REPORT NO.
UCB/SESM-86/05

STRUCTURAL ENGINEERING AND
STRUCTURAL MECHANICS

CAL - 86
COMPUTER ASSISTED LEARNING OF STRUCTURAL ANALYSIS AND THE CAL/SAP DEVELOPMENT SYSTEM

THE CAL SERIES OF PROGRAMS

BY

EDWARD L. WILSON

AUGUST 1986
Revised
AUGUST 1989

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

B A S I C C A L - 8 9 O P E R A T I O N S

```
HELP (H)           LIST CAL COMMANDS
LOAD M1 R=? C=?    LOAD MATRIX M1 OF REAL NUMBERS
ZERO  M1+ NR=? NC=? ZERO A REAL MATRIX M1
LIST (L)           LIST THE  DIRECTORY OF ALL ARRAYS
PRINT (P)  M1      LIST ARRAY NAMED "M1"'
SAVE OR STOP (S)   TERMINATE PROGRAM AND SAVE DATABASE
QUIT (Q)           TERMINATE PROGRAM
START              NEW PROBLEM NAME
DELETE (D) M1-     DELETE ARRAY NAMED "M1"'
MODIFY M1-         MODIFY TERMS IN MATRIX M1
RESUME or READC    READ INCORE DATA BASE FROM PREVIOUS RUN
WRITE M1           WRITE ARRAY M1 TO DISK
READ  M1           READ ARRAY M1 FROM DISK
RUN                EXECUTE OPERATIONS  FROM INPUT FILE
RETURN             RETURNS TO INTERACTIVE MODE
```

S T A N D A R D M A T R I X O P E R A T I O N S

```
MULT M1 M2 M3+    MULTIPLY M1 * M2 = M3
TMULT M1 M2 M3+   TRANSPOSE OF M1 * M2 = M3
ADD M1- M2        ADD MATRIX M2 TO MATRIX M1
SUB M1- M2        SUBTRACT MATRIX M2 FROM MATRIX M1
TRAN M1 M2+       TRANSPOSE MATRIX M1 TO FORM MATRIX M2
DUP M1 M2+        DUPLICATE MATRIX M1 TO MATRIX M2
STODG M1- M2      STORES MATRIX M2 ON DIAGONAL OF MATRIX M1
DUPDG M1 M2+      DUPLICATES DIAGONAL OF M1 TO MATRIX M2
SCALE M1- M2      SCALE MATRIX M1 BY THE TERM M2(1,1)
INVERT M1-        INVERSION OF SYMMETRIC MATRIX "M1"
SOLVE M1- M2- S=?  SOLVE M1 x  = M2
                   S=0  SOLVE Ax = B
                   S=1  TRIANGULARIZE M1 ONLY
                   S=2  FORWARD SUBSTITUTE ONLY
                   S=3  BACK SUBSTITUTE ONLY
STOSM M1 M2 L=L1,L2      STORES MATRIX M2 IN MATRIX
                         M1 AT LOCATION M1(L1,L2)
DUPSM M1 M2+ NR=? NC=? L=L1,L2  DUPLICATES SUBMATRIX M2
                FROM LOCATION M1(L1,L2)   M2 IS NR x NC
```

D I R E C T S T I F F N E S S O P E R A T I O N S

```
SLOPE  Ki+      E=? I=? L=?  FORMS 4 X 4 STIFFNESS MATRIX
FRAME  Ki+ Ti+ Gi+  E=? I=? A=? S=?  X=?,? Y=?,?  P=?
       GEOMETRIC STIFF. MATRIX Gi IS FORMED IF P NOT ZERO
TRUSS  Ki+ Ti+ E=? A=? N=Ni,Nj  FORMS TRUSS STIFF.
FRAME3 Ki+ Ti+ E=? A=? I=I3,I2 J=? N=Ni,Nj P=P1,P2
LOADI  ID R=? C=? LOAD ARRAY "ID" OF INTEGER NUMBER
ADDK   K+ Ki ID N=?  ADD ELEMENT STIFFNESS TO TOTAL STIFFNESS
MEMFRC Ki U ID Fi+ N=?  EVALUATION OF MEMBER FORCES
```

S T R U C T U R A L D Y N A M I C O P E R A T I O N S

```
EIGEN  K- V+ M- EIGENVALUES OF KV = MVe  - DIAGONAL MASS
JACOBI K- V+ M- e EIGEN SOLUTION FOR FULL MASS MATRIX
SQREL  M1-      REPLACES EACH TERM OF M1 WITH ITS SQUARE ROOT
INVEL  M1-      REPLACES EACH TERM OF M1 WITH ITS INVERSE
DYNAM  W C F G(t) X(t) DT=? N=?   UNCOUPLED DYNAMIC RESPONSE
NORM   M1 M2+ T=?    FORMS COLUMN MATRIX M2 WHERE
               Where T=0 SUM ABS-VALUES OF ROWS
                     T=1 SRSS OF THE ROWS
MAX  X Xmax FORMS Xmax FROM MAXIMUM ABS. VALUES OF ROWS OF X
STEP K- M C UVA- U+ P F(t) DT=? L=Li,Lmax P=deta,alpha,theta
PLOT M1 N=? R=R1,R2,.. S=S1,S2,.. PLOTS "N" ROWS MATRIX M1
     Where Ri = THE ROWS TO PLOT and Si = SYMBOLS FOR ROW i
RITZ K- M R V+  NV=?  S=?
           NV= # OF RITZ VECTORS TO BE GENERATED
            S= NONZERO IF STATIC VECTOR IS NOT RETAINED
DFT  F(T)- DT=? DISCRETE FOURIER TRANS.-F(T) REPLACED BY F(W)
IDFT F(W)-     F(W) REPLACED BY F(T)
RADIUS F(W)  R(W)+
FSOLVE W C F G(W) Y(W)+ DT=? FREQUENCY DOMAIN SOLUTION
```

L O O P I N G O P E R A T I O N S

```
LOOP SEP N=?   EXECUTE ALL OPERATIONS BEFORE SEPARATOR "SEP" ON
               INPUT FILE. N = NUMBER OF TIMES TO SUBMIT (DEFAULT=1)

IF M1  M2      TERMINATES LOOP IF M1 IS LESS THAN M2
                (PLACE BEFORE "SEP" LINE)
```

C A L - S 6

COMPUTER ASSISTED LEARNING

OF

STRUCTURAL ANALYSIS

and

THE CAL/SAP DEVELOPMENT SYSTEM

THE CAL SERIES OF PROGRAMS

BY

EDWARD L. WILSON

UNIVERSITY OF CALIFORNIA

BERKELEY, CALIFORNIA

AUGUST 1986

Revised August 1989

ACKNOWLEDGEMENTS

CAL-86 is the latest version of a series of computer programs for the Computer Assisted Learning of Structural Analysis which have been developed by the author. The first program was a matrix interpretive program which was developed in 1960 under the direction of Professor Ray Clough. The author would like to take this opportunity to thank Professor Clough for his continued support and suggestions for the improvement of the program during the past 26 years.

Several graduate students have contributed to the development of CAL-86. Martin Button, John Dickens and Eduardo Bayo programmed the frequency domain commands. Marc Hoit participated in the development and verification of CAL-80 and the CALSAP development system. Pierre Leger added the Ritz vector subroutines. Since all commands have been reprogrammed for this version of the program the author is responsible for all errors which may exist in the CAL-86 program or the users manual.

CAL-86 was developed as part of the teaching responsibilities of the author at the University of California, Berkeley, without external financial support. Research on the use of Ritz vectors in earthquake engineering was funded by the National Science Foundation.

SUMMARY

The basic purpose of the CAL language is to bridge the gap between traditional methods of teaching structural analysis and the use of automated structural analysis programs. As a result of using CAL it is hoped that engineers will understand the theory and approximations which are used in modern structural analysis programs.

CAL is a computer program which is designed to interpret a sequence of commands which are supplied by the user. The commands can be given directly in an "interactive mode" or the program can read the commands from a "batch data file". Commands for matrix analysis, direct stiffness structural analysis and dynamic response analysis are possible.

The program is written in standard FORTRAN 77 and will operate on small micro or large mainframe computer systems. Therefore, the previous version of the program, CAL-78, has been significantly rewritten and additional commands have been added. Also, the input has been redesigned in which all commands, array names and data are in a free-field form. The program is based on the use of the CALSAP development system. The FORTRAN listing of all programs is given in order for the user to verify the exact numerical method which is used within the program.

The specific version of the FORTRAN source statements, which are given in this manual, are for MS-DOS microcomputer systems and can be directly compiled with the Microsoft FORTRAN Compiler Version 3.31.

Since the program can be easily modified and new CAL commands added, the program can be used as an effective research tool. New numerical algorithms for the static or dynamic analysis of structures can be added and tested within a few hours.

TABLE OF CONTENTS

THE CAL LANGUAGE

FORM OF THE CAL COMMANDS

CAL commands and data can be entered interactively or supplied within a data input file which is prepared by an editor such as WORDSTAR or EDLIN. The user must specify the "name" of the input file which contains the CAL operations. CAL commands which are contained within the file are executed by the SUBMIT command which is entered interactively. The results of every CAL run are saved on the output file name "name.OUT" which may be displayed, printed or examined with an editor. If no file name is specified, as is the case of completely interactive use, a default name of "I" is used.

If a CAL run is terminated by an error or the STOP command all arrays which are within the computer storage are saved on the file "name.COR". The CAL program can then be restarted, with the same problem "name", with the READC command. The LIST and PRINT commands can then be used interactively to examine the data arrays contained within the computer storage.

Data on a "command line" must be separated by commas, or, one or more blanks. A typical CAL command line has the following form:

$$OP\ M_1\ M_2\ --\ A=a_1,a_2\ -\ B=?\ :(Comment)$$

Where "OP" is the name of the CAL command; and, "M_i" is a one to <u>four</u> character array name.

The notation M_i^+ indicates that the array will be created by the operation. If the array name for new data has previously been used the old array will be eliminated before the operation is executed. The notation M_i^- indicates that the array has been modified by the CAL operation.

"a_i" is data to be used by the operation and can be in either integer or floating point form. In the case of floating point numbers, they can have the form of arithmetric statements. For example, 2.5+4*2-6/2 will be interpreted as ((2.5+4)*2 - 6)/2.

A "C" in column one of a command or data line indicates that the line will be a comment line which is used to clarify input information.

SUMMARY OF BASIC CAL COMMANDS

The following list of CAL commands controls the flow of execution and allows for input or generation of arrays within the computer storage:

HELP or H

If the HELP command is executed in the interactive mode a list of all possible CAL commands will be displayed.

STOP or S

The STOP command will terminate the execution of the program and return control to the computer's operating system. All arrays which are contained in the computer's storage will be saved in the file "name.COR" where "name" is the problem (input file) name which has been specified by the user.

