. J. Bathe, "Solution Methods of Large Generalized Eigenvalue Problems in Structural Engineering," Report UC SESM 71-20, Civil Engineering Department, University of California, Berkeley, 1971.

17. K. J. Bathe and E. L. Wilson, "Large Eigenvalue Problems in Dynamic Analysis," ASCE, Journal of Engineering Mechanics Division, Vol. 98, 1972, pp. 1471-1485.

18. K. J. Bathe and E. L. Wilson, "Eigensolution of Large Structural Systems with Small Bandwidth," ASCE, Journal of Engineering Mechanics Division, Vol. 99, 1973, pp. 467-479.

19. K. J. Bathe and E. L. Wilson, "Solution Methods for Eigenvalue Problems in Structural Mechanics," International Journal for Numerical Methods in Engineering, Vol. 6, 1973, pp. 213-226.

20. A. K. Chopra, "Earthquake Analysis of Complex Structures," Proceedings ASME Conference, Applied Mechanics in Earthquake Engineering, November 1974, New York,

21. C. A. Felippa, "Procedures for Computer Analysis of Large Nonlinear Structural Systems," International Symposium on Large Engineering Systems, University of Manitoba, Winnipeg, Canada, August 9-12, 1976.

22. K. J. Bathe, H. Ozdemir, and E. L. Wilson, "Static and Dynamic Geometric and Material Nonlinear Analysis," Report UC SESM 74-4, Department of Civil Engineering, University of California, 1974.

23. K. J. Bathe, "An Assessment of Current Finite Element Analysis of Nonlinear Problems in Solid Mechanics," Proceedings Symposium on the Numerical Solution of Partial Differential Equations, May 1975, Academic Press, Inc., 1976.

24. K. J. Bathe, "ADINA--A Finite Element Program for Automatic Dynamic Incremental Nonlinear Analysis," Report 82448-1, Acoustics and Vibration Laboratory, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., 1975.

25. K. J. Bathe, E. L. Wilson, and F. E. Peterson, "SAP IV--A Structural Analysis Program for Static and Dynamic Response of Linear Systems," Report EERC 73-11, College of Engineering, University of California, Berkeley, June 1973, revised April 1974.

26. EAC/EASE 2 Dynamics--User Information Manual/Theoretical, Control Data Corporation

APPENDIX A Example Problems

1. Solution of Nonsymmetrical Set of Equations

The symmetrical equation operation SOLVE can be used to solve the set of equations AX = B where A is not a symmetrical matrix. The set of equations is premultiplied by A^T which results in a modified symmetrical set of equations A*X = B* where A* = A^TA and B* = A^TB. The CAL operations to complete this series of matrix operations are listed on the attached page. It is of interest to note that after the first blank on a CAL operation card user comments can be used to clarify the language. After each CAL operation is executed the computer time required for the operation is printed. It should be noted that it requires a fixed amount of time to read and print information associated with each operation. For the CDC 6400 this is approximately .03 seconds; therefore, this tends to be the dominant factor for small problems.

2. Example of LOOP Operation - Inverse Iteration

Table 2, page 44, is a summary of the subspace iteration method for the evaluation of the mode shapes and frequencies of structural systems. The inverse iteration method is essentially the same approach with only one vector used in the iteration. The basic iterative equation can be written in the form

$$K\phi^{(s+1)} = \lambda^{(s)} M\phi^{(s)} = R^{(s)}$$

in which the set of equations is solved for $\phi^{(s+1)}$. The approximation of the eigenvalue is calculated from

$$\lambda^{(s+1)} = \frac{\phi^{(s+1)^T} K\phi^{(s+1)}}{\phi^{(s+1)^T} M\phi^{(s+1)}} = \frac{\phi^{(s+1)} R^{(s)}}{\phi^{(s+1)^T} M\phi^{(s+1)}}$$

EXAMPLE PROBLEM - SOLUTION OF SET OF LINEAR EQUATIONS

a. Given $A X = B$

where:

$$A = \begin{bmatrix} 1 & 2 & 4 & -2 \\ 1 & 3 & 2 & 1 \\ 3 & 1 & 3 & 1 \\ 1 & 3 & 4 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 10 & 6 \\ 17 & 7 \\ 18 & 8 \\ 27 & 10 \end{bmatrix}$$

b. CAL input data

```
START  EXAMPLE 1 - SOLUTION OF NONSYMMETRICAL S
LOAD,A,4,4,1
  (4F2.0)
 2 2 4-2
 1 3 2 1
 3 1 3 1
 1 3 4 2
PRINT,A
LOAD,B,4,2,1
 (2F5.0)
 10.   6.
 17.   7.
 18.   8.
 27.   10.
TRAN,A,AT
MULT,AT,A,ATA
MULT,AT,B,X
SOLVE,ATA,X
PRINT,X    SOLUTION X
MULT,A,X,E
SUB,E,B
PRINT,E  ERROR IN SOLUTION  E = AX - B
STOP
```

c. Output from CAL

```
**START  EXAMPLE 1 - SOLUTION OF NONSYMMETRICAL SET OF EQUATIONS
------------------------------  .028 SECONDS
**LOAD,A,4,4,1
   4 ROWS      4 COLUMNS
------------------------------  .041 SECONDS
**PRINT,A
                    1                2                3                4
      1   .2000000E+01    .2000000E+01    .4000000E+01   -.2000000E+01
      2   .1000000E+01    .3000000E+01    .2000000E+01    .1000000E+01
      3   .3000000E+01    .1000000E+01    .3000000E+01    .1000000E+01
      4   .1000000E+01    .3000000E+01    .4000000E+01    .2000000E+01
------------------------------  .039 SECONDS
**LOAD,B,4,2,1
   4 ROWS      2 COLUMNS
------------------------------  .039 SECONDS
**TRAN,A,AT
   4 ROWS      4 COLUMNS
------------------------------  .030 SECONDS
**MULT,AT,A,ATA
   4 ROWS      4 COLUMNS
------------------------------  .031 SECONDS
**MULT,AT,B,X
   4 ROWS      2 COLUMNS
------------------------------  .028 SECONDS
**SOLVE,ATA,X
------------------------------  .029 SECONDS
**PRINT,X    SOLUTION X
                    1                2
      1   .1000000E+01    .1000000E+01
      2   .2000000E+01    .1000000E+01
      3   .3000000E+01    .1000000E+01
      4   .4000000E+01    .1000000E+01
------------------------------  .035 SECONDS
**MULT,A,X,E
   4 ROWS      2 COLUMNS
------------------------------  .030 SECONDS
**SUB,E,B
------------------------------  .026 SECONDS
**PRINT,E  ERROR IN SOLUTION  E = AX - B
                    1                2
      1   0.              -.5684342E-13
      2   -.7958079E-12   -.1705303E-12
      3   -.1136868E-12   -.5684342E-13
      4   .3410605E-12    0.
------------------------------  .034 SECONDS
**STOP
```

LOOP EXAMPLE - INVERSE INTERATION

Problem to be solved

$$K\phi = \lambda M\phi$$

where

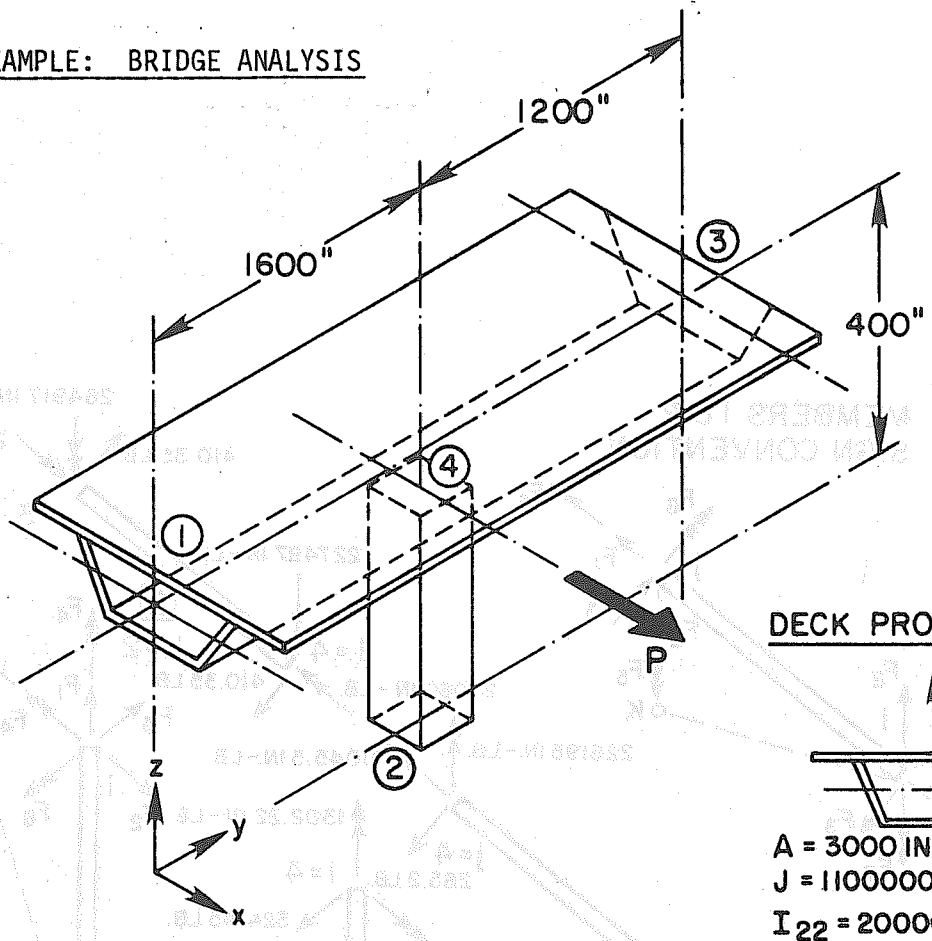
$$K = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The CAL operations are

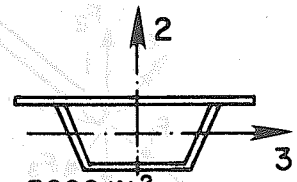
```
START          EXAMPLE OF INVERSE ITERATION
LOAD,K,4,4,1
  (4F2.0)
 2-1 0 0
-1 2-1 0
 0-1 2-1
 0 0-1 1
PRINT,K        4 x 4 STIFFNESS MATRIX
LOAD,M,4,4,1
  (4F1.0)
0000
0200
0000
0001
PRINT,M        4 x 4 MASS MATRIX
LOAD,P,4,1,1
  (4F1.0)
1111
PRINT,P        4 x 1 STARTING VECTOR
LOAD,L,1,1
 1.0
PRINT,L        STARTING EIGENVALUE
SOLVE,K,1      LDLT TRIANGULARIZATION
LOOP,10
MULT,M,P,F
SCALE,F,L
DUP,F,P
SOLVE,K,P,2    SOLVE FOR NEW EIGENVECTOR
TRAN,P,PT
MULT,PT,F,N
MULT,PT,M,PTM
MULT,PTM,P,D
INVEL,D
MULT,D,N,L
YES
PRINT,L        NEW EIGENVALUE
NO
NEXT
YES
PRINT,P        FINAL EIGENVECTOR
STOP
NEXT
LOOP
STOP
```

The attached list of CAL operations indicates how LOOP and NEXT operations can be used to solve the inverse iteration problem. The limits of the LOOP have been set to 10; however, a preferable approach would have been to specify a tolerance on the eigenvalue and to terminate the calculation with the NEXT,TOL operation. Where TOL would be a 1 x 1 matrix which is evaluated during each pass through the loop. This example is discussed in more detail in reference [2], page 421. Because the CAL output is relatively large for this problem it is not shown. The use of the YES and NO operations, as illustrated in this problem, will tend to reduce repetitive output which can be generated with the loop operations.

EXAMPLE: BRIDGE ANALYSIS

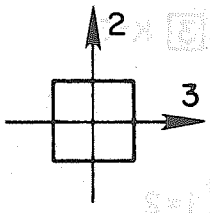


DECK PROPERTIES



A = 3000 IN.^{2}
J = 1100000 IN.^{4}
I_{22} = 2000000 IN.^{4}
I_{33} = 700000 IN.^{4}
E = 3000000 PSI.
G = 1200000 PSI.
m = 0.675 LB.-SEC.^{2}/IN.

COLUMN PROPERTIES



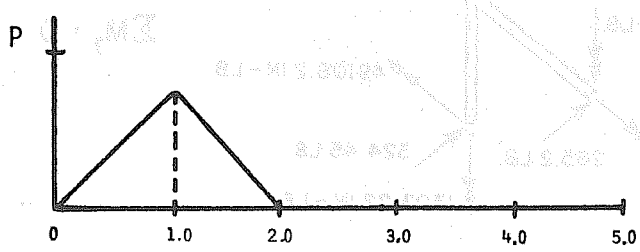
A = 1000 IN.^{2}
J = 116000 IN.^{4}
$I_{22} = I_{33}$ = 82500 IN.^{4}
E = 3000000 PSI.
G = 1200000 PSI.
m = 0.225 LB.-SEC.^{2}/IN.

PART A: STATIC

Solve for the static displacement and member forces due to a 1K force at joint 4 in the X direction. Check 3D static.