READC

The READC command reads all arrays from the file "name.COR" and allows the user to continue to enter CAL commands from the point the previous run was terminated.

LIST or L

If the LIST command is executed a list of the name and size of all arrays which are contained in the computer storage is displayed.

LOAD M_1^+ R=? C=?

The LOAD command will create a matrix named "M_1" with "R" rows and "C" columns. The data must immediately follow the LOAD command. The data must be supplied one row per line. The data is separated by commas, or, one or more blanks. A line of data may be continued by the use of a "\" at the end of the first line. If the data for a row is greater than 160 characters the matrix must be loaded by the use of submatrix operations.

ZERO M_1^+ R=? C=? T=? D=?

The ZERO command will create a R x C matrix named M_1. If "T=?" is specified all terms of the matrix will be set to "T". If the matrix is square the diagonal terms will be set to "D".

PRINT or P M_1

The PRINT command will cause the matrix "M_1" to be displayed on the terminal or transferred to the "name.OUT" file if batch input is used.

DELETE or D M_1^-

The array named M_1 will be deleted and the storage within the computer will be compacted.

MODIFY M_1^-

The MODIFY command can be used interactively to modify individual terms in the matrix named M_1.

DUP M_1 M_2^+

The DUP command forms a new matrix M_2 which is identical to the matrix M_1.

RUN

RUN will cause the input commands to the CAL program to be obtained from the Input File Name that is specified.

RETURN

The RETURN command will terminate the execution of the batch input RUN mode and return the CAL program to the interactive mode.

IF M_1 M_2

If the absolute value of $M_1(1,1)$ is less than $M_2(1,1)$ the RETURN command is executed and the RUN operation is terminated.

QUIT

Execution of CAL is terminated. The incore data base is not saved.

SUMMARY OF MATRIX OPERATION COMMANDS

ADD M_1^- M_2

The ADD operation replaces the matrix M_1 with $M_1 + M_2$

SUB M_1^- M_2

The SUB operation replaces the matrix M_1 with $M_1 - M_2$

MULT M_1 M_2 M_3^+

The MULT command creates the matrix M_3 which is the product of the matrices M_1 and M_2. Or **TMULT** where M_1 is stored in transposed form.

(The number of numerical operations required for matrix multiplication is N M L ; where, M_1 is a N x M matrix and M_2 is a M x L matrix.)

TRAN M_1 M_2

The TRAN command forms the matrix M_2 which is the transpose of the matrix M_1.

SCALE M_1^- M_2

The SCALE command multiplies each term in matrix M_1 by $M_2(1,1)$.

SOLVE M_1^- M_2^- [S=? EQ=?]

The SOLVE command operates on the matrix equation $M_1 x = M_2$ where M_1 is a symmetric matrix. The following options are possible:

S=0 The matrix M_1 is triangularized and M_2 is replaced by the solution matrix "x".

S=1 The matrix M_1 is triangularized only.

S=2 The matrix M_2 is reduced only - M_1 must have been previously triangularized.

S=3 The matrix M_2 is replaced by the solution matrix "x" by backsubstitution only.

EQ= The number of equations to be reduced - to be used in substructure analysis.

Any nonsingular set of equations can be made symmetric if both sides of the equation are multiplied by the transpose of M_1.

(The number of numerical operations required to triangularize the N x N matrix M_1 is $N^3/6$. The number of operations required for forward reduction is $N^2L/2$ and for backsubstitution is $N^2L/2$; where, L is the number of columns in the matrix M_2.)

INVERT M_1^-

The symmetric matrix M_1 is replaced by its inverse.

(The number of numerical operations required to invert a symmetric matrix is $N^3/2$.)

SUBMATRIX OPERATIONS

DUPSM M_1 M_2^+ R=? C=? L=L_1,L_2

The command DUPSM creates a new matrix M_2 with "R" rows and "C" columns. The term $M_2(1,1)$ is identical to the term $M_1(L_1,L_2)$.

STOSM M_1^- M_2 L=L_1,L_2

The command STOSM stores the submatrix M_2 in matrix M_1. The term $M_2(1,1)$ is located at row L_1 and column L_2 in matrix M_1.

DUPDG M_1 M_2^+

The command DUPDG creates a row matrix M_2 from the diagonal terms of matrix M_1.

STODG M_1^- M_2

The command STODG stores the row matrix M_2 on the diagonal of the matrix M_1.

EXAMPLE OF BASIC CAL COMMANDS

The set of equations shown below must be solved. Since the SOLVE command only solves symmetrical systems we must use additional operations to make the system symmetrical.

$$\begin{bmatrix} 0.0 & 3.4 & -2.0 \\ 4.0 & -1.0 & 0.0 \\ 0.0 & 6.0 & 4.0 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -1.0 \\ 4.0 \end{bmatrix}$$

An input file of the following form must be prepared:

```
C  SOLUTION OF EQUATIONS EXAMPLE
C
EX1
LOAD A  R=3  C=3
0     3.4     2.0
4    -1.0       0
0     6          4
PRINT A
LOAD B  R=3 C=1
3
-1
4
PRINT B
TMULT A A ATA
TMULT A B X
SOLVE ATA X
PRINT X :RESULTS
C CHECK RESULTS
MULT A X C
SUB C B
PRINT C
C Each values in this matrix should be very close to zero
RETURN
```

The CAL program will first ask for the name of the above input file. The above data is then executed by the RUN command.

DIRECT STIFFNESS COMMANDS

The direct stiffness operations allow for the automatic formulation of element stiffness matrices and the direct addition of element stiffnesses to form the global stiffness matrix. The first step in the use of these commands is for the user to identify all displacement degrees of freedom at the joints of the structural system. These displacements should be numbered $u_1, u_2, - - - - u_N$. The corresponding external joint loads will be $R_1, R_2 - - - - R_N$. The structural members should also be numbered from 1 to M.

The user must prepare an integer table which identifies the joint equilibrium equation numbers, 1 to N, for each internal member force. If each member has "I" possible member forces, then the integer table will be an "I" by "M" array. This array must be loaded by the LOADI command.

The ADDK command is used for each member to add the element stiffness matrices to the total (global) stiffness matrix. After the joint loads are defined, the joint equilibrium equations are solved for joint displacements by the SOLVE command. The use of the MEMFRC command for each member allows member forces to be calculated.

ADDK K^- K_i ID N=?

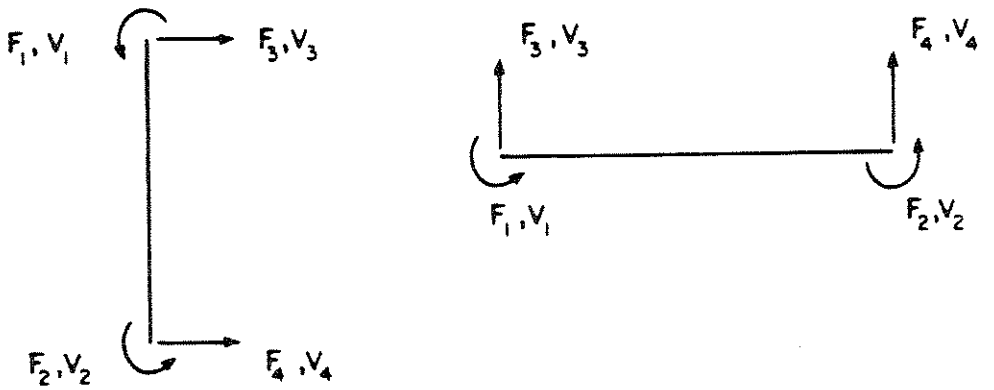
The element stiffness matrix named "K_i" is added to the total stiffness matrix named "K". The row and column numbers where the terms are to be added are obtained from the "N" column of the integer array named "ID".

MEMFRC T U ID P^+ N=?

The member forces are evaluated by the multiplication of the matrix named "T" by the joint displacement matrix named "U" and the results are stored in a matrix named "P_i". The joint displacements which are to be used in multiplication are obtained from the "N" column of the integer array named "ID" . If "T" is the element stiffness matrix the member forces are given according to the global sign convention. If "T" is a special member force-displacement transformation matrix the member forces will be given in a local member coordinate system.

SLOPE M_1 E=? I=? L=?

The slope command forms the 4 x 4 member stiffness matrix M_1 for a beam or a column. Where "E" equals the modulus of elasticity, "I" equals the moment of inertia and "L" equals the length of the member. The positive definition of member forces and displacements are shown below.



For this sign convention the classical slope-deflection equations can be written as

$$F_1 = (EI/L)\,[\,4v_1 + 2v_2 + 6\,(v_3 - v_4)\,/\,L\,]$$

$$F_2 = (EI/L)\,[\,2v_1 + 4v_2 + 6\,(v_3 - v_4)\,/\,L\,]$$

$$F_3 = -F_4 = (F_1 + F_2)\,/\,L$$

These equations can be written as the following matrix equation:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = EI/L \begin{bmatrix} 4 & 2 & 6/L & -6/L \\ 2 & 4 & 6/L & -6/L \\ 6/L & 6/L & 12/L^2 & -12/L^2 \\ -6/L & -6/L & -12/L^2 & 12/L^2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

Or symbolically, $\mathbf{F} = \mathbf{K}v$; where, $\mathbf{K}$ is the 4 x 4 stiffness matrix formed by the SLOPE command.

FRAME K T I=? A=? E=? X=X_i,X_j Y=Y_i,Y_j

The FRAME command forms the 6 x 6 element stiffness matrix named "K" and a 4 x 6 force-displacement matrix named "T" for a general two-dimensional bending member with axial deformations included in the formulation. The properties of the member are given as

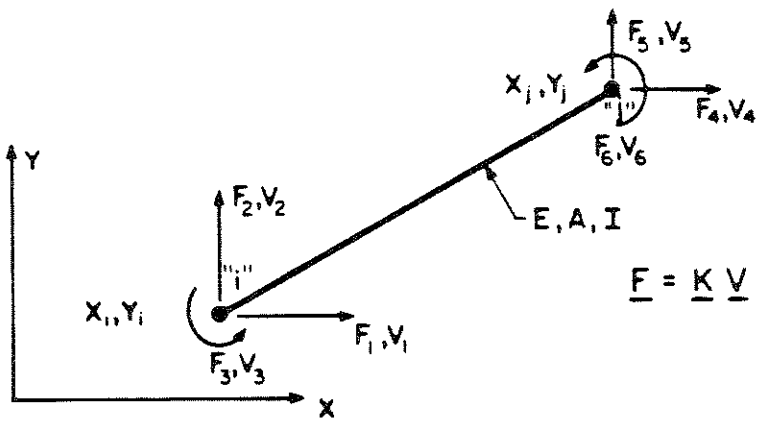
I = the Moment of Inertia of the member,

A = the Axial Area of the member, and

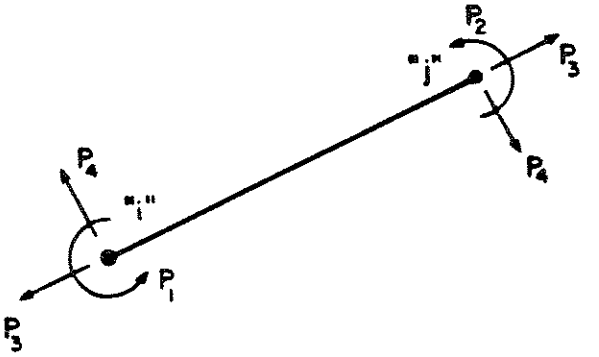
E = the Modulus of Elasticity of the member.

The coordinates of the "i" and "j" ends of the member are defined by X_i,Y_i and X_j,Y_j respectively. Note that the user is responsible for the definition of the "i" and "j" ends of the member.

The element stiffness matrix, "K", is formed with respect to the positive definition of global forces and displacements as shown below.



The member forces, with respect to the member's local coordinate system, can be evaluated by the use of the MEMFRC operation which multiplies the matrix "T" by the joint displacements. The positive definition of the member forces in the local coordinate system is shown below. The MEMFRC command will evaluate the local member forces in the order P_1 - - P_4.



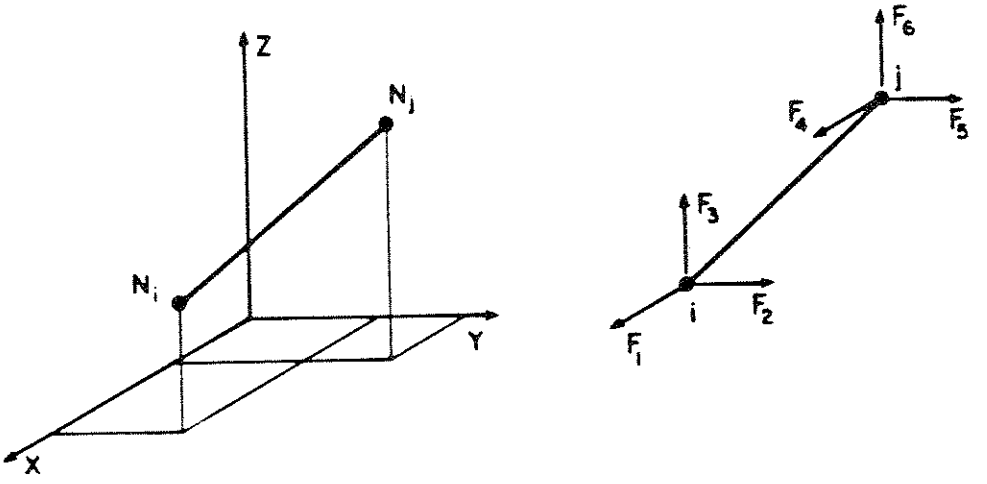
TRUSS K T A=? E=? N=Ni,Nj

The TRUSS command forms the 6 x 6 element stiffness matrix named "K" and a 1 x 6 force-displacement matrix named "T" for a general three-dimensional member with axial deformation only included in the formulation. The properties of the member are given as

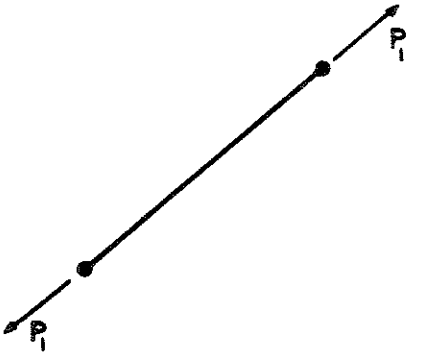
A = the Axial Area of the member

E = the Modulus of Elasticity of the member

The coordinates of joint numbers Ni and Nj must have been previously loaded in a NT x 3 array named "XYZ". Where NT is the number of joint coordinates. Therefore, Ni and Nj refer to the row numbers in the "XYZ" array. The element stiffness matrix, "K", is formed with respect to the positive definition of global forces and displacements as shown below.



The member axial force can be evaluated by the use of the MEMFRC operation which multiplies the matrix "T" by the joint displacements. A positive axial force indicates tension.



FRAME3 K T I=I33,I22 A=? J=? E=? G=? N=Ni,Nj P=P1,P2

The FRAME3 command forms the 12 x 12 element stiffness matrix named "K" and an 8 x 12 force-displacement matrix named "T" for a general three-dimensional member with axial, bending and torsional deformations included in the formulation. The properties of the member are given as

I33 = the Moment of Inertia about the 3-axis

I22 = the Moment of Inertia about the 2-axis

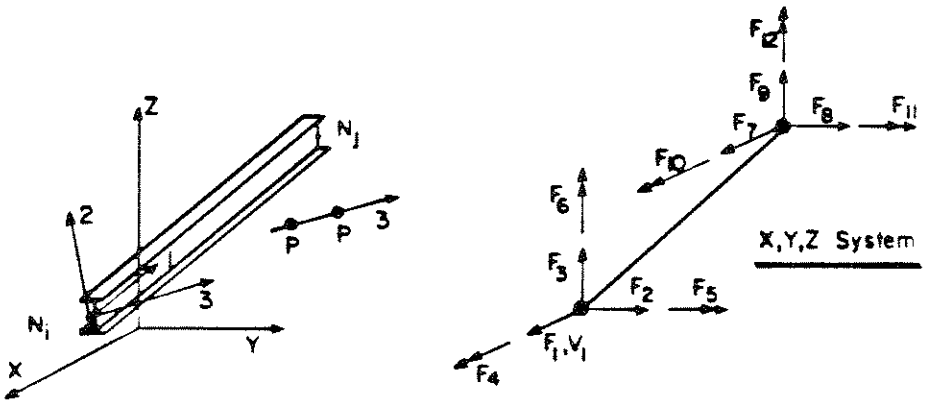
J = the Torsional Moment of Inertia about the 1-axis

A = the Axial Area of the member

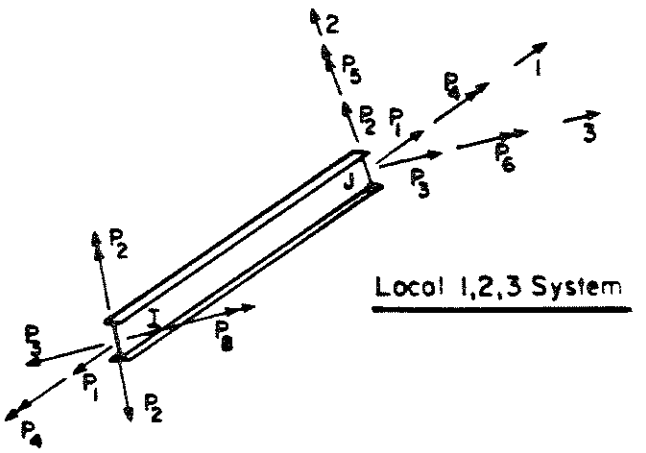
G = the Shear Modulus, and

E = the Modulus of Elasticity of the member.

The coordinates of joint number Ni, Nj, P1 and P2 must have been previously loaded in an array named "XYZ". The element stiffness matrix, "K", is formed with respect to the positive definition of global forces and displacements as shown below.



The member forces, with respect to the member's local coordinate system, can be evaluated by the use of the MEMFRC operation which multiplies the matrix "T" by the joint displacements. The positive definition of the member forces in the local coordinate system is shown below. The MEMFRC command will evaluate the local member forces in the order P_1 - - P_8.



The section properties I_{22} and I_{33} of a three-dimensional frame member must be specified with respect to a 1-2-3 local member coordinate system. In addition, member forces, which are produced by the computer programs, are defined in reference to this local system. Therefore, it is the user's responsibility to define the member 1-2-3 system in reference to the global x-y-z system. Both systems must be right-hand coordinate systems.

The positive 1-axis, V_1 vector, is defined by a line along the axis of the member from joint "I" to joint "J".

The 2 and 3-axes can be specified, with the $P=p_1,p_2$ option, by any one of the following three methods:

METHOD 1 - GLOBAL PLANES ONLY - P=?,0

xy plane	P=1,0	3-axis is Z-axis	and	$V_2 = V_3 \times V_1$
zx plane	P=2,0	3-axis is Y-axis	and	$V_2 = V_3 \times V_1$
yz plane	P=3,0	3-axis is X-axis	and	$V_2 = V_3 \times V_1$

METHOD 2 - SPECIFICATION OF "V_p" VECTOR - P=P1,P2

The coordinates of joint numbers p_1 and p_2 are specified by the user in the joint coordinate information. The vector V_p is defined by the line from joints p_1 to p_2. The 2 and 3-axes are then calculated as follows:

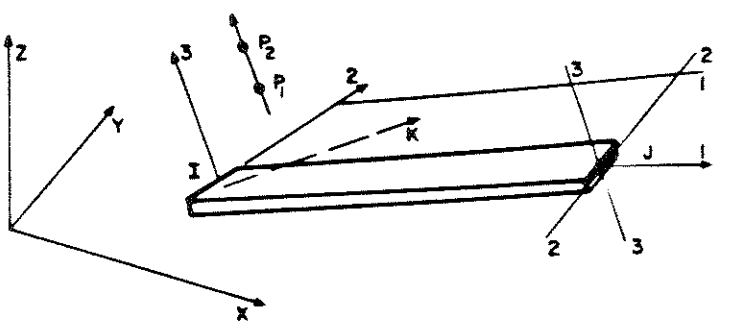
$$V_2 = V_p \times V_1$$
$$V_3 = V_1 \times V_2$$

If necessary, additional (dummy) joints may be added which are not connected to members.

METHOD 3 - SPECIFICATION OF "k" JOINT - P=0,k

The V_k vector is defined by the line from joint "I" to joint "k". The 3 and 2-axes are then calculated as follows:

$$V_3 = V_1 \times V_k$$
$$V_2 = V_3 \times V_1$$



DIRECT STIFFNESS METHOD EXAMPLE - TWO-DIMENSIONAL FRAME

The two-dimensional frame structure shown below was selected to illustrate the direct stiffness operations contained within the CAL program. Different element types can be used within the same problem.