PART B: DYNAMIC

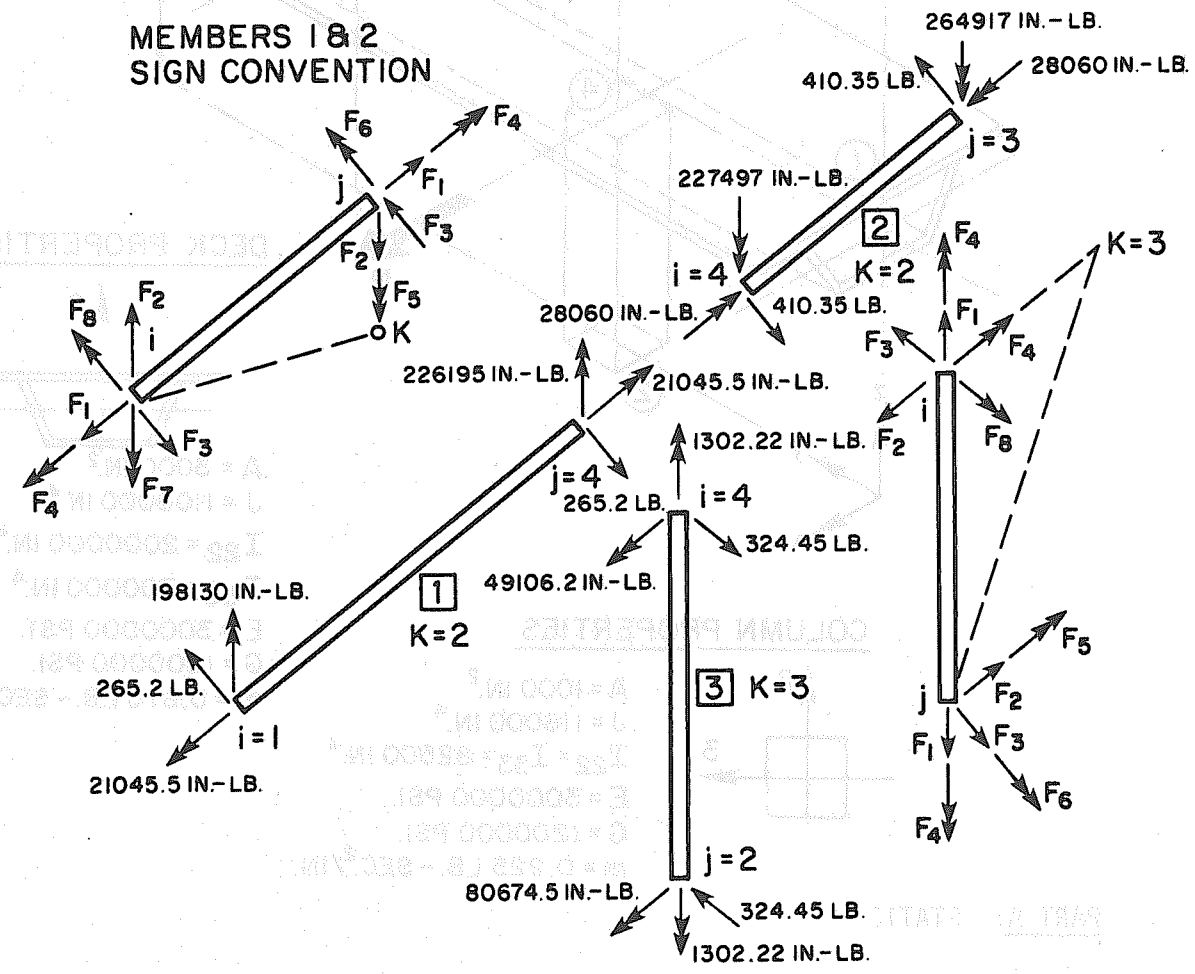
B.1. Use the STEP operation to solve for the time dependent load shown below.



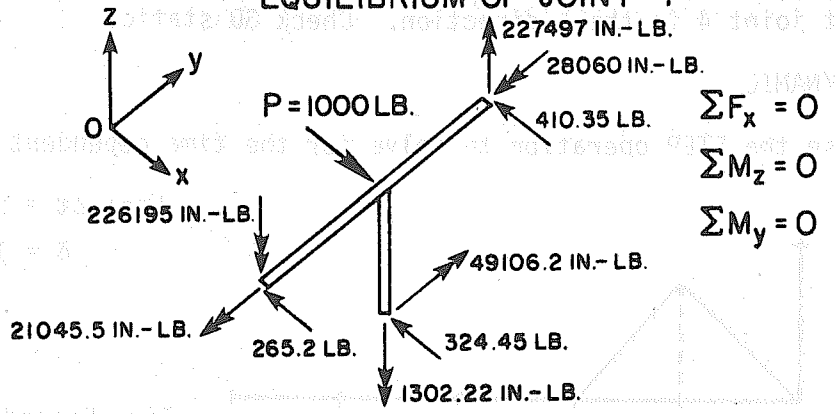
Use: $\Delta t = 0.05$ $\alpha = 1/6$
$\delta = 1/2$ $\theta = 1.42$

Use a total of 100 steps, and use the PLOT operation to plot the X displacement at joint 4.

B.2. Use the EIGEN operation to evaluate all mode shapes and frequencies.



EQUILIBRIUM OF JOINT 4



$\Sigma F_x = 0$

$\Sigma M_z = 0$

$\Sigma M_y = 0$

PART A - INPUT

```
START
LABEL,3
                                  ********************************
                                  ** STATIC ANALYSIS EXAMPLE    **
                                  ********************************
NODES,JOI,4                       ** GEOMETRY OF THE STRUCTURE **
    1              0.          0.          400.
    2              0.       1600.0           0.
    3              0.       2800.0        400.
    4              0.       1600.0        400.

PRINT,JOI                         ** NODE NUMBERS AND COORDINATES **
BOUND,BC                          ** NON ZERO DISPLACEMENTS **
    4    0    1    1    1    1    1    1

LOAD,BPR,2,7,1                    ** BEAM PROPERTIES  **
  (7E10.0)
    3.0E03     1.1E06     2.0E06     7.0E05     3.0E06     1.2E06   0.675E00
    1.0E03    11.6E04    82.5E03    82.5E03     3.0E06     1.2E06   0.225E00
PRINT,BPR
BEAM,BRIDGE,JOI,BC,BPR
    1    1    4    2    1
    2    4    3    2    1
    3    4    2    3    2

ADDSF,K,MAS                       ** FORM STIFFNESS AND MASS MATRICES **
PRINT,K                           ** STIFFNESS MATRIX **
LOADS,LDS,BC,1                    ** LOAD VECTOR **
    4    1    1000.0        0.0         0.0         0.0         0.0         0.0

SOLVE,K,LDS
DISPL,LDS,BC                      ** DISPLACEMENTS **
FORCE,BRIDGE,LDS                  ** MEMBER FORCES **
STOP
```

PART A - OUTPUT

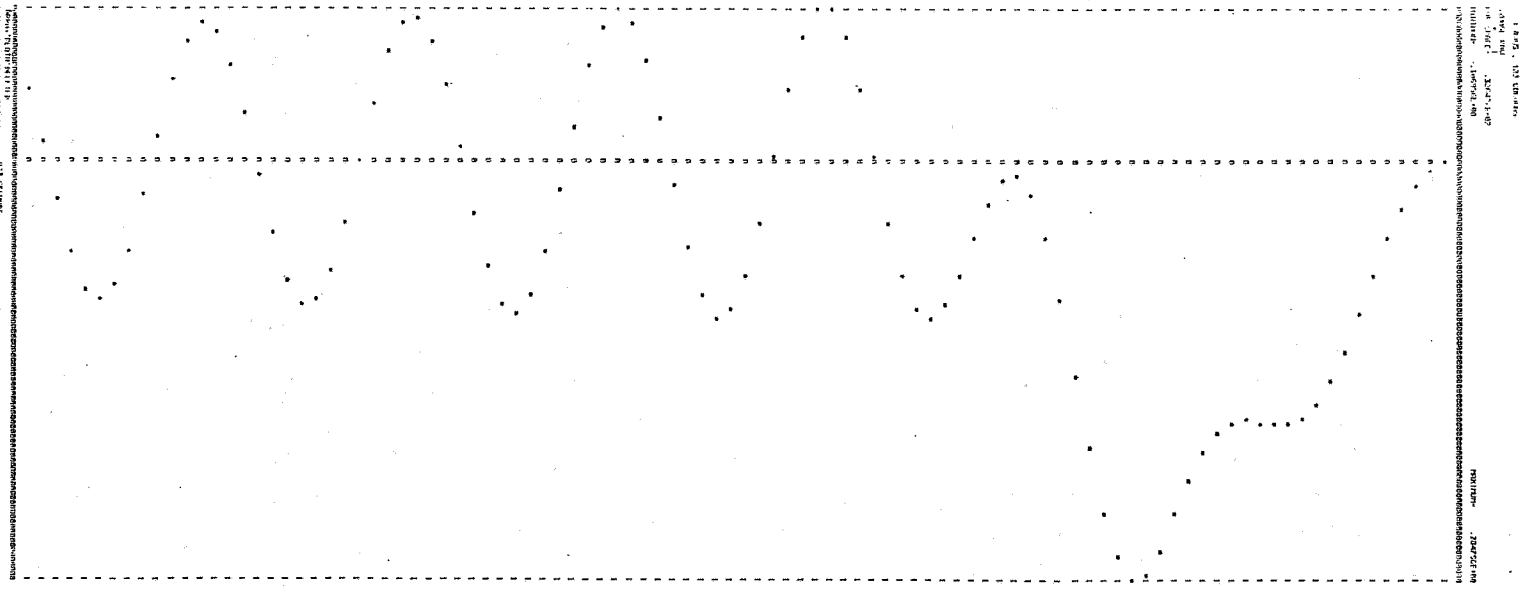
```
-------------------------------   .001 SECONDS
**START
-------------------------------   .028 SECONDS
**LABEL,3
                                 ********************************
                                 ** STATIC ANALYSIS EXAMPLE    **
                                 ********************************
-------------------------------   .054 SECONDS
**NODES,JOI,4                    ** GEOMETRY OF THE STRUCTURE **
   4 ROWS     3 COLUMNS

NODE NO              X              Y              Z
     1              0.             0.          400.000
     2              0.          1600.000         0.
     3              0.          2800.000      400.000
     4              0.          1600.000      400.000
-------------------------------   .050 SECONDS
**PRINT,JOI                      ** NODE NUMBERS AND COORDINATES **
                    1                2              3
     1  0.              0.                        .4000000E+03
     2  0.                .1600000E+04  0.
     3  0.                .2800000E+04    .4000000E+03
     4  0.                .1600000E+04    .4000000E+03
-------------------------------   .036 SECONDS
**BOUND,BC                       ** NON ZERO DISPLACEMENTS **
   4 ROWS     6 COLUMNS
   4    0    1    1    1    1    1    1

EQUATION NUMBERS FOR NODAL DISPLACEMENTS
NODE    X    Y    Z   XX   YY   ZZ
   1    0    0    0    0    0    0
   2    0    0    0    0    0    0
   3    0    0    0    0    0    0
```

```
   4    1    2    3    4    5    6
------------------------------  .045 SECONDS
**LOAD,BPR,2,7,1                ** BEAM PROPERTIES  **
   2 ROWS     7 COLUMNS
------------------------------  .043 SECONDS
**PRINT,BPR
                    1                2                3                4                5                6                7
      1   .3000000E+04   .1100000E+07   .2000000E+07   .7000000E+06   .3000000E+07   .1200000E+07   .6750000E+00
      2   .1000000E+04   .1160000E+06   .8250000E+05   .8250000E+05   .3000000E+07   .1200000E+07   .2250000E+00
------------------------------  .037 SECONDS
**BEAM,BRIDGE,JOI,BC,BPR

 EL.     I     J     K    NP          A         J11         I22         I33           E           G           M
   1     1     4     2     1    3000.00   1100000.    2000000.     700000.   3000000.    1200000.      .6750
   2     4     3     2     1    3000.00   1100000.    2000000.     700000.   3000000.    1200000.      .6750
   3     4     2     3     2    1000.00    116000.      82500.      82500.   3000000.    1200000.      .2250
   1 ROWS     5 COLUMNS
------------------------------  .128 SECONDS
**ADDSF,K,MAS                   ** FORM STIFFNESS AND MASS MATRICES **
   6 ROWS     1 COLUMNS
   6 ROWS     6 COLUMNS
------------------------------  .048 SECONDS
**PRINT,K                       ** STIFFNESS MATRIX **
                    1                2                3                4                5                6
      1   .1056510E+06  0.               0.               0.   -.9281250E+07  -.1093750E+08
      2   0.               .1317141E+08  0.                .9281250E+07  0.               0.
      3   0.               0.               .7520736E+07   .3828125E+07  0.               0.
      4   0.                .9281250E+07   .3828125E+07   .1472500E+11  0.               0.
      5  -.9281250E+07  0.               0.               0.                .4400000E+10  0.
      6  -.1093750E+08  0.               0.               0.               0.                .3534800E+11
------------------------------  .048 SECONDS
**LOADS,LDS,BC,1                ** LOAD VECTOR **
   6 ROWS     1 COLUMNS
 NODE  LOAD              FX               FY               FZ               MX               MY               MZ
    4     1    .100000E+04   0.               0.               0.               0.               0.
------------------------------  .042 SECONDS
**SOLVE,k,LDS
------------------------------  .031 SECONDS
**DISPL,LDS,BC                  ** DISPLACEMENTS **
 NODE  LOAD              UX               UY               UZ               TX               TY               TZ
    1     1   0.               0.               0.               0.               0.               0.
 NODE  LOAD              UX               UY               UZ               TX               TY               TZ
    2     1   0.               0.               0.               0.               0.               0.
 NODE  LOAD              UX               UY               UZ               TX               TY               TZ
    3     1   0.               0.               0.               0.               0.               0.
 NODE  LOAD              UX               UY               UZ               TX               TY               TZ
    4     1    .120935E-01   0.               0.               0.    .255097E-04    .374201E-05
------------------------------  .052 SECONDS
**FORCE,BRIDGE,LDS              ** MEMBER FORCES **
LOAD/FORCE   1       1          2                3                4                5                6                7                8
       1  0.               0.    -.26520E+03   .21046E+05  -.22619E+06  0.    -.19813E+06  0.
LOAD/FORCE   2       1          2                3                4                5                6                7                8
       1  0.               0.     .41035E+03  -.28061E+05   .26492E+06  0.     .22750E+06  0.
LOAD/FORCE   3       1          2                3                4                5                6                7                8
       1  0.               0.    -.32445E+03   .13022E+04  -.80674E+05  0.    -.49106E+05  0.
------------------------------  .065 SECONDS
**STOP
```