MEMBER PROPERTIES

Members 1 to 4

$I = 1000\ in^4$

$A = 20\ in^2$

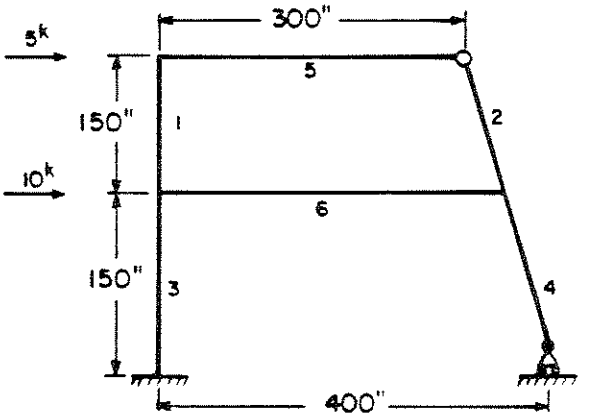
$E = 30{,}000\ ksi$

Members 5 & 6

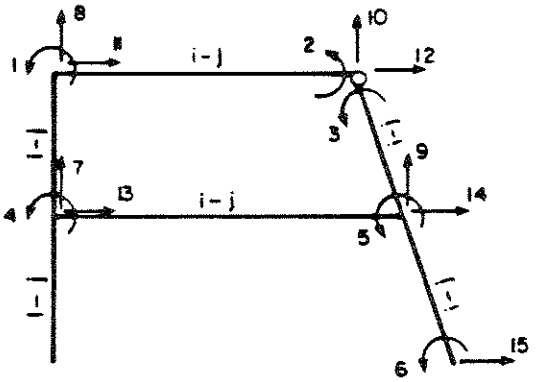
$I = 2000\ in^4$

$A = 30\ in^2$

$E = 30{,}000\ ksi$



STRUCTURAL DIMENSIONS, PROPERTIES AND LOADS



NUMBERS ASSIGNED TO EXTERNAL LOADS "R" AND JOINT DISPLACEMENTS "U"

Member Force	Member					
	1	2	3	4	5	6
F1	13	14	0	15	11	13
F2	7	9	0	0	8	7
F3	4	5	0	6	1	4
F4	11	12	13	14	12	14
F5	8	10	7	9	10	9
F6	1	3	4	5	2	5

SUMMARY OF EQUILIBRIUM EQUATIONS

SAP-86 INPUT DATA FILE - TWO-DIMENSIONAL FRAME

```
2DEX : SEPARATOR LINE
C FORMATION OF ELEMENT MATRICES
FRAME K1 T1 I=1000 A=20 E=30000  X=0,0      Y=150,300
FRAME K2 T2 I=1000 A=20 E=30000  X=350,300  Y=150,300
FRAME K3 T3 I=1000 A=20 E=30000  X=0,0      Y=0,150
FRAME K4 T4 I=1000 A=20 E=30000  X=400,350  Y=0,150
FRAME K5 T5 I=2000 A=30 E=30000  X=0,300    Y=300,300
FRAME K6 T6 I=2000 A=30 E=30000  X=0,350    Y=150,150
C LOAD LOCATION ARRAY
LOADI LM R=6 C=6
 13 14  0 15 11 13
  7  9  0  0  8  7
  4  5  0  6  1  4
 11 12 13 14 12 14
  8 10  7  9 10  9
  1  3  4  5  2  5
PRINT LM
C FORMATION OF GLOBAL STIFFNESS MATRIX
ZERO K R=15 C=15
ADDK K K1 LM N=1
ADDK K K2 LM N=2
ADDK K K3 LM N=3
ADDK K K4 LM N=4
ADDK K K5 LM N=5
ADDK K K6 LM N=6
C ENTER LOAD MATRIX
LOAD RT R=1 C=15
0 0 0 0 0 0 0 0 0 0 5 0 10 0 0
TRAN RT R
P R
C SOLVE EQUILIBRIUM EQUATIONS
SOLVE K R
P R
C CALCULATE MEMBER FORCES - LOCAL SYSTEM
MEMFRC T1 R LM F1 N=1
P F1
MEMFRC T2 R LM F2 N=2
P F2
MEMFRC T3 R LM F3 N=3
P F3
MEMFRC T4 R LM F4 N=4
P F4
MEMFRC T5 R LM F5 N=5
P F5
MEMFRC T6 R LM F6 N=6
P F6
C CALCULATE MEMBER FORCES - GLOBAL SYSTEM
MEMFRC K4 R LM G4 N=4
P G4
RETURN TO INTERACTIVE MODE
```

COMMANDS FOR DYNAMIC ANALYSIS

In this section several commands are presented which allow CAL-80 to perform linear dynamic analysis of small structural systems. With the aid of other CAL commands it is possible to solve the following types of dynamic problems:

1. Evaluation of free-vibration mode shapes and frequencies.
2. Automatic generation of Ritz vectors to be used in a mode superposition analysis or response spectra analysis.
3. Mode superposition analysis due to arbitrary loading.
4. Response spectra analysis due to earthquake loading.
5. Step-by-step analysis of structural systems with arbitrary viscous damping.
6. Dynamic analysis in the frequency domain

All commands assume that the mass and stiffness matrices have been calculated by other CAL commands. The PLOT command can be used to produce printer plots of results.

EIGEN K^- V^+ M^-

This command solves the following eigenvalue problem for the mode shapes and frequencies:

$$\mathbf{KV} = \mathbf{MV}\boldsymbol{\omega}$$

Where "K" is the name of the N x N stiffness matrix $\mathbf{K}$. The command is restricted to a diagonal mass matrix; therefore, the array named "M" must be given as a row or column array of the diagonal terms of the N x N mass matrix $\mathbf{M}$.

The N x N matrix $\mathbf{V}$, which contains all the eigenvectors (mode shapes) stored columnwise, is named "V" and is normalized in order that $\mathbf{V}^T\mathbf{M}\mathbf{V} = \mathbf{I}$.

The N x N matrix $\boldsymbol{\omega}$ is a diagonal matrix of eigenvalues w_i^2 (frequencies w_i are in radians per sec.^{2}). The EIGEN command stores the eigenvalues e_i in place of the mass terms M_i in the array named "M".

The program uses the standard Jacobi method; therefore, both $\mathbf{K}$ and $\mathbf{M}$ must be symmetric and positive definite matrices.

SQREL M_1^-

The SQREL command replaces each term in matrix M_1 with the square root of the term.

INVEL M_1^-

The INVEL command replaces each term in matrix M_1 with the inverse of the term.

DYNAM W C F G(t) X(t)$^{+}$ DT=? N=?

This command evaluates a set of "I" uncoupled second order differential equations which are generated in the mode superposition analysis of a structural system. The typical equation is of the following form:

$$\ddot{X}_i + 2 c_i w_i \dot{X}_i + w_i^2 X_i = f_i g(t) ; \quad i = 1,---I$$

Where

W is a row or column array of the frequencies w_i in radians per second.

C is the name of a row or column array of the damping ratios c_i.

F is the name of I x 1 column array of the terms f_i.

G is the name of a 2 x M array which can be used to define the time function g(t).

X(t) is the name of the I x N array where the results are stored.

DT is the time increment for which the results are produced.

The array G defines a time function in terms of straight line segments where G(1,J) defined the time t_j and G(2,J) is the value $g(t_j)$. The time function must be defines in the range T = 0 to Tmax. Where Tmax = N x DT. Therefore, the maximum value of G(1,M) must be greater than Tmax.

The accuracy of the solution is not a function of the output time increment "DT" since the command produces the exact solution for straight line segments.

MAX X(t) Xmax^{+}

The MAX command locates the maximum absolute value in each row of the array named X(t) and stores the results in a column matrix Xmax. The maximum value and its column number is also printed or displayed.

PLOT M1 N=? R=R1,R2 - - RN S=S1,S2 - - RN

The PLOT command will produce a printer plot of "N" rows of matrix M1. Where "Ri" is the row number to be plotted and "Si" is the symbol used.

FUNCT G F(t)$^{+}$ N=? DT=?

The FUNCTion command forms a 1 x N array named F(t). The terms are extracted at "DT" intervals from the time function defined in the array named G. The array G defines a time function in terms of straight line segments where G(1,J) defined the time t_j and G(2,J) is the value $g(t_j)$. The time function must be defined in the range T = 0 to Tmax, where Tmax = N x DT. Therefore, the maximum value of G(1,M) must be greater than Tmax.

STEP K⁻ M C UVA⁻ U⁺ P F(t) DT=? L=Li,Lmax P=delta,alpha,theta

This command evaluates the displacements u, at equal time steps, of a structural system where the dynamic equilibrium equations are specified in the following form:

$$Ma(t) + Cv(t) + Ku(t) = PF(t)$$

Where

$a(t)$, $v(t)$ and $u(t)$ are the time-dependent acceleration, velocity and displacement vectors respectively.

K, M and C are the names of the N x N stiffness matrix, K, mass matrix, M, and damping matrix, C respectively.

The loads are specified as the product of a N x 1 vector P named P and a 1 x J array $F(t)$ named F(t). The loads F(i) are given at equal time steps as specified by "DT".

UVA is the name of a N x 3 array of initial conditions in which

The first column is a vector of initial displacements $u(0)$
The second column is a vector of initial velocities $v(0)$
The third column is a vector of initial accelerations $a(0)$

After STEP is executed this array will contain the displacements, velocities and accelerations at the last time step.

U is the name of the displacements which are stored as an N x Lmax array. The step-by-step integration is conducted with a time increment "DT"; however, the displacements are stored at Li time steps (or at "Li x DT" time intervals). Therefore, the number of loads specified "J" must be greater than " Li x Lmax ".

The Newmark-Wilson step-by-step integration method is used where the parameters are specified by delta,alpha and theta. The following table lists possible values:

	delta	alpha	theta
Newmark's Average Acceleration	1/2	1/4	1.00
Newmark's Linear Acceleration	1/2	1/6	1.00
Theta Method - Low Damping	1/2	1/6	1.42
Theta Method - High Damping	1/2	1/6	2.00

If the P parameters are not specified the linear acceleration method is used.

JACOBI $K^- V^+ M^- E^+$

This command solves the following eigenvalue problem for the mode shapes and frequencies:

$$\mathbf{KV = MVe}$$

Where "K" is the name of the N x N stiffness matrix $\mathbf{K}$ and "M" is the name of the N x N mass matrix.

The N x N matrix $\mathbf{V}$, which contains all eigenvectors (mode shapes) stored columnwise, is named "V" and is normalized in order that

$$\mathbf{V^T M V = I}$$

The N x N matrix $\mathbf{e}$ is a diagonal matrix of eigenvalues w_i^2 (frequencies w_i are in radians per sec.^{2}). The JACOBI command stores the eigenvalues e_i as a N x 1 column matrix named "E".

The program uses a modified Jacobi method where both $\mathbf{K}$ and $\mathbf{M}$ must be symmetric and positive definite matrices.

RITZ $K^- M F^- V^+$ NV=? S=?

Given a N x N stiffness matrix named "K", a N x 1 mass matrix named "M" and a N x 1 force vector named "F", a N x NV matrix of orthogonal vectors, $\mathbf{V}$, named "V" is generated using the WYD algorithm. The matrix $\mathbf{V}$ is normalized in order that

$$\mathbf{V^T M V = I}$$

The generated vectors $\mathbf{V}$ are not orthogonal with respect to the stiffness matrix $\mathbf{K}$. IF "S" is a nonzero number the static vector response is not included in the response.

NORM M1 $M2^+$ T=?

A row matrix M2 is formed in which each column contains the sum of the corresponding column of the matrix M1. If "T" is not equal to zero the square root of the sum of the square is calculated.

PROD M1 D

This command forms a 1 x 2 array named "D" which contains the product of all terms in the array named "M1". The produce is stored as two numbers of the form $D(1)\ 10^{D(2)}$.

LOG

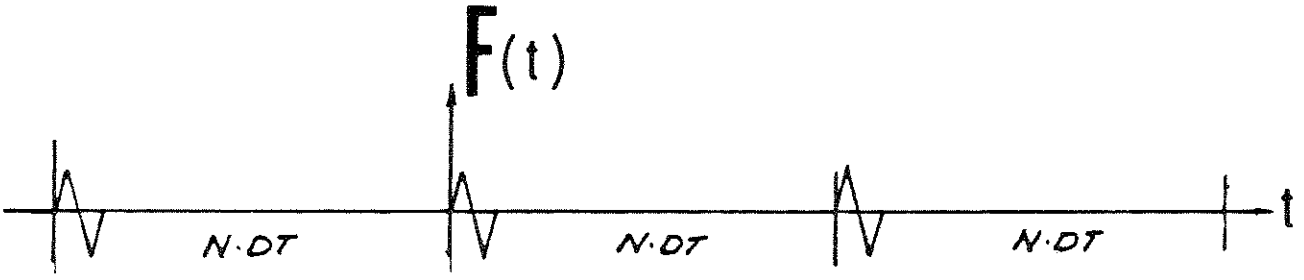
The LOG command replaces each term in matrix M_1 with the natural log of the term.

DYNAMIC ANALYSIS IN THE FREQUENCY DOMAIN

The CAL-80 operations DFT, IDFT and FSOLVE are provided in order to solve linear dynamic analysis problems in the frequency domain. This approach can be very effective for problems in which the loading is periodic over a very large time span such as machine vibrations or wind and wave loading on structures. It can be used for other types of loading, (i.e. earthquake ground motions), if the period of the loading is selected to be sufficiently long to assure that the response of the structure at the end of each loading period is essentially zero. If the damping of the system is small a very large period may be required if accurate results are to be obtained for loading which is not basically periodic. Following is a summary of these basic CAL operations:

DFT F(m,t)- DT=?

The M x N array named F(m,t) contains M different time functions. Each row in the array contains values of the function at equal time intervals DT. The time functions F(m,t) represent a time span of - ∞ to + ∞ ; however, only the values within a typical period (N DT) are specified as shown below:



TYPICAL TIME FUNCTION "m"

The term F(m,1) represents the value of the function "m" at the beginning and end of the basic time period N DT. The DFT operation expands the time functions in a series of the following form:

$$f(t) = f_0 + \sum f_{ck} \, COS(k\,dw) + \sum f_{sk} \, SIN(k\,dw)$$

where k=1,2,----- (N-1)/2 (for N odd), or, (N-2)/2 (for N even), and $dw = 2\pi / (N\,DT)$. The calculated constants for time function "m" are stored in the m^{th} row in the order f_0, f_{c1}, f_{s1}, f_{c2} - - and replace the original terms in the F(m,t) array. For N even the N^{th} column will be zero.

IDFT F(m,w)-

This operation transforms the frequency domain functions back to the time domain. The M x N array named F(m,w), which is in the form generated by the DFT or FSOLVE operations, is replaced by time function values at equal time intervals.

RADIUS F(m,w) R(m,w)

This command operates on the M x N F(m,w) array and creates a M x L R(m,w) array, where L = (N - 1) / 2. The terms are calculated from the following equations:

$$R(m,i) = \sqrt{F(m,2i)^2 + F(m,2i+1)^2}$$

FSOLVE W C F P(w) Y(m,w)+ DT=?

This operation evaluates the solution of a set of uncoupled second order differential equations which are generated in the mode superposition analysis of a structural system for which the loading has been transformed to the frequency domain. W, C, and F are M x 1 arrays and have the same definition as given by the DYNAM operation. The 1 x N array named P(w) is in the same form as produced by the DFT operation and DT is the time step which was used to transform the time domain to the frequency domain.

The m^{th} row in the M x N array named Y(m,w) contains the terms y_0, y_{c1}, y_{s1}, y_{c2} ----- which is the solution of the m^{th} mode written in the following form:

$$y(w) = y_0 + \sum y_{ck} \, COS(k\,dw) + \sum y_{sk} \, SIN(k\,dw)$$

The frequency domain solution Y(m,w) can be transformed to the modal time domain by the IDFT operation - IDFT Y(m,t) .

A COMPUTER ADAPTIVE LANGUAGE FOR THE DEVELOPMENT OF STRUCTURAL ANALYSIS PROGRAMS

SUMMARY

A group of FORTRAN 77 subroutines is presented which are designed to augment the standard FORTRAN language and to produce a new higher-order, machine-independent language for the development of structural engineering software . The group of subroutines which comprise the Computer Adaptive Language for the development of Structural Analysis Programs, CAL/SAP, is designed to effectively operate on micro, supermini and mainframe computers. The CAL/SAP system has been used as the basis for the development of the SAP-80 series of programs and for CAL-80, a series of interactive programs for Computer Assisted Learning of structural analysis and design.

The subroutines which define the CAL/SAP development system are divided into the following three categories:

First, a series of free-field input routines allows input data to be specified in a consistent manner, in arbitrary order, with optional name identification and in arithmetic statement form.

Second, a set of incore data management subroutines allows dynamic storage allocation to be accomplished with integer, real and ASCII data with a minimum of programming effort. These subroutines eliminate paging problems on modern super minicomputers with virtual operating systems.

Third, an out-of-core data management system allows different programs to access the same data. Simple operations allow data transfer between the in and out-of-core systems. The out-of-core data base provides for sequential, direct access and bulk data files. Communication between different data bases allow techniques such as multilevel substructure analysis to be implemented with a minimum of programming.

The use of the CAL/SAP development system allows computer programs to be rapidly developed and maintained. Also, it can be used to upgrade existing software in order to obtain modularity and to operate efficiently on the new generation of computer systems. The purpose of this paper is to present the CAL/SAP development system and to illustrate that computer independent programs in structural engineering and computational mechanics can be developed which operate on both large and small computers.

INTRODUCTION

The use of a series of utility subroutines to facilitate the development of computer programs in the general area of structural analysis and design is not new. Programs such as SMIS [1], STRUDL [2], ASKA [3], NASTRAN [4], GIFTS [5], POLO-FINITE [6], IPAD [7], SESAM [8], NORSAM [9], CAL [10], FEMALE [11], FACTS [12] and NICE [13] are examples of programs which are based on symbolic input or use special techniques to manage core and secondary storage. Also, general purpose database management systems such as RIM [14] have a potential for use with structural engineering programs. Most of these programming systems were developed for use on mainframe computers and cannot be easily modified to operate on the new generation of microcomputers which have a 64k byte limitation. Also, the FORTRAN source statements for many of these programs are not available to be used by other developers.

Most special purpose computer programs in structural engineering, which have been developed within the past twenty years, have been designed to operate on large computers in a batch data input mode. Also, the internal structure of many of these older programs is monolithic in which different areas of the program communicate with COMMON data areas and external sequential temporary TAPE files. Larger programs required complex overlay structure because of core storage limitation. Very few programs of the past generation have simple restart options since the data structure is fragmented between core storage and several external sequential storage devices. Therefore, these programs are difficult to maintain and modify. In addition, many of these older programs are based on obsolete numerical methods and many of the structural elements should be replaced with more modern and accurate elements.

Since the introduction of inexpensive and very powerful mini and microcomputers, there has been an attempt to convert these old programs to the new computer systems. This approach not only tends to perpetuate obsolete technology, but it does not take advantage of modern computer hardware and software. Of course, this approach is justified because of the large cost required to develop and verify new computer programs. Also, most users have little motivation to learn how to use a new program unless it has new capabilities. A reduction in the cost of running a program or an increase in the accuracy of the results are usually not strong enough reasons for the normal user to change to a new, unfamiliar program.

The major purpose of this paper is to present methods of programming which will reduce the cost of development and maintenance of new and old programs in structural engineering. The standardization of the user-friendly free-field input data files allows the user to learn how to use a new program with a minimum of effort. The incore and out-of-core storage data management routines allow new and modern elements and numerical methods to be rapidly incorporated. Therefore, many of the obstacles in the development of new programs which were previously described can be eliminated.

The techniques presented in this paper were used as the basis for the development of the SAP-80 series of programs [15]. Also, they were used to modernize the CAL program, which was initially designed to operate in the batch mode. CAL is now a series of programs with both batch and interactive options which operate on both micro and mainframe computers [16]. The CAL-80 program not only has its previous options, but can be used as the basis for many new program modules in computer graphics, computer aided design and data reduction associated with experimental projects.

The CAL/SAP development system is based on the assumption that modern computer programs are subdivided into several "program modules" which are executed separately. The components and operation of a typical module is illustrated in figure 1. Each set of subroutines is independent; therefore, all three sets of subroutines may not be required in each program module. For a small special purpose program, a developer may find that only the data input interpretation subroutines are required.

The CAL/SAP system and the CAL program are protected by copyright law; therefore, they cannot be modified and resold as a part of a proprietary program. However, organizations which join the CAL/SAP users group may develop programs which interface with the CAL/SAP system and will have the opportunity to interchange program modules.