PART B - INPUT

```
START
LABEL,3
                                  ********************************
                                  ** DYNAMIC ANALYSIS EXAMPLE   **
                                  ********************************

NODES,JOI,4
     1              0.          0.         400.
     2              0.       1600.0          0.
     3              0.       2800.0       400.
     4              0.       1600.0       400.

LOAD,BPR(2,7)1
  (7E10.0)
    3.0E03     1.1E06     2.0E06     7.0E05     3.0E06     1.2E06   0.675E00
    1.0E03    11.6E04    82.5E03    82.5E03     3.0E06     1.2E08   0.225E00
PRINT,BPR                          ** BEAM PROPERTIES **
BOUND,BC
     4    0     1     1     1     1     1     1

BEAM,BRIDGE,JOI,BC,BPR
     1     1     4     2     1
     2     4     3     2     1
     3     4     2     3     2

ADDSF,K,MAS
DUP,K,KK
ZERO,MASS(6,6)0,0
STODG,MASS,MAS
ZERO,C(6,6)0,0
LOAD,DELTA(1,1)1
 (F10.0)
    0.05
PRINT,DELTA                        ** INCREMENT **
LOAD,FUA(2,4)1
 (4F10.0)
       0.         1.0        2.0         5.0
      0.0    20000.0        0.0         0.0
PRINT,FUA                           *  TIME DEPENDENT LOAD  *
FUNG,FUA,FUAI,DELTA,100,0
LOAD,SCALEP(6,1)
        1.          0.         0.          0.         0.          0.
PRINT,SCALEP
ZERO,INCO(6,3)0,0
STEP,K,MASS,C,INCO,DISP,SCALEP1FUAI,DELTA,1,100
.5         .16666667 1.42
PLOT,DISP,1                        ** X DISPLACEMENT AT JOINT 4 **
*    1
EIGEN,KK,MODES,MAS
PRINT,MODES                        **  MODE SHAPES **
SQREL,MAS
PRINT,MAS                          **  FREQUENCIES **
STOP
```

```
---------- 0.    SECONDS
**START
---------- .027 SECONDS
**LABEL,3
                          ********************************
                          ** DYNAMIC ANALYSIS EXAMPLE   **
                          ********************************
---------- .054 SECONDS
**NODES,JOI,4
   4 ROWS     3 COLUMNS

NODE NO              X              Y              Z
     1           0.             0.          400.000
     2           0.          1600.000          0.
     3           0.          2800.000       400.000
     4           0.          1600.000       400.000
---------- .050 SECONDS
**LOAD,BPR(2,7)1
   2 ROWS     7 COLUMNS
---------- .044 SECONDS
**PRINT,BPR                 ** BEAM PROPERTIES **
                    1              2              3              4              5              6              7
     1   .3000000E+04   .1100000E+07   .2000000E+07   .7000000E+06   .3000000E+07   .1200000E+07   .6750000E+00
     2   .1000000E+04   .1160000E+06   .8250000E+05   .8250000E+05   .3000000E+07   .1200000E+07   .2250000E+00
---------- .039 SECONDS
**BOUND,BC
   4 ROWS     6 COLUMNS
   4    0    1    1    1    1    1    1

EQUATION NUMBERS FOR NODAL DISPLACEMENTS
NODE     X    Y    Z   XX   YY   ZZ
   1     0    0    0    0    0    0
   2     0    0    0    0    0    0
   3     0    0    0    0    0    0
   4     1    2    3    4    5    6
---------- .042 SECONDS
**BEAM,BRIDGE,JOI,BC,BPR

 EL.    I    J    K   MP          A         J11         I22         I33           E           G           M
   1    1    4    2    1    3000.00   1100000.   2000000.    700000.   3000000.    1200000.      .6750
   2    4    3    2    1    3000.00   1100000.   2000000.    700000.   3000000.    1200000.      .6750
   3    4    2    3    2    1000.00    116000.     82500.     82500.   3000000.    1200000.      .2250
   1 ROWS     5 COLUMNS
---------- .129 SECONDS
**ADDSF,K,MAS
   6 ROWS     1 COLUMNS
   6 ROWS     6 COLUMNS
---------- .053 SECONDS
**DUP,K,KK
   6 ROWS     6 COLUMNS
---------- .028 SECONDS
**ZERO,MASS(6,6)0.0
   6 ROWS     6 COLUMNS
---------- .027 SECONDS
**STODG,MASS,MAS
---------- .027 SECONDS
**ZERO,C(6,6)0.0
   6 ROWS     6 COLUMNS
---------- .027 SECONDS
**LOAD,DELTA(1,1)1
   1 ROWS     1 COLUMNS
---------- .035 SECONDS
**PRINT,DELTA               ** INCREMENT **
                    1
     1   .5000000E-01
---------- .028 SECONDS
**LOAD,FUA(2,4)1
   2 ROWS     4 COLUMNS
---------- .040 SECONDS
**PRINT,FUA                 *  TIME DEPENDENT LOAD  *
                    1              2              3              4
     1  0.              .1000000E+01   .2000000E+01   .5000000E+01
     2  0.              .2000000E+05   0.                       0.
---------- .032 SECONDS
**FUNG,FUA,FUA1,DELTA,100,0
   1 ROWS   100 COLUMNS
---------- .042 SECONDS
**LOAD,SCALEP(6,1)
   6 ROWS     1 COLUMNS
---------- .031 SECONDS
**PRINT,SCALEP
                    1
     1   .1000000E+01
     2  0.
     3  0.
     4  0.
     5  0.
     6  0.
---------- .035 SECONDS
**ZERO,INCO(6,3)0.0
   6 ROWS     3 COLUMNS
---------- .028 SECONDS
**STEP,K,MASS,C,INCO,DISP,SCALEP,FUA1,DELTA,1,100
   6 ROWS   100 COLUMNS
 DEL=    .500000E+00   ALF=    .166667E+00   THE=    .142000E+01
---------- .451 SECONDS
**PLOT,DISP,1               ** X DISPLACEMENT AT JOINT 4 **
```



```
**EIGEN,KK,MODES,MAS
   6 ROWS     6 COLUMNS
 4 FIGURE TOLERANCE SPECIFIED
    6 ROTATIONS PERFORMED
------------------------------  .052 SECONDS
**PRINT,MODES                  **  MODE SHAPES **
                 1                 2                 3                 4                 5                 6
      1   .3108912E-01  0.                -.6255025E-02  -.2107663E-02  0.                0.
      2  0.                .5532924E-04  0.                0.                .6993498E-07   .3178204E-01
      3  0.                .4017185E-04  0.                0.               -.3178206E-01  0.
      4  0.               -.7799165E-04  0.                0.               -.9857978E-07   .1357755E-06
      5   .6674320E-04  0.                -.1385352E-04   .1025611E-02  0.                0.
      6   .1541338E-04  0.                 .7659790E-04   .3160330E-07  0.                0.
------------------------------  .048 SECONDS
**SOREL,MAS
------------------------------  .026 SECONDS
**PRINT,MAS                    **  FREQUENCIES **
                 1
      1   .9006345E+01
      2   .9461294E+01
      3   .1487432E+02
      4   .6832904E+02
      5   .8715913E+02
      6   .1153451E+03
------------------------------  .034 SECONDS
**STOP
```

APPENDIX B - INTERNAL ORGANIZATION OF CAL

1. List of Subroutines

CAL is composed of a sequence of FORTRAN subroutines which can be categorized as follows:

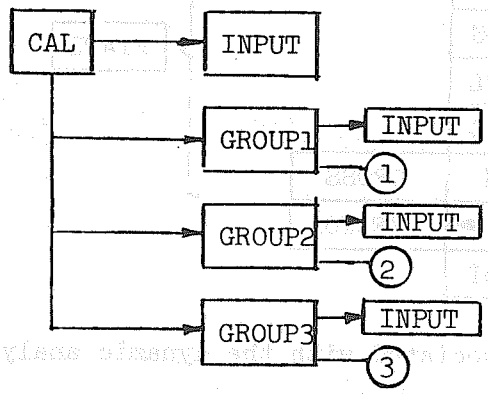
a. The main program CAL and GROUP1, GROUP2 or GROUP3 interpret the next operation and transfers control to the appropriate subprograms.

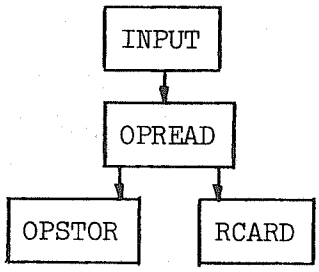
b. A series of subroutines INPUT, OPREAD, OPSTOR and RCARD supply the next operation to be executed.

c. The individual operations are executed by different subroutines which have been subdivided into three different groups.

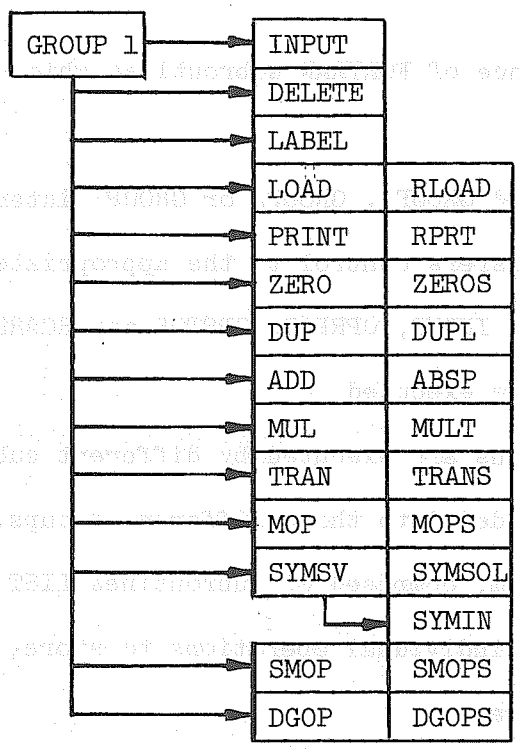
d. A data management system, composed of subroutines LIST, FIND and DELETE, is used by the individual operations to store, locate or delete arrays from storage.

The operations are subdivided into three groups in order to reduce the length of the main CAL program. For a small computer system it may be desirable to make these separate overlays. A simple block diagram illustrating the interconnection between these subprograms is shown below.

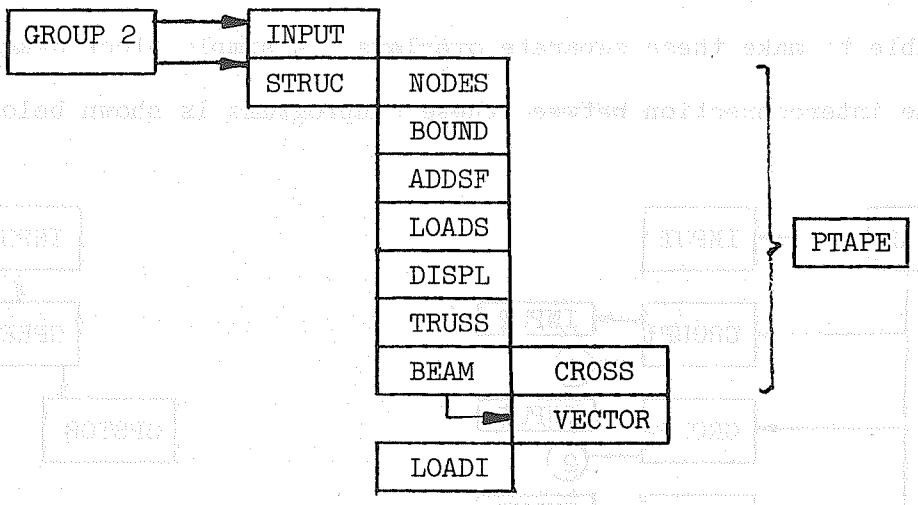




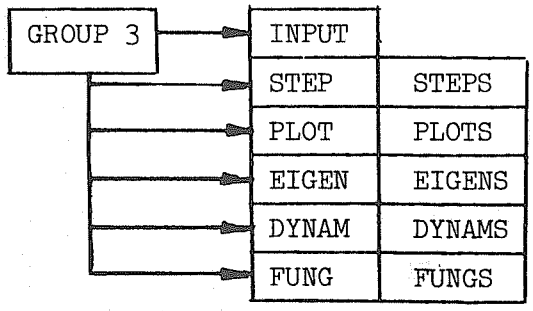
The subroutines in GROUP 1 are the general matrix operations.



The subroutines in GROUP 2 are the structural analysis operations.



The subroutine in GROUP 3 are associated with the dynamic analysis operations.



2. Array Storage and Directories

All arrays defined by the user are stored in one internal array dimensioned L(4000) the size of this array is variable and is set by the statement MTOT = 4000.

Arrays are stored sequentially, column-wise, in the L(K) array. Each array is preceeded by a 6 word directory which contains the following information.