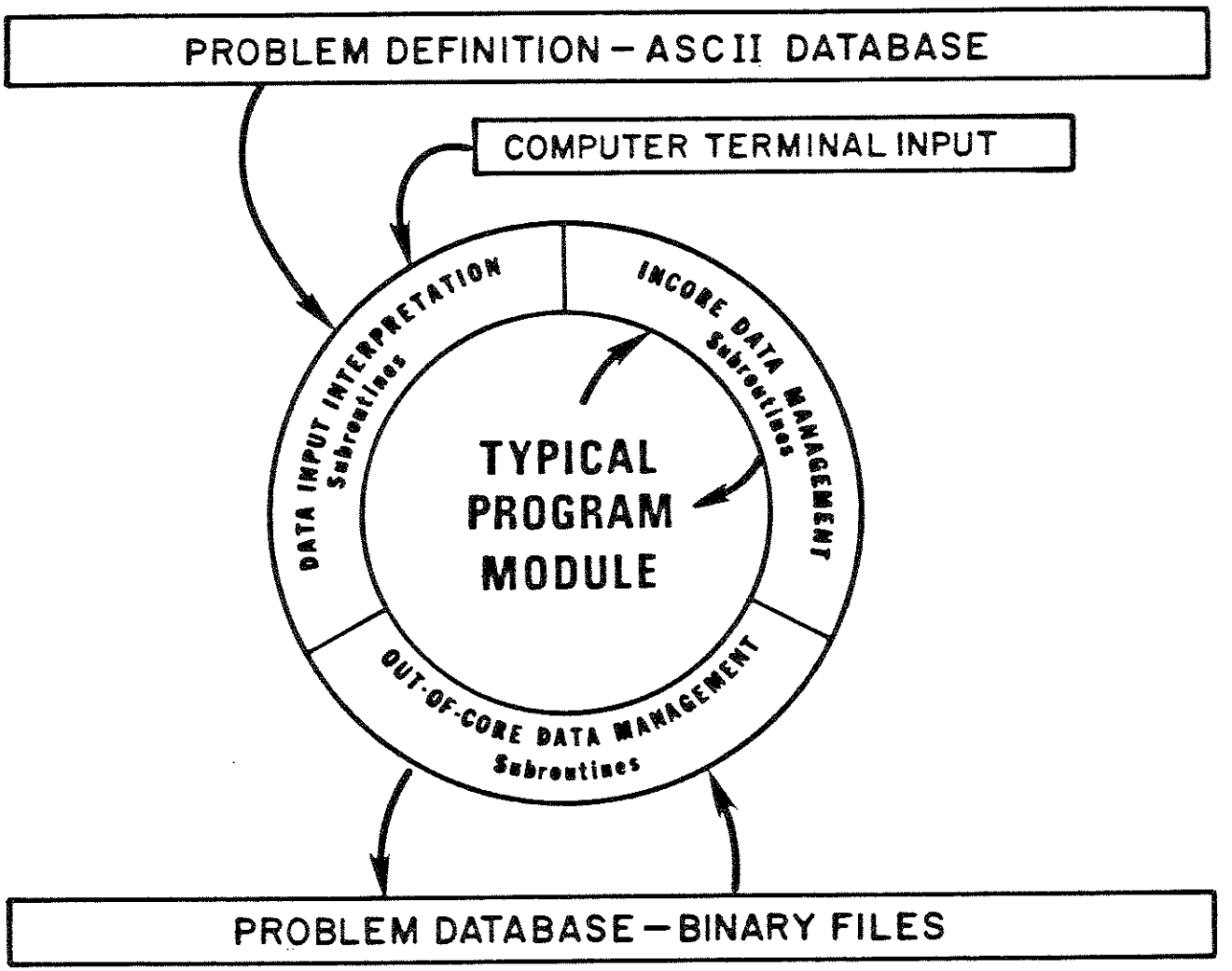


FIGURE I. THE CAL/SAP DEVELOPMENT SYSTEM

INTERACTIVE AND FREE-FIELD INPUT

One of the obvious advantages in the use of modern micro and minicomputers is that the input data can be specified in an interactive manner. In an interactive mode of data input the computer program writes a request to the terminal screen and the user responds by typing the appropriate information at the keyboard. This approach can eliminate the need for an elaborate users manual. After careful consideration, however, it was decided that additional input data options were required for the following reasons:

The programming effort required to prepare and edit the input data interactively is comparable or greater than the programming of a new structural element subroutine.

Many very competent engineers do not type rapidly and prefer to prepare data on forms and then have assistants enter the information into the computer. Also, after a user becomes familiar with a program the screen prompts may tend to slow down the experienced user.

A complete copy of the input data is necessary if problems are to be rerun at a later time with a small fraction of the input data altered.

It is desirable that the permanent copy of the input information be in a compact and symbolic form for rapid terminal display.

A large number of very excellent open screen editors are now available for preparation of text or data files on all modern computer systems. The user of the computer generally is familiar with such an editor and its use does not require the confusion of learning several different interactive data entering techniques used by different programs.

The selection of a symbolic free-field input data structure offers many of the advantages of interactive input with respect to ease of use and error minimizations.

The same free-field input subroutines can be used in all new program segments; therefore, a minimum of new programming effort is required to expand the capability of a program.

Preformatted data files can be prepared for certain classes of problems in which the location and definition of the data to be entered is given by comment information which is ignored when the file is read.

Interactive input can still be used when it is the most appropriate method of entering a small amount of data.

TYPICAL LINE OF DATA

A typical line of input information which is read from the input data file (or read from the computer console) is entered in the following free-field form:

N1,N2,N3,--- A=A1,A2,A3--- B=B1,B2,B3------

Where the input data is designated by Ni, Ai or Bi. The data fields must be separated by a single comma or by one or more blanks; therefore, a fixed field format can be used for some data if required. The data N1,N2,N3-- without identification must be the first information on the line. Input data of the form B=B1,B2,B3,--- can be in any order or location on the line.

For example, the following line of data can be entered:

2,4,5 O=DEBUG E=29600*144 C=200*12+3.5, 400/12 AREA=10+20/5-2

Simple arithmetric statements are possible when entering floating point real numbers. The statement 10+20/5-2 is evaluated as ((10+20)/5)-2).

The "mini name-lists" of data A= or B= can be in any order and provide a flexible method of supplying a large amount of information on a single line. The symbolic identifiers make the data line very readable on a video terminal and allow information to be rapidly modified by a standard open-screen text editor.

FREE-FIELD INPUT SUBROUTINES

Five FORTRAN subroutines are presented which are designed to standardize the transfer of data from the input data file to information to be used within the computer program. The file is divided into data groups with each group identified by a separator or header. This separator is used as a title for the data that follows. For example, the separator LOADS can be used as the header for the concentrated joint loads. Separators can be variable length alphanumeric strings. The separator must begin in the first column of an input line. Only the first four characters of the string are compared while searching the input file for the separator. The rest of the line can be used for comments.

Within a FORTRAN program the calls to these subroutines are of the following form:

```
CALL FIND('SEPA',KEY)
CALL FREE
CALL FREEPT
CALL FREEI('I',ISYM,NUM)
CALL FREER('R',RSYM,NUM)
CALL FREEH('H',HSYM,NC,NUM)
```

The FIND subroutine searches the input data file for the specified four character separator SEPA in order that the data which follows the separator can be read in sequence. If the separator is not found, the routine will return with KEY set equal to 1 . The data following a separator line is specified in a "mini-namelist" fashion as previously indicated and can contain 0 to 160 characters in each record.

The FREE subroutine reads one record from the input file and transfers that information to an internal line buffer. Each call to FREE advances the input file by one record.

The FREEPT subroutine will echo the last record read on the console and output file.

The FREEI subroutine will search the line buffer for the symbol "I=" and extract the next "NUM" integers and store them in the internal array named "ISYM".

The FREER subroutine will search the line buffer for the symbol "R=" and extract the next "NUM" real numbers and store them in the internal array named "RSYM". Arithmetic statements are allowed only for real data.

The FREEH subroutine will search the line buffer for the symbol "H=" and extract the next "NUM" Hollerith words and store them in the internal array named "HSYM", where "NC" indicates the number of characters in each word. If less than NC characters are specified the remaining characters are set to blanks. Within the program the Hollerith data must be specified as character data and dimensioned correctly (i.e. HSYM(NC,NUM).

These routines are very flexible and offer many options that can make an input file very readable. The following is a summary of the options and conventions which are provided if this package of subroutines is used:

A "C" in column 1 of any line will cause the line to be echoed as a comment on the console.

A backslash "\" at the end of information on the a line will allow the next line to be interpreted as a continuation of the previous line; therefore, a 160 character record is possible.

A colon ":" indicates the end of information on a line. Information entered to the right of the colon is ignored by the program. Therefore, it can be used to provide additional comments within the input file.

If a blank identifier is specified, the data string is assumed to be the first data string of the record.

If less data exists than is specified by NUM, the values returned will be either a zero or blank according to the routine used.

When an identifier is not found, the values will not be changed and will be the same as before the call.

Real numbers do not require decimal points; E formats with + or - exponents are accepted.

Simple arithmetric statements can be used within the input. The functions that can be used are +, -, *, /. The order of evaluation is sequential, not hierarchical as in the FORTRAN language.

More than one character can be used as an identifier. For example, TOL = 0.001 would be extracted by CALL FREER('L',TOL,1). Note that when using multiple character IDs, the last character must be unique on a given line.

The FORTRAN 77 source statements for these subroutines are given. These should run on most systems without modification. Only a few statements were required to be changed for them to be operational with Microsoft FORTRAN which is the standard for CP/M microcomputer systems.

EXAMPLE OF FREE FIELD INPUT

One subroutine will be given in order to illustrate the use of some of the possible options of this group of input subroutines. The following input data entered after the JOINTS separator is intended to describe the joint geometry of a simple one-dimensional structural system.

```
C EXAMPLE DATA FILE
:
JOINTS                 :JOINT COORDINATES
1    X=100             :JOINT AT LEFT END OF BEAM
11   X=250 G=1,10,1    :JOINT AT RIGHT END and GENERATION
                       :END OF JOINT INPUT DATA
```

The following subroutine will read this data and generate the internal joint coordinates at equal intervals:

```
      SUBROUTINE JOINTS(X,NUMNP,NOUT)
      DIMENSION  X(NUMNP)
      COMMON /TEMP/ NMIN,NMAX,NINC
C-----SEARCH INPUT FILE FOR JOINT DATA---
      CALL FIND('JOIN',KEY)
      IF(KEY.EQ.0) GO TO 100
      WRITE(NOUT,2000)
      STOP
C-----READ OF JOINT COORDINATES----------
 100  N = 0
      CALL FREE
      CALL FREEPT
      CALL FREEI(' ',N,1)
      IF(N.EQ.0) RETURN
      CALL FREER('X',X(N),1)
C-----CHECK FOR GENERATION---------------
      NMIN = 0
      CALL FREEI('G',NMIN,3)
      IF(NMIN.EQ.0) GO TO 100
C-----GENERATE JOINT GEOMETRY------------
      IF(NINC.EQ.0) NINC = 1
      XN = ( NMAX - NMIN ) / NINC
      DX = ( X(NMAX) - X(NMIN) ) / XN
      ML = NMIN + NINC
      MH = NMAX - NINC
C
      DO 200 M=ML,MH,NINC
      MM = M - NINC
 200  X(M) = X(MM) + DX
      GO TO 100
C----------------------------------------
 2000 FORMAT (' * * JOINT DATA NOT FOUND * *')
      END
```

It is apparent that the use of this group of input subroutines offers versatility in creating readable input files. Also, the internal use of the subroutines is relatively simple since all input FORMAT statements have been eliminated. The practical application of this form of input is only limited by the creativity of the engineer/programmer.

INCORE DATA MANAGEMENT

The standard approach in the development of FORTRAN programs is to reserve storage for arrays by the use of DIMENSION statements. Except for small, special purpose programs, the size required by many of the dimension statements is not known at the time the program is written. The concept of dynamic storage allocation has been used in order to avoid the problem of recompiling the program for different size problems.

In dynamic storage allocation, core storage for all arrays is reserved in the form of a single one-dimensional array in blank common. Then, at the time of execution of the program on different computers only the size of the one-dimensional array need be changed. Also, all data storage is compacted into a local area of computer storage. This has advantages for operation on modern computers with virtual operating systems. These computers are very inefficient if the data is fragmented in a large area of storage since this will require extensive paging.

The series of subroutines which are presented in this section are designed to allow storage to be easily allocated and managed during the execution of the program. It also allows data to be accessed from any subroutine without passing the array names through as arguments to subroutines.

Each subroutine which communicates with the incore data requires a statement of the following form:

```
COMMON MTOT,NP,IA(1)
```

Where MTOT will be the actual size of the IA array which must be set in the main program. All arrays which are contained in the incore data base system are designated by a four character ASCII name which is selected by the programmer. Before data is entered in the data area the constants in the COMMON/DBSYS/ must be initialized.

The five subroutines which are used to allocate and manage storage are called by the following statements:

```
CALL DEFINE('NAME',NA,NR,NC)
CALL DEFINI('NAME',NA,NR,NC)
CALL DEFINH('NAME',NA,NR,NC)
CALL LOCATE('NAME',NB,NR,NC)
CALL DELETE('NAME')
```

Where 'NAME' is a four character name to be assigned by the user. The subroutines DEFINE, DEFINI and DEFINH reserve storage for REAL, INTEGER and Hollerith data respectively. The arrays are specified to have NR rows and NC columns. The value of NA is returned which indicates the location of the array in the IA array.

The subroutine LOCATE returns the address NB and the number of rows NR and the number of columns NC of an array 'NAME' which has been previously defined. The value of NB will be zero if the array has not been previously defined.

The subroutine DELETE removes the array 'NAME' from the incore storage area and then releases the storage for use by other data. It should be noted that new arrays are always added at the end of the IA storage array. Therefore, if arrays are deleted the other arrays will be relocated in storage unless the array deleted is at the end of the IA array. It should be noted that if the arrays are deleted in reverse order of definition the incore data will not be moved. Also, the use of small program segments should reduce the requirement of incore deletion of data.

The incore data base can be saved on an externally named file. If another program requires the same incore data base, the appropriately named file can be opened and the incore data restored. With this method of transferring data between different programs, it is possible for large programs to be subdivided into small program segments and run on microcomputers with small storage requirements for each program segment. This technique is used extensively in the CAL-80 series of programs.

OUT-OF-CORE DATA MANAGEMENT

Many programs, in the general area of structural analysis, use out-of-core data base management systems. Most of these programs were developed prior to the release of FORTRAN 77 and prior to the existence of modern computer operating systems such as CP/M and UNIX. Therefore, these programs use one large disk storage area in which the data base system allocates storage for an array and then records its location as a direct access record number.

The standard FORTRAN 77 language allows direct access to named files which reside in the computer operating system and has introduced several new commands which are associated with file manipulation. As a result of these new developments the advantages of using only one general purpose data base management system has been diminished.

The group of subroutines which are responsible for the out-of-core data management in the CAL/SAP development system contains the option of direct access files as well as some new options for direct communication with files which reside in the computer's operating system. Some of the advantages of the data base management system, DBM, presented in this paper are as follows:

First, very little additional storage is required for programs which use the DBM system. Therefore, it is possible to use these techniques on microcomputer systems with a limitation of 64k bytes of core storage. This allows programs which are developed on very small computer systems to be transferred without modification to larger computers.

Second, the direct use of files within the operating system is more efficient with respect to the use of disk storage. This is because the deletion of information from the data base and the use of the deleted file storage areas is automatically accomplished by the computer's operating system. This task is one of the most complicated phases in the use of any general purpose data base management system. Therefore, the subroutines presented in this paper are very simple and compact when compared with other out-of-core data base management systems.

Third, most of the out-of-core storage requirements associated with element data within a structural analysis program are best served by standard FORTRAN sequential files, rather than by separate calls to a DBM system for each element. In the CAL/SAP DBM system both types of files are possible.

The most important aspect with respect to the use of any data base is that the FORTRAN CALL statements be defined and used consistently. If this is accomplished computer programs which are written for one data base system can easily be converted to operate on another system. The following is a list of subroutine CALLs to the CAL/SAP DBM system:

```
CALL IFILE
CALL POPEN('ext ')
CALL FOPEN(LUN,'ext ')
CALL FCLOSE(LUN)
CALL FILE (RSYM,'ext ',NR,NC)
CALL FILEI(ISYM,'ext ',NR,NC)
CALL RFILE (RSYM,'ext ',NR,NC)
CALL RFILEI(ISYM,'ext ',NR,NC)
```

The subroutine IFILE reads the name of the input data file. This name will be the first name for all data files associated with this structure (or substructure). This name will be contained in the character array named FIN which is made available to all data base subroutines by the named COMMON /PARC/ statement. All files which are associated with the problem are referred to by a second name 'ext ' which is used as an argument in the subroutine calls.

The subroutine POPEN opens the formatted print file, 'ext ', and connects the file to the logical unit number NOT. Therefore, each phase of the program can have a different output file.

The FOPEN subroutine opens an existing unformatted sequential access file, 'name', and connects the file to the logical unit specified by LUN. This file may then be written or read with standard FORTRAN WRITE or READ statements. If LUN is negative the file will be formatted. The NOPEN subroutine opens a new unformatted sequential access file.

The FCLOSE subroutine closes the previously named file and disconnects it from logical unit LUN. The same logical unit number can then be attached to another named file within the same program by another FOPEN call. This illustrates that a logical unit number can be thought of as a temporary input/output buffer which is defined within a FORTRAN 77 program. This is a significant change from traditional FORTRAN in which a logical unit number was defined as an external tape-like device.

The subroutines FILE and RFILE write and read real data to and from the file, 'ext ', where NR and NC indicate the number of rows and columns in the array identified by RSYM. NR and NC are returned when the file is read. The subroutines FILEI and RFILEI are used to transfer integer data. Logical unit number NFL is reserved for use by these subroutines which transfer array data in and out of core storage.

The FORTRAN source statements for these out-of-core data base operations are given. One notes that they involve a minimum number of FORTRAN statements; therefore, they can be used effectively on microcomputers with a limited amount of storage.

USE OF THE CAL/SAP SYSTEM

It is apparent that the input/output logical unit numbers must be set and other system dependent variables must be set prior to the use of the subroutines presented in this paper. The IOSET subroutine given performs that function for MS-DOS types of computer systems. The subroutines STOPC save the incore data base at the termination of a program module. The subroutine READC reads the incore data base information. This allows different program modules to transfer information very effectively. Since CAL-86 can read information from other program modules the information can be examined interactively and is of great value in a debugging mode of operation.

FINAL REMARKS

The group of subroutines presented in this paper is intended to augment the FORTRAN language for the development of program modules in the general area of structural engineering. It assumes that structural analysis and design is accomplished by a series of programs which operate on modern interactive computer systems. It is intended to make maximum use of the features of FORTRAN 77 and Microsoft FORTRAN for microcomputers. Also, it is designed to utilize the data management techniques which are inherently contained in modern computer operating systems. The use of these subroutines should allow computer programs to be rapidly developed or modified in a very short period of time and at a minimum cost. In addition, the resulting software should be portable and have a long functional life which is independent of the rapid changes in modern computer hardware, operating systems and FORTRAN compilers.

REFERENCES

1 E. L . Wilson, "SMIS, Symbolic Matrix Interpretive System", University of California, Berkeley, Department of Civil Engineering, Report No. UCSESM73-3.

2 R. D. Logcher, G. M. Sturman, "STRUDL - A Computer System for Structural Design", Journal of the Structural Division, ASCE No. ST6, Paper 5010, December, 1966.

3 E. Schrem, J. R. Roy, "An Automatic System For Kinematic Analysis ASKA", Part 1 IUTAM Colloq. High Speed Computing of Elastic Structures, University of Liege, 1970.

4 C. McCormick (Ed.), NASTRAN Users Manual, NASA, 1969.

5 H. Kamel, M. W. McCabe, "GIFTS System Manual", Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson, Arizona, 1978.

6 L. A. Lopez, "FINITE: An Approach to Structural Mechanics Systems", International Journal for Numerical Methods in Engineering, Vol. 11, 1977.

7 R. E. Fulton, "IAPD Project Overview", NASA Conference Publication 2143, September 17-19, 1980.

8 Egeland, Araldsen, "SESAM-69: A General Purpose Finite Element Method Program", Journal of Computers and Structures, 4, 1974.

9 Bell, Hatlestad, Hansteen, Araldsen, "NORSAM Users Manual", Trondheim, Norway, 1973.

10 E. L. Wilson, "CAL - A Computer Analysis Language for Teaching Structural Analysis", Journal of Computers and Structures, Volume 10, 1979.

11 G.J.V. Shopee, P. J. Jeanes, T. B. Griffin, "A Finite Element Modelling and Analysis Language for Engineers - The Program for Computational Mechanics", Journal of Computers and Structures, Volume 13, 1981.

12 J. P. Hollings, "The Substructuring Technique as Applied to Linear Elastic Analysis", Ph.D. Dissertation, Department of Civil Engineering, University of California, Berkeley, 1978

13 C. Felippa, " Architecture of a Distributed Analysis Network for Computational Mechanics", Computers and Structures, Vol. 13, 1981

14 C. L. Blackburn, O. O. Storaasli, R. E. Fulton, "The Role and Application of Data Base Management in Integrated Computer-Aided Design", IPAD Project Report, NASA.

15 M. I. Hoit, "New Computer Techniques for Structural Engineering", Ph.D. Dissertation, Department of Civil Engineering, Berkeley, 1883.

SECTION 5

FORTRAN LISTING

(Contained on the diskette supplied with this report)

CAL 86 – COMPUTER ASSISTED LEARNING

This listing may not be the most current listing of CAL. For the lastest version of the program please contact

NISEE/COMPUTER APPLICATIONS

DAVIS HALL

UNIVERSITY OF CALIFORNIA

BERKELEY, CALIFORNIA 94720

(415) 642-5113

Numerical Methods for Dynamic Analysis

E. L. Wilson

6.1 INTRODUCTION

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 re-analysis 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 the minimization of computer execution time. The purpose of this paper is to present a summary of various numerical methods which are used in the dynamic analysis of offshore 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 behaviour of the system. Therefore a considerable insight into the expected dynamic behaviour 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 behaviour.