I(1)=L(K)	=	Number of terms in array.
I(2)=L(K+1)	=	Number of computer storage locations per term.
I(3)=L(K+2)	=	Number of columns in array.
I(4)=L(K+3)	=	Form of storage (1 = rectangular form)
I(5)=L(K+4)	=	Name of Array
I(6)=L(K+5)	=	Name of Associated Array (not used in this version of CAL)
L(NA)=L(K+6)	=	First term in Array.

3. Incore Data Management System

The subroutine LIST will setup the directory and reserve storage at the end of the L array and return the starting address, NA, to the calling program. The subroutine FIND will search through all directories to locate a specific array and return its size and location to the calling program. The subroutine DELETE will eliminate an array and its directory from storage and move all arrays and directories stored at higher locations in order to compact storage. All new arrays generated will be stored at the end of the L array.

APPENDIX C

FORTRAN LISTING OF CAL

This listing may not be the
most current version of the program.
For the latest version, you
can obtain the source deck from:

NISEE/Computer Applications
Davis Hall
University of California
Berkeley, California 94720

```
      PROGRAM CAL(INPUT,OUTPUT,TAPE1)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(5000)
      COMMON /LOOPD/ LA,IN,INDEX(5),LSTART(5),IJKLM(5)
      COMMON /TEMP/ SSS(100)
      COMMON /ADIR/ NA,I(10)
C-----SET COMPUTER PRECISION INDICATORS AND CONSTANTS
      MTOT=5000
      NO=0
      NH=1
      NSP=1
      NDP=1
      NDIR=4+2*NH
      LA=0
      IN=0
      NARA=0
      NSIZE=1
      CALL INPUT
      LAST=1
      GO TO 110
C----- SEARCH GROUPS OF OPERATIONS FOR INPUT OPERATION
  100 IF(LAST.EQ.1) GO TO 300
  110 CALL GROUP1(LAST)
      IF(LAST.EQ.2) GO TO 300
      CALL GROUP2(LAST)
      IF(LAST.EQ.3) GO TO 300
      CALL GROUP3(LAST)
      GO TO 100
  300 PRINT 3000
      STOP
 3000 FORMAT (30HOOPERATION UNDEFINED OR BLANK    )
      END
```

```
      SUBROUTINE INPUT
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /LOOPD/ LA,IN,INDEX(5),LSTART(5),IJKLM(5)
      COMMON /ADIR/ NA,I(10)
      DATA LNAME /6HLINAME/
      NW=10*NH+4
      IF(TO.LT.0.0) CALL SECOND(TO)
  100 CALL SECOND(T)
      TD=T-TO
      IF(NO.EQ.0)PRINT 2000,TD
      TO=T
      IF(LA.EQ.0) CALL OPREAD
      IF(LA.EQ.0) GO TO 175
      CALL FIND(LNAME)
      CALL OPSTOR(L(NA),NW)
      IF(LA.EQ.0) GO TO 100
  175 RETURN
 2000 FORMAT(1X,30(1H-),F6.3,8H SECONDS )
      END
```

```
      SUBROUTINE RCARD
      COMMON NO
      COMMON /CARD/ INHOL(3,10),NN(4),S1,S2
       COMMON /TEMP/ IN(80),IH(20)
      DATA IBL/2H  /,IZERO/2H0 /,NINE/2H9 /,IA/2HA /,IZ/2HZ /
      DATA NC/10/,MASK/77B/,NB/6/,NW/1/,NCW/10/
C     DATA NC/6/,MASK/377B/,NB/8/,NW/3/,NCW/2/-----FOR 16 BIT COMPUTERS
C-----READ  OP,N1,N2,N3,N4,N5,N1,N2,N3,N4 ----
      READ 1000, IN
      IF(NO.EQ.0) PRINT 2000,IN
 1000 FORMAT (80A1)
 2000 FORMAT (3H **,80A1)
C-----SET INITIAL CONDITIONS ------------------
      DO 50 J=1,10
      DO 50 I=1,NW
   50 INHOL(I,J)=0
      DO 60 J=1,4
   60 NN(J)=0
      J=0
      K=0
      L=1
   70 DO 80 JJ=1,NC
   80 IH(JJ)=IBL
      M=0
   90 J=J+1
C-----INTERPRETE CHARACTER ---------------------
      II=IN(J)
      IF(II.GE.IZERO.AND.II.LE.NINE) GO TO 100
      IF(II.GE.IA.AND.II.LE.IZ) GO TO 110
      IF(KT.EQ.0) GO TO 120
      L=L+1
      GO TO 190
C-----NUMERICAL DATA -------------
  100 NNN=MASK.AND.LEFT(II-IZERO,NB)
      NN(L)=10*NN(L) + NNN
      KT=1
      GO TO 90
C-----HOLLERITH DATA ---------------------
  110 M=M+1
      IH(M)=II
      KT=0
      GO TO 90
C------STORE HOLLERITH STRING -------------
  120 K=K+1
      JJ=0
      DO 130 I=1,NW
      DO 130 M=1,NCW
      JJ=JJ+1
      NNN=MASK.AND.LEFT(IH(JJ),NB)
  130 INHOL(I,K)=LEFT(INHOL(I,K),NB) + NNN
C------CHECK FOR END OF INFORMATION ------------
  190 IF(IN(J).NE.IBL ) GO TO 70
      RETURN
      END
```

```
      SUBROUTINE OPREAD
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),NN(4),S1,S2
      COMMON /LOOPD/ LA,IN,INDEX(5),LSTART(5),IJKLM(5)
      DATA LOOP /5HLOOP /,NEXT /5HNEXT /
      DATA LNAME /6HLINAME/
      LOGICAL COMP
C-----READ AND PRINT ALL  INPUT OPERATION CARDS-----------------
      N=NSIZE+NDIR
      IN=0
  100 CALL RCARD
      IF(COMP(INHOL(1,1),LOOP)) IN=IN+1
      IF((IN.EQ.1).AND.(LA.EQ.0)) LA=1
      IF(LA.EQ.0) RETURN
C------STORE ALL OPERATIONS WITHIN OUTER LOOP----------
      NW=10*NH+4
      DO 200 J=1,10
      DO 200 I=1,NH
      L(N)=INHOL(I,J)
  200 N=N+1
      DO 210 I=1,4
      L(N)=NN(I)
  210 N=N+1
      IF(COMP(INHOL(1,1),NEXT)) IN=IN-1
      IF(IN.EQ.0) GO TO 250
      LA=LA+1
      GO TO 100
C     SET NAME AND STORAGE FOR LOOP DATA
  250 CALL LIST(LNAME,NW,LA,1)
      LA=1
      IN=0
      RETURN
      END
```

```
      SUBROUTINE OPSTOR(LOOPA,NW)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),NN(4),S1,S2
      COMMON /LOOPD/ LA,IN,INDEX(5),LSTART(5),IJKLM(5)
      COMMON /ADIR/NA,III(10)
      DIMENSION LOOPA(NW,1)
      DATA LOOP /5HLOOP /,NEXT /5HNEXT /,SKIP /5HSKIP /,LNAME /6HLINAME/
      LOGICAL COMP
      IF(LA.NE.1) GO TO 50
      DO 40 I=1,5
   40 IJKLM(I)=0
      IF(NO.EQ.0) PRINT 2001
C-----TRANSFER OPERATION FROM STORAGE TO CARD COMMON  ----------
   50 I=1
      DO 100 J=1,10
      DO 100 K=1,NH
      INHOL(K,J)=LOOPA(I,LA)
  100 I=I+1
      DO 110 J=1,4
      NN(J)=LOOPA(I,LA)
  110 I=I+1
      LA=LA+1
      IF(NO.EQ.0)PRINT 2000,((INHOL(I,J),I=1,NH),J=1,10)
     1 ,(IJKLM(I),I=1,IN)
      IF(COMP(INHOL(1,1),SKIP)) GO TO 200
      IF(COMP(INHOL(1,1),LOOP)) GO TO 300
      IF(COMP(INHOL(1,1),NEXT)) GO TO 400
      RETURN
C-----SKIP OPERATION- CHECK SIGN AND INCREMENT COUNTER------
  200 CALL FIND(INHOL(1,2))
      CALL MOVE(L(NA),S1,NSP)
      IF(NO.EQ.0)PRINT 2002,INHOL(1,2),S1,NN(1)
      IF(S1.LT.0.0) LA=LA+NN(1)
      GO TO 50
C-----LOOP OPERATION- SET INDEX LIMIT FOR LOOP ----------
  300 IN=IN+1
      INDEX(IN)=NN(1)
      IJKLM(IN)=1
      LSTART(IN)=LA
      GO TO 50
C-----NEXT OPERATION- INCREMENT INDEX AND CHECK FOR LAST OPERATION---
  400 INDEX(IN)=INDEX(IN)-1
      IJKLM(IN)=IJKLM(IN)+1
      IF(NO.EQ.0) PRINT 2003,IN
      IF(INHOL(1,2).EQ.0) GO TO 450
      CALL FIND(INHOL(1,2))
      CALL MOVE(L(NA),S1,NSP)
      IF(NO.EQ.0)PRINT 2002,INHOL(1,2),S1
      IF(S1.LT.0.0) INDEX(IN)=0
  450 IF(INDEX(IN).NE.0) LA=LSTART(IN)
      IF(INDEX(IN).EQ.0) IJKLM(IN)=0
      IF(INDEX(IN).EQ.0) IN=IN-1
      IF(IN.EQ.0) GO TO 500
      GO TO 50
C-----DELETE ARRAY OF OPERATIONS WITHIN LOOPS--------
  500 LA=0
      CALL DELETE(LNAME)
      RETURN
 2000 FORMAT(5H  *** 10A6,I5,3H=L1,I5,3H=L2,I5,3H=L3,I5,3H=64,I5,3H=L5)
 2001 FORMAT(26HOSTART OF LOOPING OPERATIONS  )
 2002 FORMAT (1X,A6,3H = E16.7,17H IF NEGATIVE SKIP    I5)
 2003 FORMAT(25H LAST OPERATION IN LOOP L ,I1/1H0)
      END
```

```
      SUBROUTINE LIST(II,NR,NC,NP)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /ADIR/ NA,I(10)
      IF(NO.EQ.0) PRINT 2001, NR,NC
      I(1)=NR*NC
      I(2)=NP
      I(3)=NC
      I(4)=1
      CALL MOVE(II,I(5),NH)
C-----CHECK CAPACITY OF L ARRAY
      NN=I(1)*I(2)+NDIR
      IF(NN.LT.MTOT-NSIZE) GO TO 50
      NX=NSIZE+NN
      PRINT 2000,MTOT,NX
      STOP
C-----SET DIRECTORY AND RESERVE STORAGE FOR NEW ARRAY------
   50 DO 100 N=1,NDIR
      IN=NSIZE-1+N
  100 L(IN)=I(N)
      NA=NSIZE+NDIR
      NARA=NARA+1
      NSIZE=NSIZE+NN
      RETURN
 2000 FORMAT (19HOSTORAGE EXCEEDED-- I5,11H RESERVED-- I5,9H REQUIRED)
 2001 FORMAT(I5,6H ROWS ,I5,8H COLUMNS )
      END
```

```
      SUBROUTINE FIND(II)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /ADIR/ NA,I(10)
      LOGICAL COMP
C-----LOCATE ARRAY AND ITS DIRECTORY-- NA=0 IF ARRAY II IS NOT FOUND
      N=1
      NA=0
  100 IF(N.EQ.NSIZE) GO TO 400
      IF(COMP(II,L(N+4))) GO TO 200
      N=N+L(N)*L(N+1)+NDIR
      GO TO 100
C-----COMPUTE ADDRESS AND TRANSFER DIRECTORY
  200 DO 300 K=1,NDIR
      M=N+K-1
  300 I(K)=L(M)
      NA=M+1
      RETURN
  400 PRINT 2000, II
      STOP
 2000 FORMAT ( 7HOARRAY# A5,23H#NOT PREVIOUSLY DEFINED )
      END
```

```
      SUBROUTINE DELETE(II)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /ADIR/ NA,I(10)
      LOGICAL COMP
C-----LOCATE ARRAY TO BE DELETED
      N=1
      NA=0
   10 IF(N.EQ.NSIZE) RETURN
      IF(COMP(II,L(N+4))) GO TO 20
      N=N+L(N)*L(N+1)+NDIR
      GO TO 10
C-----COMPUTE ADDRESS AND TRANSFER DIRECTORY
   20 DO 30  K=1,NDIR
      M=N+K-1
   30 I(K)=L(M)
      NA=M+1
C-----DELETE ARRAY AND RELOCATE REMAINING ARRAYS
      NARA=NARA-1
      NN=NA+I(1)*I(2)
      NL=NSIZE-1
      NSH=NDIR+I(1)*I(2)
      IF(NN.EQ.NSIZE) GO TO 200
      DO 100 N=NN,NL
  100 L(N-NSH)=L(N)
  200 NSIZE=NSIZE-NSH
      IF(NO.EQ.0) PRINT 2000,II
      RETURN
 2000 FORMAT (8H -ARRAY# A5,8H#DELETED )
      END
```

```
      FUNCTION COMP(I,J)
      DIMENSION I(3),J(3)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      LOGICAL COMP
      COMP=.FALSE.
      DO 100 K=1,NH
      IF(I(K).NE.J(K)) RETURN
  100 CONTINUE
      COMP=.TRUE.
      RETURN
      END
```

```
      SUBROUTINE MOVE(II,JJ,N)
      DIMENSION II(3),JJ(3)
      DO 100 I=1,N
  100 JJ(I)=II(I)
      RETURN
      END
```

```
      SUBROUTINE ERCOM
      PRINT 2000
      STOP
 2000 FORMAT (24HOMATRICES NOT COMPATIBLE )
      END
```

```
      SUBROUTINE GROUP1(LAST)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      LOGICAL COMP
      DIMENSION IOP(3,40)
      DATA IOP(1,1)/6HSTART /
      DATA IOP(1,2)/6HLOAD  /
      DATA IOP(1,3)/6HPRINT /
      DATA IOP(1,4)/6HZERO  /
      DATA IOP(1,5)/6HDELETE/
      DATA IOP(1,6)/6HDUP   /
      DATA IOP(1,7)/6HADD   /
      DATA IOP(1,8)/6HSUB   /
      DATA IOP(1,9 )/6HNO    /
      DATA IOP(1,10)/6HYES   /
      DATA IOP(1,11)/6HSTOP  /
      DATA IOP(1,12)/6HMULT  /
      DATA IOP(1,13)/6HTRAN  /
      DATA IOP(1,14)/6HINVEL /
      DATA IOP(1,15)/6HSQREL /
      DATA IOP(1,16)/6HLOG   /
      DATA IOP(1,17)/6HSCALE /
      DATA IOP(1,18)/6HSOLVE /
      DATA IOP(1,19)/6HUSERA /
      DATA IOP(1,20)/6HUSERB /
      DATA IOP(1,21)/6HMAX   /
      DATA IOP(1,22)/6HNORM  /
      DATA IOP(1,23)/6HPROD  /
      DATA IOP(1,24)/6HDUPSM /
      DATA IOP(1,25)/6HSTOSM /
      DATA IOP(1,26)/6HDUPDG /
      DATA IOP(1,27)/6HSTODG /
      DATA IOP(1,28)/6HLABEL /
      NUMOP=28
      GO TO 175
C---- READ OPERATION FROM CARD OR STORAGE  ----------------
  100 CALL INPUT
C-----INTERPRETE OPERATION ------------------------------
  175 DO 200 J=1,NUMOP
      N=J
      IF (COMP(INHOL(1,1),IOP(1,J))) GO TO 300
  200 CONTINUE
      RETURN
C-----EXECUTE APPROPRIATE OPERATION ---------------------
  300 LAST=1
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
     1 23,24,25,26,27,28),N
    1 NARA=0
      NSIZE=1
      GO TO 100
    2 CALL LOAD
      GO TO 100
    3 IF(N1.GT.0) CALL LABEL(N1)
      IF(NO.EQ.0) CALL PRINT
      GO TO 100
    4 CALL ZERO
      GO TO 100
    5 CALL DELETE(INHOL(1,2))
      GO TO 100
    6 CALL DUP
      GO TO 100
    7 CALL ADD(1.)
      GO TO 100
    8 CALL ADD(-1.)
      GO TO 100
    9 NO=1
      GO TO 100
   10 NO=0
      GO TO 100
   11 STOP
   12 CALL MUL
      GO TO 100
   13 CALL TRAN
      GO TO 100
   14 CALL MOP(1)
      GO TO 100
   15 CALL MOP(2)
      GO TO 100
   16 CALL MOP(3)
      GO TO 100
   17 CALL MOP(4)
      GO TO 100
   18 CALL SYMSV
      GO TO 100
   19 CALL USERA
      GO TO 100
   20 CALL USERB
      GO TO 100
   21 CALL MOP(5)
      GO TO 100
   22 CALL MOP(6)
      GO TO 100
   23 CALL MOP(7)
      GO TO 100
   24 CALL SMOP(1)
      GO TO 100
   25 CALL SMOP(2)
      GO TO 100
   26 CALL DGOP(1)
      GO TO 100
   27 CALL DGOP(2)
      GO TO 100
   28 IF(N1.GT.0) CALL LABEL(N1)
      GO TO 100
      END
```

```
      SUBROUTINE LOAD
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,2))
      CALL LIST(INHOL(1,2),N1,N2,NSP)
      CALL RLOAD(L(NA),N1,N2,N3)
      RETURN
      END
      SUBROUTINE RLOAD(A,NR,NC,N3)
      COMMON /TEMP/ FOR(40)
      DIMENSION A(NR,NC)
      IF(N3.NE.0) GO TO 100
      READ 1000, ((A(I,J),J=1,NC),I=1,NR)
      RETURN
  100 READ 1001,FOR
      READ FOR,((A(I,J),J=1,NC),I=1,NR)
      RETURN
 1000 FORMAT (8F10.0)
 1001 FORMAT(40A2)
      END
```

```
      SUBROUTINE PRINT
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL FIND(INHOL(1,2))
      NR=I(1)/I(3)
      CALL RPRT(L(NA),NR,I(3))
      RETURN
      END
      SUBROUTINE RPRT(A,NR,NC)
      DIMENSION A(NR,NC)
      DO 100 I=1,NC,8
      IH=I+7
      IF(IH.GT.NC) IH=NC
      PRINT 2000, (K,K=I,IH)
      DO 100 J=1,NR
  100 PRINT 2001, (J,(A(J,K),K=I,IH))
      RETURN
 2000 FORMAT (8X,8I15)
 2001 FORMAT (I8,8E15.7)
      END
```

```
      SUBROUTINE DUP
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,3))
      CALL FIND (INHOL(1,2))
      NR=I(1)/I(3)
      NC=I(3)
      NB=NA
      CALL LIST(INHOL(1,3),NR,NC,NSP)
      CALL DUPL(L(NB),L(NA),NR,NC)
      RETURN
      END
      SUBROUTINE DUPL(A,B,NR,NC)
      DIMENSION A(NR,NC),B(NR,NC)
      DO 100 I=1,NR
      DO 100 J=1,NC
  100 B(I,J)=A(I,J)
      RETURN
      END
```

```
      SUBROUTINE LABEL(N1)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /TEMP/ II(40)
      DO 100 I=1,N1
      READ 1000, II
      IF(NO.EQ.0) PRINT 1000 , II
  100 CONTINUE
      RETURN
 1000 FORMAT (40A2)
      END
```

```
      SUBROUTINE DGOP(N)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
C-----SUBROUTINE TO DUPLICATE OR STORE DIAGONAL----------
      IF(N.EQ.1) CALL DELETE(INHOL(1,3))
      CALL FIND(INHOL(1,2))
      N1=I(3)
      IF(I(3).NE.I(1)/I(3)) CALL ERCOM
      N2=NA
      IF(N.NE.1) GO TO 100
      CALL LIST(INHOL(1,3),1,N1,NSP)
      GO TO 200
  100 CALL FIND(INHOL(1,3))
      IF(I(1).NE.N1) CALL ERCOM
  200 N3=NA
      CALL DGOPS(L(N2),L(N3),N1,N)
      RETURN
      END
      SUBROUTINE DGOPS(A,B,M,N)
      DIMENSION A(M,M),B(M)
      DO 300 I=1,M
      GO TO (1,2),N
    1 B(I)=A(I,I)
      GO TO 300
    2 A(I,I)=B(I)
  300 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MOP(N)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      IF(N.GT.4) CALL DELETE(INHOL(1,3))
      CALL FIND(INHOL(1,2))
      N3=NA
      NC=I(3)
      NR=I(1)/I(3)
      IF(N.EQ.4)CALL FIND(INHOL(1,3))
      IF(N.EQ.5) CALL LIST(INHOL(1,3),NR,1,NSP)
      IF(N.EQ.6) CALL LIST(INHOL(1,3),1,NC,NSP)
      IF(N.EQ.7) CALL LIST(INHOL(1,3),1,2 ,NSP)
      CALL MOPS(L(N3),L(NA),NR,NC,N,N1)
      RETURN
      END
      SUBROUTINE MOPS(A,B,NR,NC,N,N1)
      DIMENSION A(NR,NC),B(NC)
      GO TO (100,200,300,400,500,600,700),N
C-----REPLACE EACH ELEMENT WITH ITS INVERSE-------
  100 DO 150 I=1,NR
      DO 150 J=1,NC
  150 A(I,J)=1.0/A(I,J)
      RETURN
C-----REPLACE EACH ELEMENT WITH ITS SQUARE ROOT---
  200 DO 250 I=1,NR
      DO 250 J=1,NC
  250 A(I,J)=SQRT(A(I,J))
      RETURN
C-----REPLACE EACH ELEMENT WITH ITS LOG-----------
  300 DO 350 I=1,NR
      DO 350 J=1,NC
  350 A(I,J)=ALOG(A(I,J))
      RETURN
C-----REPLACE EACH ELEMENT WITH ITS SELF TIMES A SCALAR
  400 DO 450 I=1,NR
      DO 450 J=1,NC
  450 A(I,J)=A(I,J)*B(1)
      RETURN
C-----EVALUATE THE MAXIMUM OF EACH ROW------------
  500 DO 550 I=1,NR
      B(I)=0.0
      DO 540 J=1,NC
      X=ABS(A(I,J))
      IF(X.LT.B(I)) GO TO 540
      B(I)=X
      JMAX=J
  540 CONTINUE
      IF(NO.EQ.0) PRINT 2000, JMAX,A(I,JMAX)
 2000 FORMAT (I8,E16.7)
  550 CONTINUE
      RETURN
C-----EVALUATE COLUMN NORMS---------------------
  600 DO 650 J=1,NC
      B(J)=0.0
      DO 640 I=1,NR
      IF(N1.GT.0) GO TO 630
      B(J)=B(J)+ABS(A(I,J))
      GO TO 640
  630 B(J)=B(J)+A(I,J)*A(I,J)
  640 CONTINUE
      IF(N1.GT.0) B(J)=SQRT(B(J))
  650 CONTINUE
      RETURN
C------EVALUATE PRODUCT OF ALL ELEMENTS----------
  700 B(1)=1.0
      B(2)=1.0
      DO 760 I=1,NR
      DO 740 J=1,NC
      B(1)=B(1)*A(I,J)
      IF(B(1).EQ.0.) GO TO 740
  710 IF(ABS(B(1)).LT.1.) GO TO 720
      B(1)=B(1)/10.
      B(2)=B(2)+1.0
      GO TO 710
  720 IF(ABS(B(1)).GT.0.1) GO TO 740
      B(1)=B(1)*10.
      B(2)=B(2)-1.0
      GO TO 720
  740 CONTINUE
  760 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE ZERO
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      S1=N3
      S2=N4
      CALL DELETE(INHOL(1,2))
      CALL LIST(INHOL(1,2),N1,N2,NSP)
      CALL ZEROS(L(NA))
      RETURN
      END
      SUBROUTINE ZEROS(A)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      DIMENSION A(1)
      K=1
      DO 300 II=1,N2
      A(K)=S1
      K=K+1
      DO 300 JJ=1,N1
      A(K)=S2
  300 K=K+1
      RETURN
      END
```

```
      SUBROUTINE ADD(S)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL FIND(INHOL(1,3))
      NR=I(3)
      NC=I(1)/NR
      NB=NA
      CALL FIND(INHOL(1,2))
      IF(NR.NE.I(3)) CALL ERCOM
      CALL ADSB(L(NA),L(NB),NR,NC,S)
      RETURN
      END
      SUBROUTINE ADSB(A,B,NR,NC,S)
      DIMENSION A(NR,NC),B(NR,NC)
      DO 100 I=1,NR
      DO 100 J=1,NC
  100 A(I,J)=A(I,J)+S*B(I,J)
      RETURN
      END
```

```
      SUBROUTINE MUL
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,4))
      CALL FIND(INHOL(1,2))
      NR=I(1)/I(3)
      NC=I(3)
      N1=NA
      CALL FIND(INHOL(1,3))
      IF(NC.NE.I(1)/I(3)) CALL ERCOM
      N2=NA
      N3=I(3)
      CALL LIST(INHOL(1,4),NR,N3,NSP)
      CALL MULT(L(N1),L(N2),L(NA),NR,N3,NC)
      RETURN
      END
      SUBROUTINE MULT(A,B,C,NR,NC,NT)
      DIMENSION A(NR,NT),B(NT,NC),C(NR,NC)
      DO 200 I=1,NR
      DO 200 J=1,NC
      X=0.0
      DO 100 K=1,NT
  100 X=X+A(I,K)*B(K,J)
  200 C(I,J)=X
      RETURN
      END
```

```
      SUBROUTINE TRAN
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,3))
      CALL FIND(INHOL(1,2))
      NR=I(1)/I(3)
      NC=I(3)
      N1=NA
      CALL LIST(INHOL(1,3),NC,NR,NSP)
      CALL TRANS(L(N1),L(NA),NR,NC)
      RETURN
      END
      SUBROUTINE TRANS(A,B,NR,NC)
      DIMENSION A(NR,NC),B(NC,NR)
      DO 100 I=1,NR
      DO 100 J=1,NC
  100 B(J,I)=A(I,J)
      RETURN
      END
```

```
      SUBROUTINE SMOP(N)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
C-----SUBROUTINE TO DUPLICATE OR STORE SUBMATRICES--------
      IF(N.NE.1) GO TO 100
      CALL DELETE(INHOL(1,3))
      CALL LIST(INHOL(1,3),N3,N4,NSP)
      GO TO 200
  100 CALL FIND(INHOL(1,3))
      N3=I(1)/I(3)
      N4=I(3)
  200 NB=NA
      CALL FIND(INHOL(1,2))
      NR=I(1)/I(3)
      NC=I(3)
      IF(N1+N3-1.GT.NR) CALL ERCOM
      IF(N2+N4-1.GT.NC) CALL ERCOM
      CALL SMOPS(L(NA),L(NB),NR,NC,N1,N2,N3,N4,N)
      RETURN
      END
      SUBROUTINE SMOPS(A,B,NR,NC,N1,N2,N3,N4,N)
      DIMENSION A(NR,NC),B(N3,N4)
      II=N1
      DO 400 I=1,N3
      JJ=N2
      DO 300  J=1,N4
      GO TO (1,2) ,N
    1 B(I,J)=A(II,JJ)
      GO TO 300
    2 A(II,JJ)=B(I,J)
  300 JJ=JJ+1
  400 II=II+1
      RETURN
      END
```

```
      SUBROUTINE SYMSV
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      IF(NO.NE.0) GO TO 100
      IF(N1.EQ.1) PRINT 2000
      IF (N1.EQ.2) PRINT 2001
      IF(N1.EQ.3) PRINT 2002
  100 CALL FIND(INHOL(1,2))
      NR=I(1)/I(3)
      IF(N1.EQ.3) GO TO 300
      IF(NR.NE.I(3)) CALL ERCOM
      N2=NA
      IF(N1.EQ.1) GO TO 200
      CALL FIND(INHOL(1,3))
      IF(NR.NE.I(1)/I(3)) CALL ERCOM
  200 CALL SYMSOL(L(N2),L(NA),NR,I(3),N1)
      RETURN
  300 CALL SYMIN(L(NA),NR)
      RETURN
 2000 FORMAT (20H TRIANGULARIZE ONLY  )
 2001 FORMAT (44H FORWARD REDUCTION AND BACKSUBSTITUTION ONLY )
 2002 FORMAT (22H MATRIX INVERSION ONLY )
      END
```

```
      SUBROUTINE SYMSOL(A,B,NN,LL,M)
C     SYMMETRIC EQUATION SOLVER- E.L. WILSON 1976
C     M=0 TRIANGULARIZE AND SOLVE
C     M=1 TRIANGULARIZE ONLY
C     M=2 FORWARD REDUCTION AND BACKSUBSTITUTION ONLY
      DIMENSION A(NN,NN),B(NN,LL)
C
      IF(M.EQ.2) GO TO 500
      DO 400 N=1,NN
      IF(N.EQ.NN) GO TO 600
      D=A(N,N)
      IF(D.EQ.0.0) PRINT 2000,N
      N1=N+1
      DO 300 J=N1,NN
      IF(A(N,J).EQ.0.0) GO TO 300
      A(N,J)=A(N,J)/D
      DO 200 I=J,NN
      A(I,J)=A(I,J)-A(I,N)*A(N,J)
  200 A(J,I)=A(I,J)
  300 CONTINUE
  400 CONTINUE
C     FORWARD REDUCTION AND BACKSUBSTITUTION
  500 IF(M.EQ.1) RETURN
      DO 700 N=1,NN
      DO 600 L=1,LL
  600 B(N,L)=B(N,L)/A(N,N)
      IF(N.EQ.NN) GO TO 800
      N1=N+1
      DO 700 L=1,LL
      DO 700 I=N1,NN
  700 B(I,L)=B(I,L)-A(I,N)*B(N,L)
C
  800 N1=N
      N=N-1
      IF(N.EQ.0) RETURN
      DO 900 L=1,LL
      DO 900 J=N1,NN
  900 B(N,L)=B(N,L)-A(N,J)*B(J,L)
      GO TO 800
C
 2000 FORMAT (39H0***ZERO DIAGONAL TERM EQUATION NUMBER    I4)
      END
```

```
      SUBROUTINE SYMIN(A,M)
C-----INVERSION OF POSITIVE DEFINITE SYMMETRIC MATRIX-------
      DIMENSION A(M,M)
C     EVALUATION OF NEGATIVE INVERSE
      DO 400 N=1,M
      D=A(N,N)
      DO 300 I=1,M
      AN=A(I,N)/D
      IF(I.EQ.N) GO TO 200
      DO 100 J=I,M
      A(I,J)=A(I,J)-AN*A(N,J)
  100 A(J,I)=A(I,J)
  200 A(I,N)=AN
  300 A(N,I)=AN
  400 A(N,N)=-1.0/D
C     CHANGE SIGN OF INVERSE
      DO 600 I=1,M
      DO 500 J=1,M
  500 A(I,J)=-A(I,J)
C
      RETURN
      END
```

```
      SUBROUTINE GROUP2(LAST)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NOP,NH,NDIR,L(1)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /TEMP/ SSS(100)
      LOGICAL COMP
      DIMENSION IOP(3,20)
      DATA IOP(1, 1)/6HNODES /,IOP(1, 2)/6HBOUND /,IOP(1, 3)/6HTRUSS /
      DATA IOP(1, 4)/6HBEAM  /,IOP(1, 5)/6HADDSF /,IOP(1, 6)/6HLOADS /
      DATA IOP(1, 7)/6HFORCE /,IOP(1, 8)/6HDISPL /,IOP(1, 9)/6HLOADI /
      NUMOP=9
      GO TO 175
C---- READ OPERATION FROM CARD OR STORAGE  ----------------
  100 CALL INPUT
C-----INTERPRETE OPERATION ------------------------------
  175 DO 200 J=1,NUMOP
      N=J
      IF (COMP(INHOL(1,1),IOP(1,J))) GO TO 300
  200 CONTINUE
      RETURN
C-----EXECUTE APPROPRIATE OPERATION ---------------------
  300 LAST=2
      CALL STRUC(N)
      GO TO 100
      END
```

```
      SUBROUTINE STRUC(NOP)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NOP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /STR/ ND,NS,NRP,LM(12),ST(8,12),XM(12),S(12,12)
      COMMON /ADIR/ NA,I(10)
      EQUIVALENCE (M1,INHOL(1,2)),(M2,INHOL(1,3)),(M3,INHOL(1,4))
      EQUIVALENCE (M4,INHOL(1,5)),(M5,INHOL(1,6)),(M6,INHOL(1,7))
      GO TO (100,200,300,300,500,600,700,800,900),NOP
C-----READ NODE COORDINATES-----------------------
  100 CALL DELETE(M1)
      NUMNP=N1
      CALL LIST(M1,N1,3,NSP)
      CALL NODES(L(NA),N1)
      RETURN
C-----SPECIFICATION OF DISPLACEMENT BOUNDARY CONDITIONS
  200 CALL DELETE(M1)
      CALL LIST(M1,NUMNP,6,1)
      CALL BOUND(L(NA),NUMNP,NEQ)
      CALL PTAPE(1,1)
      NUMEL=0
      RETURN
C-----CALCULATION OF TRUSS AND BEAM STIFFNESS MATRICES------
  300 CALL FIND(M2)
      N1=NA
      CALL FIND(M3)
      N2=NA
      CALL FIND(M4)
      NPROP=I(1)/I(3)
      N3=N1+NUMNP
      N4=N1+2*NUMNP
      IF(NOP.EQ.4) GO TO 350
C-----CALCULATION OF TRUSS ELEMENT MATRICES-----NOP=3
      ND=6
      NS=1
      CALL TRUSS(L(N1),L(N3),L(N4),L(N2),L(NA),NUMNP,NTRUSS,NPROP)
      GO TO 400
C-----CALCULATION OF BEAM ELEMENT STIFFNESS MATRICES-- NOP=4
  350 ND=12
      NS=8
      CALL BEAM(L(N1),L(N3),L(N4),L(N2),L(NA),NUMNP,NTRUSS,NPROP)
  400 CALL LIST(M1,1,5,1)
      L(NA)=NTRUSS
      L(NA+1)=ND
      L(NA+2)=NS
      L(NA+4)=NUMEL+1
      NUMEL=NUMEL+NTRUSS
      RETURN
C------FORMATION OF TOTAL STIFFNESS AND MASS MATRICES
  500 CALL DELETE(M1)
      IF(M2.NE.0) CALL DELETE(M2)
      CALL PTAPE(1,1)
      IF(M2.EQ.0) GO TO 550
      CALL LIST(M2,NEQ,1,NSP)
      N1=NA
  550 CALL LIST(M1,NEQ,NEQ,NSP)
      CALL ADDSF(L(NA),L(N1),NEQ,NUMEL,M2)
      RETURN
C-----FORMATION OF LOAD MATRIX--------------------
  600 CALL DELETE(M1)
      CALL FIND(M2)
      N2=NA
      CALL LIST(M1,NEQ,N1,NSP)
      CALL LOADS(L(N2),L(NA),NEQ,N1,NUMNP)
      RETURN
C-----EVALUATION OF MEMBER FORCES
  700 CALL FIND(M1)
      NUME=L(NA)
      ND=L(NA+1)
      NS=L(NA+2)
      NR=L(NA+4)
      CALL PTAPE(1,NR)
      CALL FIND(M2)
      N1=NA
      NL=I(3)
      NSS=0
      IF(M3.EQ.0) GO TO 1050
      NSS=NS*NUME
      CALL LIST(M3,NSS,NL,NSP)
 1050 CALL FORCE(L(N1),L(NA),NEQ,NL,NUME,NSS)
      RETURN
C-----PRINT OF NODE DISPLACEMENTS-----------------
  800 IF(NO.NE.0) RETURN
      CALL FIND(M1)
      N2=NA
      NL=I(3)
      CALL FIND(M2)
      CALL DISPL(L(N2),L(NA),NEQ,NL,NUMNP)
      RETURN
C----- LOAD INTERGER ARRAY-------------------------
  900 CALL DELETE(M1)
      CALL LIST(M1,N1,N2,1)
      CALL LOADI(L(NA),N1,N2,N3)
      RETURN
      END
```

```
      SUBROUTINE NODES(XYZ,NUMNP)
      DIMENSION XYZ(NUMNP,3)
C-----READ AND PRINT OF NODAL POINT COORDINATES---
      COMMON NO
      IF(NO.EQ.0) PRINT 2001
  100 READ  1000, N,X,Y,Z
      IF(N.EQ.0) RETURN
      XYZ(N,1)=X
      XYZ(N,2)=Y
      XYZ(N,3)=Z
      IF(NO.EQ.0)PRINT 2000, N,XYZ(N,1),XYZ(N,2),XYZ(N,3)
      GO TO 100
 1000 FORMAT (I5,6X,3F10.0)
 2001 FORMAT (8HONODE NO   14X,1HX,14X,1HY,14X,1HZ)
 2000 FORMAT(I8,3F15.3)
      END
```

```
      SUBROUTINE BOUND(ID,NUMNP,NEQ)
      COMMON NO
      DIMENSION ID(NUMNP,6),II(6)
C-----ZERO ID ARRAY-------------------------------
      DO 100 N=1,NUMNP
      DO 100 I=1,6
  100 ID(N,I)=0
C-----SPECIFICATION OF UNKNOWN DISPLACEMENTS
  200 READ 1000,NL,NH,(II(I),I=1,6),INC
      IF(NL.EQ.0) GO TO 300
      IF(INC.EQ.0) INC=1
      IF(NO.EQ.0) PRINT 1000,NL,NH,(II(I),I=1,6),INC
      DO 250 J=NL,NH,INC
      DO 250 I=1,6
  250 ID(J,I)=II(I)
      GO TO 200
C-----EVALUATION OF EQUATION NUMBERS--------------
  300 NEQ=0
      DO 400 N=1,NUMNP
      DO 380 I=1,6
      IF(ID(N,I).EQ.0) GO TO 350
      NEQ=NEQ+1
      ID(N,I)=NEQ
  350 CONTINUE
  400 CONTINUE
      IF(NO.EQ.0) PRINT 2000,(N,(ID(N,I),I=1,6),N=1,NUMNP)
      RETURN
 1000 FORMAT (9I5)
 2000 FORMAT(42H0EQUATION NUMBERS FOR NODAL DISPLACEMENTS  /
     1 35H NODE     X     Y     Z    XX    YY    ZZ /(7I5))
      END
```

```
      SUBROUTINE ADDSF(A,B,NEQ,NUMEL,M2)
      COMMON /STR/ ND,NS,NRP,LM(12),ST(8,12),XM(12),S(12,12)
      DIMENSION A(NEQ,NEQ),B(NEQ)
C
      DO 100 I=1,NEQ
      IF(M2.NE.0) B(I)=0.0
      DO 100 J=1,NEQ
  100 A(I,J)=0.0
      DO 300 N=1,NUMEL
      CALL PTAPE(3,NR)
      DO 200 I=1,ND
      II=LM(I)
      IF(II.LE.0) GO TO 200
      IF(M2.NE.0) B(II)=B(II)+XM(I)
      DO 150 J=1,ND
      JJ=LM(J)
      IF(JJ.LE.0) GO TO 150
      A(II,JJ)=A(II,JJ)+S(I,J)
  150 CONTINUE
  200 CONTINUE
  300 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LOADS(ID,R,NEQ,NL,NUMNP)
      DIMENSION R(NEQ,NL),ID(NUMNP,6)
      COMMON NO
      COMMON /TEMP/ RR(6)
      DO 100 N=1,NEQ
      DO 100 L=1,NL
  100 R(N,L)=0.0
      IF(NO.EQ.0) PRINT 2000
  200 READ 1000, N,L,RR
      IF(N.EQ.0) RETURN
      IF(NO.EQ.0) PRINT 2001, N,L,RR
      DO 300 I=1,6
      II=ID(N,I)
      IF(II.LE.0) GO TO 300
      R(II,L)=RR(I)
  300 CONTINUE
      GO TO 200
 1000 FORMAT (2I5,6F10.0)
 2000 FORMAT (12H  NODE  LOAD  13X 2HFX 13X 2HFY 13X 2HFZ 13X 2HMX
     1 13X 2HMY 13X 2HMZ )
 2001 FORMAT (2I6,6E15.6)
      END
```

```
      SUBROUTINE DISPL(U,ID,NEQ,NL,NUMNP)
      DIMENSION U(NEQ,NL),ID(NUMNP,6)
      COMMON /TEMP/UU(6)
C
      DO 200 N=1,NUMNP
      PRINT 2000
      DO 200 L=1,NL
      DO 100 I=1,6
      UU(I)=0.0
      II=ID(N,I)
      IF(II.LE.0) GO TO 100
      UU(I)=U(II,L)
  100 CONTINUE
  200 PRINT 2001,N,L,UU
      RETURN
 2000 FORMAT(12H  NODE  LOAD 13X 2HUX 13X 2HUY 13X 2HUZ 13X 2HTX
     1 13X 2HTY 13X 2HTZ )
 2001 FORMAT (2I6,6E15.6)
      END
```

```
      SUBROUTINE FORCE(D,F,NEQ,NL,NUME,NSS)
      DIMENSION D(NEQ,NL),F(NSS,NL)
      COMMON NO
      COMMON /STR/ ND,NS,NRP,LM(12),ST(6,12),XM(12),S(12,12)
      DO 600 N=1,NUME
      IF(NO.EQ.0) PRINT 2000, N,(I,I=1,NS)
      CALL PTAPE(3,NR)
      DO 500 L=1,NL
      DO 200 I=1,NS
      XM(I)=0.0
      DO 200 J=1,ND
      JJ=LM(J)
      IF(JJ.LE.0) GO TO 200
      XM(I)=XM(I)+ST(I,J)*D(JJ,L)
  200 CONTINUE
      IF(NO.EQ.0) PRINT 2001, L,(XM(I),I=1,NS)
      IF(NSS.EQ.0) GO TO 500
      M=NS*(N-1)+1
      DO 300 I=1,NS
      F(M,L)=XM(I)
  300 M=M+1
  500 CONTINUE
  600 CONTINUE
      RETURN
 2000 FORMAT (12H LOAD/FORCE I3,I10,7I16)
 2001 FORMAT (I10,8E16.6)
      END
      SUBROUTINE BEAM(X,Y,Z,ID,P,NUMNP,NUME,NP)
      DIMENSION X(NUMNP),Y(NUMNP),Z(NUMNP),ID(NUMNP,6),P(NP,7)
      COMMON NO
      COMMON /STR/ ND,NS,NRP,LM(12),ST(6,12),XM(12),S(12,12)
      COMMON /TEMP/ A(6,12),T(3,3),V(4,4)
C-----READ AND PRINT BEAM DATA----------
      NUME=0
      IF(NO.EQ.0) PRINT 2000
  100 READ 1000, N,I,J,K,NN
      IF(N.EQ.0) RETURN
      IF(NO.EQ.0) PRINT 2001, N,I,J,K,NN,(P(NN,L),L=1,7)
      NUME=NUME+1
C-----FORM 3 X 3 TRANSFORMATION MATRX ----------
      CALL VECTOR(V(1,1),X(I),Y(I),Z(I),X(J),Y(J),Z(J))
      XL=V(4,1)
      CALL VECTOR(V(1,4),X(I),Y(I),Z(I),X(K),Y(K),Z(K))
      CALL CROSS(V(1,1),V(1,4),V(1,3))
      CALL CROSS(V(1,3),V(1,1),V(1,2))
      DO 200 K=1,3
      DO 200 L=1,3
  200 T(K,L)=V(L,K)
C-----FORM DESTINATION VECTOR ----------
      DO 300 L=1,6
      LM(L)=ID(I,L)
  300 LM(L+6)=ID(J,L)
C-----FORM STIFFNESS MATRIX IN LOCAL SYSTEM-------
      DO 400 I=1,6
      DO 400 J=1,6
  400 S(I,J)=0.0
      S(1,1)=P(NN,1)*P(NN,5)/XL
      S(4,4)=P(NN,2)*P(NN,6)/XL
      Q=4.*P(NN,5)/XL
      S(5,5)=Q*P(NN,3)
      S(6,6)=Q*P(NN,4)
      Q=3./(XL*XL)
      S(3,3)=Q*S(6,6)
      S(2,2)=Q*S(6,6)
      Q=1.5/XL
      S(2,6)=-Q*S(6,6)
      S(6,2)=S(2,6)
      S(3,5)=Q*S(5,5)
      S(5,3)=S(3,5)
C-----FORM LOCAL-GLOBAL TRANSFORMATION----------
      DO 500 I=1,6
      DO 500 J=1,12
  500 A(I,J)=0.0
      DO 600 J=1,3
      A(2,J+3)=-XL*T(3,J)
      A(3,J+3)= XL*T(2,J)
      DO 600 I=1,3
      A(I,J)=-T(I,J)
      A(I+3,J+3)=-T(I,J)
      A(I,J+6)=T(I,J)
  600 A(I+3,J+9)=T(I,J)
C----- FORM GLOBAL STIFFNESS MATRIX
      DO 800 I=1,6
      DO 800 J=1,12
      ST(I,J)=0.0
      DO 800 K=1,6
  800 ST(I,J)=ST(I,J)+S(I,K)*A(K,J)
      DO 900 I=1,12
      ST(7,I)= ST(3,I)*XL-ST(5,I)
      ST(8,I)=-ST(2,I)*XL-ST(6,I)
      DO 900 J=I,12
      Q=0.0
      DO 850 K=1,6
  850 Q=Q+A(K,I)*ST(K,J)
      S(I,J)=Q
  900 S(J,I)=Q
C-----CALCULATE MASS MATRIX ----------
      XMASS=P(NN,7)*XL/2.
      V(1,1)=XMASS*P(NN,2)/P(NN,1)
      V(2,1)=XMASS*XL*XL/12.
      V(3,1)=V(2,1)
      DO 960 I=1,3
      XM(I)=XMASS
      XM(I+6)=XMASS
      XM(I+3)=0.0
      DO 950 J=1,3
  950 XM(I+3)=XM(I+3)+T(J,I)*V(J,1)
      XM(I+3)=ABS(XM(I+3))
  960 XM(I+9)=XM(I+3)
C-----STORE ELEMENT MATRICES ON LOW SPEED STORAGE--
      CALL PTAPE(2,NR)
      GO TO 100
 1000 FORMAT(6I6)
 2000 FORMAT (26H0 EL.    I    J    K    NP 9X,1HA,7X,3HJ11,7X,3HI22,7X,
     1 3HI33,9X,1HE,9X,1HG,9X,1HM )
 2001 FORMAT (6I6,1P10.2,6P10.0,F10.4)
      END
      SUBROUTINE VECTOR(V,XI,YI,ZI,XJ,YJ,ZJ)
      DIMENSION V(4)
      X=XJ-XI
      Y=YJ-YI
      Z=ZJ-ZI
      V(4)=SQRT(X*X+Y*Y+Z*Z)
      V(3)=Z/V(4)
      V(2)=Y/V(4)
      V(1)=X/V(4)
      RETURN
      END
      SUBROUTINE CROSS(A,B,C)
      DIMENSION A(4),B(4),C(4)
      X=A(2)*B(3)-A(3)*B(2)
      Y=A(3)*B(1)-A(1)*B(3)
      Z=A(1)*B(2)-A(2)*B(1)
      C(4)=SQRT(X*X+Y*Y+Z*Z)
      C(3)=Z/C(4)
      C(2)=Y/C(4)
      C(1)=X/C(4)
      RETURN
      END
      SUBROUTINE TRUSS(X,Y,Z,ID,EE,NUMNP,NTRUSS,NPROP)
      DIMENSION X(1),Y(1),Z(1),ID(NUMNP,6),EE(NPROP,3)
      COMMON /STR/ ND,NS,NRP,LM(12),ST(6,12),XM(12),S(12,12)
C------EVALUATION OF TRUSS ELEMENT MATRICES--------
      COMMON NO
      NTRUSS=0
      IF(NO.EQ.0) PRINT 2000
  100 READ 1001,M,I,J,NP,NPT
      IF(M.EQ.0) RETURN
      AREA=EE(NP,1)
      E=EE(NP,2)
      DEN=EE(NP,3)
      IF(NO.EQ.0) PRINT 2001, M,I,J,AREA,E,DEN
      DX=X(I)-X(J)
      DY=Y(I)-Y(J)
      DZ=Z(I)-Z(J)
      XL2=DX*DX+DY*DY+DZ*DZ
      XL=SQRT(XL2)
      DEN=DEN*XL/2.
      XX=E*AREA/XL
      ST(1,1)=DX/XL
      ST(1,2)=DY/XL
      ST(1,3)=DZ/XL
      ST(1,4)=-ST(1,1)
      ST(1,5)=-ST(1,2)
      ST(1,6)=-ST(1,3)
C
      DO 300 L=1,6
      YY=ST(1,L)*XX
      DO 250 K=L,6
      S(K,L)=ST(1,K)*YY
  250 S(L,K)=S(K,L)
      ST(1,L)=YY
  300 XM(L)=DEN
      DO 400 L=1,3
      LM(L)=ID(I,L)
  400 LM(L+3)=ID(J,L)
      CALL PTAPE(2,NR)
      NTRUSS=NTRUSS+1
      GO TO 100
C
 1001 FORMAT(5I5)
 2000 FORMAT( 7HONUMBER 6X 1HI 6X 1HJ 11X 4HAREA 14X 1HE 12X 3HDEN )
 2001 FORMAT (3I7,F15.3,2E15.6)
      END
      SUBROUTINE LOADI(L,NR,NC,N3)
      COMMON /TEMP/ FOR(40)
      DIMENSION L(NR,NC)
      COMMON NO
C------ SUBPROGRAM TO LOAD AND PRINT INTERGER ARRAY----
      IF(N3.NE.0) GO TO 100
      READ 1000, ((L(I,J),J=1,NC),I=1,NR)
      GO TO 200
  100 READ 1001,FOR
      READ FOR,((L(I,J),J=1,NC),I=1,NR)
  200 IF(NO.NE.0) RETURN
      DO 300 I=1,NC,20
      IH=I+19
      IF(IH.GT.NC) IH=NC
      PRINT 2000, (K,K=I,IH)
      DO 300 J=1,NR
  300 PRINT 2001,(J,(L(J,K),K=I,IH))
      RETURN
 1000 FORMAT(16I5)
 1001 FORMAT (40A2)
 2000 FORMAT (6X,20I5)
 2001 FORMAT (2116)
      END
      SUBROUTINE PTAPE(N,NR)
      COMMON /STR/ ND,NS,NRP,LM(12),ST(6,12),XM(12),S(12,12)
      GO TO (50,600,700),N
C-----POSITION TAPE----------
   50 NT=1
      IF(NR.NE.1) GO TO 100
      REWIND NT
      NRP=1
  100 IF(NRP.EQ.NR) RETURN
      ND=NR-NRP
      IF(ND.LT.0) GO TO 300
      DO 200 N=1,ND
  200 READ (NT)
      GO TO 500
  300 ND=-ND
      DO 400 N=1,ND
  400 BACKSPACE NT
  500 NRP=NR
C-----WRITE  RECORD ON LOW SPEED STORAGE---------
  600 WRITE (1) ND,NS,LM,ST,XM,S
      NRP=NRP+1
      RETURN
  700 READ  (1) ND,NS,LM,ST,XM,S
      NRP=NRP+1
      RETURN
      END
```

```
      SUBROUTINE GROUP3(LAST)
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      LOGICAL COMP
      DIMENSION IOP(3,10)
      DATA IOP(1, 1)/6HFUNG  /,IOP(1, 2)/6HSTEP  /,IOP(1, 3)/6HEIGEN /
      DATA IOP(1, 4)/6HDYNAM /,IOP(1, 5)/6HPLOT  /
      NUMOP=5
      GO TO 175
C---- READ OPERATION FROM CARD OR STORAGE  ----------------
  100 CALL INPUT
C-----INTERPRETE OPERATION ----------------------
  175 DO 200 J=1,NUMOP
      N=J
      IF (COMP(INHOL(1,1),IOP(1,J))) GO TO 300
  200 CONTINUE
      RETURN
C-----EXECUTE APPROPRIATE OPERATION -------------
  300 LAST=3
      GO TO (1,2,3,4,5),N
    1 CALL FUNG
      GO TO 100
    2 CALL STEP
      GO TO 100
    3 CALL EIGEN
      GO TO 100
    4 CALL DYNAM
      GO TO 100
    5 CALL PLOT
      GO TO 100
      END
```

```
      SUBROUTINE STEP
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,6))
      CALL FIND(INHOL(1,2))
      IF(I(3).NE.I(1)/I(3)) CALL ERCOM
      NK=NA
      N=I(3)
      CALL FIND(INHOL(1,3))
      IF(I(3).NE.I(1)/I(3)) CALL ERCOM
      NM=NA
      CALL FIND(INHOL(1,4))
      IF(I(3).NE.I(1)/I(3)) CALL ERCOM
      NC=NA
      CALL FIND(INHOL(1,5))
      NI=NA
      CALL LIST(INHOL(1,6),N,N2,NSP)
      NU=NA
      READ 1000, DEL,ALF,THE
      IF(NO.EQ.0) PRINT 2000, DEL,ALF,THE
      CALL FIND(INHOL(1,7))
      ND=NA
      CALL FIND(INHOL(1,8))
      NF=NA
      CALL FIND(INHOL(1,9))
      CALL STEPS(L(NK),L(NM),L(NC),L(NI),L(ND),L(NF),L(NU),N,N1,N2,L(NA)
     1,DEL,ALF,THE)
      RETURN
 1000 FORMAT(3F10.0)
 2000 FORMAT(2X,4HDEL=E15.6,3X4HALF=E15.6,3X,4HTME=E15.6)
      END
```

```
      SUBROUTINE STEPS(S,XM,C,UI,D,F,U,N,N1,N2,DTT,DEL,ALF,THE)
      DIMENSION S(N,N),XM(N,N),C(N,N),UI(N,3),U(N,N2),D(1),F(1)
C---- COMPUTE INTEGRATION CONSTANTS----------------------
      IF(THE.EQ.0.0) THE=1.0
      DT=THE*DTT
      A0=1./(ALF*DT*DT)
      A1=DEL/(ALF*DT)
      A2=1./(ALF*DT)
      A3=0.5/ALF-1.
      A4=DEL/ALF -1.
      A5=0.5*DT*(DEL/ALF-2.)
      A6=DTT*(1.-DEL)
      A7=DTT*DEL
      A8=(.5-ALF)*DTT**2
      A9=ALF*DTT**2
C---- FORM AND TRIANGULARIZE EFFECTIVE STIFFNESS MATRIX--------
      DO 100 I=1,N
      DO 100 J=1,N
  100 S(I,J)=S(I,J)+A0*XM(I,J)+A1*C(I,J)
      CALL SYMSOL(S,UI,N,1,1)
C---- FOR EACH TIME STEP----------------------------
      K=1
      DO 400 I=1,N2
      DO 400 J=1,N1
C       1. CALCULATE EFFECTIVE LOAD AT TIME T+DT
      K=K+1
      FF=F(K)+THE*(F(K)-F(K-1))
      DO 200 L=1,N
  200 U(L,I)=D(L)*FF
      DO 250 L=1,N
      X=A0*UI(L,1)+A2*UI(L,2)+A3*UI(L,3)
      Y=A1*UI(L,1)+A4*UI(L,2)+A5*UI(L,3)
      DO 250 M=1,N
  250 U(M,I)=U(M,I)+XM(M,L)*X+C(M,L)*Y
C       2. SOLVE FOR DISPLACEMENT AT T+DT
      CALL SYMSOL(S,U(1,I),N,1,2)
C       3. CALCULATE ACCELERATIONS AND VELOCITIES AT TIME T+DT
      DO 300 L=1,N
      A=A0*(U(L,I)-UI(L,1))-A2*UI(L,2)-A3*UI(L,3)
      DA=(A-UI(L,3))/THE
      A=UI(L,3)+DA
      V=UI(L,2)+A6*UI(L,3)+A7*A
      U(L,I)=UI(L,1)+DTT*UI(L,2)+A8*UI(L,3)+A9*A
      UI(L,3)=A
      UI(L,2)=V
  300 UI(L,1)=U(L,I)
  400 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE FUNG
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,3))
      IF(N1.LE.0) CALL ERCOM
      IF(N2.NE.0) N2=1
      N2=N2+1
      CALL LIST(INHOL(1,3),N2,N1,NSP)
      N3=NA
      CALL FIND(INHOL(1,4))
      N4=NA
      CALL FIND(INHOL(1,2))
      IF(I(1)/I(3).NE.2) CALL ERCOM
      CALL FUNGS(L(NA),L(N3),I(3),N1,N2,L(N4))
      RETURN
      END
      SUBROUTINE FUNGS(G,GG,NC,N1,N2,DT)
      DIMENSION G(2,NC),GG(N2,N1)
      T=G(1,1)
      J=1
      NCC=NC-1
      DO 200 I=1,NCC
      S=(G(2,I+1)-G(2,I))/(G(1,I+1)-G(1,I))
  100 GG(1,J)=T
      GG(N2,J)=G(2,I)+S*(T-G(1,I))
      IF(J.EQ.N1) RETURN
      J=J+1
      T=T+DT
      IF(T.LT.G(1,I+1)) GO TO 100
  200 CONTINUE
      CALL ERCOM
      END
```

```
      SUBROUTINE DYNAM
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NH,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
C-----SUBROUTINE TO EVALUATE MODAL DYNAMIC RESPONSE--------
      CALL DELETE(INHOL(1,6))
      CALL FIND(INHOL(1,2))
      N2=NA
      N=I(1)
      CALL FIND(INHOL(1,3))
      N3=NA
      IF(N.NE.I(1)) CALL ERCOM
      CALL FIND(INHOL(1,4))
      N4=NA
      IF(N.NE.I(1)) CALL ERCOM
      CALL FIND(INHOL(1,5))
      N5=NA
      CALL FIND(INHOL(1,7))
      N6=NA
      CALL LIST(INHOL(1,6),N,N1,NSP)
      CALL DYNAMS(L(N2),L(N3),L(N4),L(N5),L(NA),N1,N,L(N6))
      RETURN
      END
```

```
      SUBROUTINE DYNAMS(FQ,DAM,P,PA,X,NTIME,NM,DDT)
      DIMENSION FQ(NM),DAM(NM),P(NM),X(NM,NTIME),PA(2,1)
C     EVALUATION OF MODAL RESPONSE ... USING EXPLICIT INTEGRATION
      DO 700 M=1,NM
      W=FQ(M)
      DAMP=DAM(M)
      WW=W*W
      ZW=DAMP*W
      TZW=2.*ZW
      WD=W*SQRT(1.0-DAMP**2)
      FB=TZW/(WW*WW)
      FA=ZW/WW
      FV=ZW*WW
      FVD=WW*(2.*DAMP**2-1.0)
      FBB=(2.*DAMP**2-1.0)/WW
      L=0
      II=1
      VO=0.
      VDO=0.
      TO=PA(1,1)
C
   10 DT=DDT
   50 B=(PA(2,II+1)-PA(2,II))/(PA(1,II+1)-PA(1,II))
      B=B*P(M)
      A=P(M)*PA(2,II)+B*(TO-PA(1,II))
      TT=TO+DT
      IF(PA(1,II+1).GT.TT) GO TO 100
      DELT=PA(1,II+1)-TO
      GO TO 200
  100 DELT=DT
C
  200 EX=EXP(-ZW*DELT)
      FT=WD*DELT
      CS=COS(FT)
      SN=SIN(FT)
      VT=(VDO+ZW*VO-FA*A+FBB*B)*SN/WD
      VT=VT+(VO-A/WW+FB*B)*CS
      VT=VT*EX+A/WW-FB*B+B*DELT/WW
      VDT=(A-WW*VO-ZW*(VDO+B/WW))*SN/WD
      VDT=EX*((VDO-B/WW)*CS+VDT)+B/WW
      VDDT=(B+FV*VO+FVD*VDO-ZW*A)/WD
      VDDT=EX*((A-WW*VO-TZW*VDO)*CS+VDDT*SN)
      VO=VT
      VDO=VDT
C
      IF(PA(1,II+1).GT.TT) GO TO 500
      DT=DT-DELT
      II=II+1
      TO=PA(1,II)
      IF(DT.EQ.0.) GO TO 600
      GO TO 50
  500 TO=TO+DT
  600 L=L+1
      X(M,L)=VT
      IF(L.LT.NTIME) GO TO 10
  700 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE EIGEN
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NM,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      CALL DELETE(INHOL(1,3))
      CALL FIND(INHOL(1,2))
      IF(I(3).NE.I(1)/I(3)) CALL ERCOM
      N2=I(1)/I(3)
      N3=NA
      CALL FIND(INHOL(1,4))
      IF(I(1).NE.N2) CALL ERCOM
      N4=NA
      CALL LIST(INHOL(1,3),N2,N2,NSP)
      CALL EIGENS(L(N3),L(NA),L(N4),N2,N1,N3)
      PRINT 2000, N1,N3
      RETURN
 2000 FORMAT(13,27H FIGURE TOLERANCE SPECIFIED  / I6,
     1 20H ROTATIONS PERFORMED )
      END
      SUBROUTINE EIGENS(H,U,E,N,NS,NR)
      DIMENSION H(N,N),U(N,N),E(N)
C
      IF(NS.EQ.0) NS=4
      TEST=1.0/10.**(2*NS)
      NN=N-1
      NR=0
      NRLM=5*N**2
      TOLER=0.1
C-----NORMALIZE TO UNIT MATRIX--------------------
      DO 10 I=1,N
   10 E(I)=1.0/SQRT(E(I))
      DO 30 I=1,N
      DO 20 J=1,N
      H(I,J)=E(I)*H(I,J)*E(J)
   20 U(I,J)=0.0
   30 U(I,I)=1.0
C---- REDUCE MATRIX ------------------------------
   50 XMAX=0.0
      DO 700 II=1,NN
      JL=II+1
      DO 600 JJ=JL,N
C-----CHECK IF ROTATION IS REQUIRED  -------------
      HII=H(II,II)
      HIJ=H(II,JJ)
      HJJ=H(JJ,JJ)
      X=HIJ*HIJ/(HII*HJJ)
      IF(X.GT.XMAX) XMAX=X
      IF(X.LT.TOLER) GO TO 600
C-----COMPUTE TAN, SIN AND COS -------------------
      NR=NR+1
      HT=.5*(HII-HJJ)/HIJ
      TN=-HT*(1.+SQRT(1.+1./(HT*HT)))
      CS =1.0/SQRT (1.+TN **2)
      SN=CS*TN
      C2=CS**2
      S2=SN**2
C-----REDUCE II,JJ ELEMENT TO ZERO ---------------
      HT=2.*HIJ*CS*SN
      H(II,JJ)=0.0
      H(II,II)=HII*C2+HT+HJJ*S2
      H(JJ,JJ)=HII*S2-HT+HJJ*C2
      DO 530 I=1,N
      IF(I-II) 370,530,420
  370 HT=H(I,II)
      H(I,II)=CS *HT+SN *H(I,JJ)
      H(I,JJ)=-SN *HT+CS *H(I,JJ)
      GO TO 530
  420 IF(I-JJ) 430,530,480
  430 HT=H(II,I)
      H(II,I)=CS *HT+SN *H(I,JJ)
      H(I,JJ)=-SN *HT+CS *H(I,JJ)
      GO TO 530
  480 HT=H(II,I)
      H(II,I)=CS *HT+SN *H(JJ,I)
      H(JJ,I)=-SN *HT+CS *H(JJ,I)
  530 CONTINUE
C-----OPERATE ON EIGENVECTORS --------------------
  540 DO 550 I=1,N
      HT=U(I,II)
      U(I,II)=CS *HT+SN *U(I,JJ)
  550 U(I,JJ)=-SN *HT+CS *U(I,JJ)
  600 CONTINUE
  700 CONTINUE
C-----TEST FOR END OF ITERATION AND SET NEW TOLERANCE
      IF(NRLM.LT.NR) GO TO 1000
      IF(XMAX.LT.TEST) GO TO 710
      TOLER=0.10*XMAX
      GO TO 50
C-----NORMALIZE AND ORDER EIGENVECTORS ------------
  710 DO 800 I=1,N
      DO 750 J=1,N
  750 U(I,J)=U(I,J)*E(I)
  800 E(I)=H(I,I)
      DO 900 I=1,NN
C-----ORDER EIGENVALUES AND EIGENVECTORS ----------
      JL=I+1
      HT=E(I)
      IM=I
      DO 850 J=JL,N
      IF(HT.LT.E(J)) GO TO 850
      HT=E(J)
      IM=J
  850 CONTINUE
      E(IM)=E(I)
      E(I)=HT
      DO 900 J=1,N
      HT=U(J,I)
      U(J,I)=U(J,IM)
  900 U(J,IM)=HT
      RETURN
C
 1000 PRINT 2000
      RETURN
 2000 FORMAT(41HOITERATION TERMINATED WITHOUT CONVERGENCE )
      END
```

```
      SUBROUTINE PLOT
      COMMON NO,MTOT,NARA,NSIZE,NSP,NDP,NM,NDIR,L(1000)
      COMMON /CARD/ INHOL(3,10),N1,N2,N3,N4,S1,S2
      COMMON /ADIR/ NA,I(10)
      DATA IPLOT /6HPLOTNM /
      NT=121+N1+N1
      CALL LIST(IPLOT,1,NT,1)
      NP=NA
      NK=NA+121
      NN=NK+N1
      CALL FIND(INHOL(1,2))
      CALL PLOTS(L(NA),L(NP),L(NK),L(NN),I(1)/I(3),I(3),N1)
      CALL DELETE(IPLOT)
      RETURN
      END
      SUBROUTINE PLOTS(A,K,KODE,NROW,NR,NC,N1)
      DIMENSION A(NR,NC),K(121),KODE(N1),NROW(N1)
      COMMON NO
C---- SUBROUTINE TO PRINTER PLOT N1 ROWS OF ARRAY A
      READ  1000, (KODE(I),NROW(I),I=1,N1)
      IF(NO.NE.0) RETURN
      PRINT 2000, (KODE(I),NROW(I),I=1,N1)
C----- LOCATE LARGEST AND SMALLEST ELEMENTS--------
      II=NROW(1)
      AL=A(II,1)
      AS=AL
      DO 100 I=1,N1
      II=NROW(I)
      DO 100 J=1,NC
      AX=A(II,J)
      IF(AX.GT.AL) AL=AX
      IF(AX.LT.AS) AS=AX
  100 CONTINUE
      AX=(AL-AS)/120.
      PRINT 2001,AX,AS,AL
      KO=-AS/AX +1.
      DO 500 I=1,NC
C---- FILL LINE BUFFER WITH BLANKS AND CODES---
      DO 300 J=1,121
  300 K(J)=1H
      K(1)=1HI
      K(121)=1HI
      IF(KO.GT.0) K(KO)=1H0
      DO 400 J=1,N1
      JJ=NROW(J)
      II=(A(JJ,I)-AS)/AX+1.
  400 K(II)=KODE(J)
  500 PRINT 2003,K
      PRINT 2004
      RETURN
 1000 FORMAT (1A1,I4)
 2000 FORMAT (11HOSYMBOL ROW /(5X,1A1,I6))
 2001 FORMAT (11H ONE SPACE= E15.6/9HOMINIMUM= E15.6,7HX 9HMAXIMUM=
     1 E15.6/  122(1H0))
 2003 FORMAT(1H0,121A1)
 2004 FORMAT(122(1H0))
      END
```