The force equilibrium of an offshore platform modelled as a system with a finite number of degrees of freedom may be written as

$$F_i + F_d + F_s = F \tag{6.1}$$

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 (6.1) holds for both linear and non-linear systems. However the appropriate numerical method for solution will depend on the degree of non-linearity which is present and if linearization is possible.

6.2 LINEAR DYNAMIC ANALYSIS

For linear systems with viscous damping Equation (6.1) can be written in the form

$$M\ddot{U}(t)+C\dot{U}(t)+KU(t)=F(t) \tag{6.2}$$

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 vectors $U(t)$, $\dot{U}(t)$ and $\ddot{U}(t)$ are the displacements, velocities and accelerations respectively.

The time dependent external forces $F(t)$ may be due to wave or seismic forces. Normally the calculation of wave forces is a complicated procedure which depends on the shape of the structural elements. Other papers in these proceedings are referred to for the evaluation of wave forces. In the case of earthquake motion in three-dimensions the loading is of the form

$$F(t)=-M_x\ddot{U}_{xg}-M_y\ddot{U}_{yg}-M_z\ddot{U}_{zg} \tag{6.3}$$

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 behaviour, 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 6.1 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

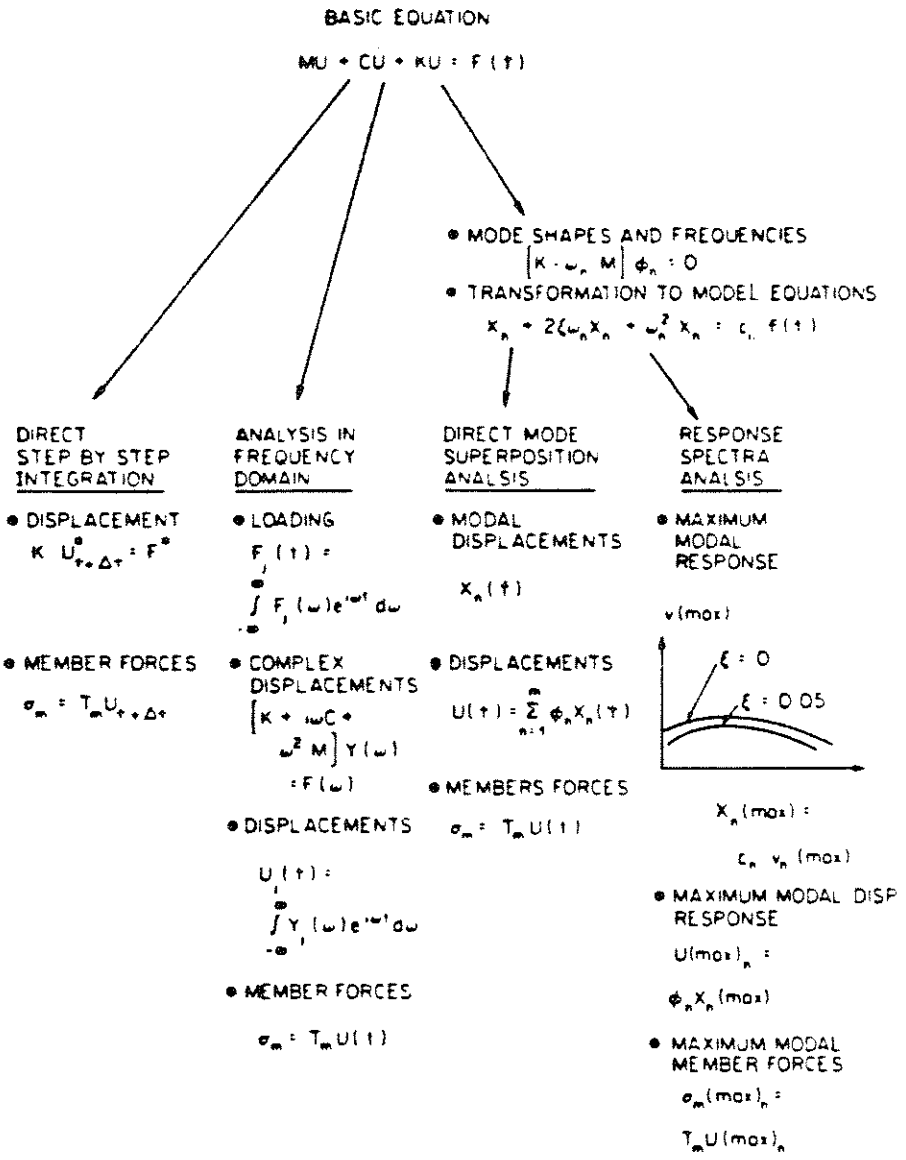


Figure 6.1 Methods for linear dynamic analysis

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 6.1 involves 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.

6.2.1 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

$$AX = b \tag{6.4}$$

where A is an $N \times N$ symmetrical matrix and 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 nonzero 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 6.2(a) can be stored as a one-dimensional array as shown in Figure 6.2(b) along with an integer pointer array indicating the location of each diagonal term. Figure 6.2(c) 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*) *Triangularization of* A ($j = 2$ *to* N)

$$A = LU \quad \text{or} \quad A = LDL^T \tag{6.5}$$

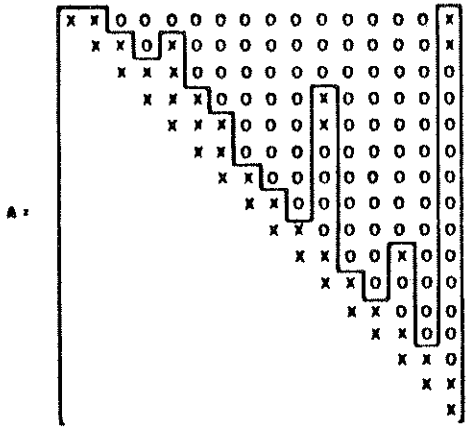
in which the jth column of upper triangular matrix U is evaluated from

$$U_{ij} = A_{ij} - \sum_{k=km}^{j-1} L_{ik} U_{kj} \qquad i = fj \text{ to } j \tag{6.6}$$

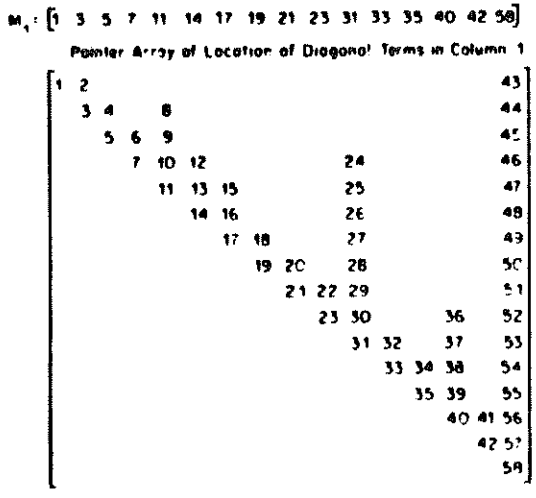
and the jth row of lower triangular matrix is given by

$$L_{ji} = U_{ij} / D_{ii} \qquad i = fj \text{ to } j-1 \tag{6.7}$$

where D is a diagonal matrix, fj is the first non-zero term in column j and fi is

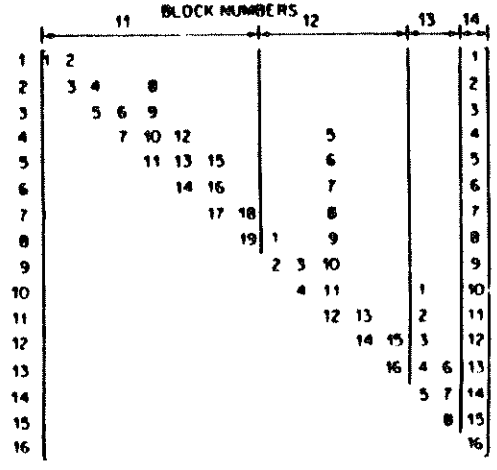


(a) Elements in Original Matrix



(b) Storage Sequence for in core 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	9						
4	1	16	16	16							

(c) Example of Matrix Stored in a Maximum of 20 Blocks

Figure 6.2

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

$$z_i = b_i - \sum_{k=fi}^{i-1} L^*_{ki} z_k \qquad i = 1 \text{ to } N \tag{6.8}$$

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

$$y_i = z_i / D_{ii} \qquad i = 1 \text{ to } N \tag{6.9}$$

(*c*) *Backsubstitution*
From $y = L^T x$

$$x_i = y_i - \sum_{k=i+1}^{N} L^*_{ik} y_k \qquad i = N \text{ to } 1 \tag{6.10}$$

It is important to note that all zero operations are skipped by this technique and that neither the number of operations or the the required storage is a function of the band width. Also the triangularization operation on the A matrix is independent of the forward or backsubstitution operations; therefore this operation need be done only once. The forward and backsubstitution 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.

6.2.2 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 behaviour 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.[2-12] 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

$$M\ddot{U}_t + C\dot{U}_t + KU_t = F_t \tag{6.11}$$

The following standard finite difference relationships are used:

$$\ddot{U}_t = \frac{1}{\Delta t^2}(U_{t-\Delta t} - 2U_t + U_{t+\Delta t}) \tag{6.12}$$

$$\dot{U}_t = \frac{1}{2\,\Delta t}(U_{t+\Delta t} - U_{t-\Delta t}) \tag{6.13}$$

Equations (6.12) and (6.13) can be substituted into Equation (6.11) to form a set of linear equations of the form

$$M^* U_{t+\Delta t} = F^* \tag{6.14}$$

where

$$M^* = \frac{1}{\Delta t^2}M + \frac{1}{2\,\Delta t}C \tag{6.15}$$

$$F^* = F_t - KU_t + \frac{1}{\Delta t^2}M(2U_t - U_{t-\Delta t}) + \frac{1}{2\,\Delta t}CU_{t-\Delta t} \tag{6.16}$$

If M and C are diagonal one notes that the solution of Equation (6.14) 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:

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta t\,(1-\delta)\ddot{U}_t + \Delta t\,\delta\ddot{U}_{t+\Delta t} \tag{6.17}$$

$$U_{t+\Delta t} = U_t + \Delta t\,\dot{U}_t + \Delta t^2\,(\tfrac{1}{2}-\alpha)\ddot{U}_t + \Delta t^2\,\alpha\ddot{U}_{t+\Delta t} \tag{6.18}$$

where α and δ are selected to produce the desired accuracy and stability. If $\delta = \frac{1}{2}$ and $\alpha = \frac{1}{6}$ 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 ($\delta = \frac{1}{2}$ and $\alpha = \frac{1}{4}$) 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

$$M\ddot{U}_{t+\Delta t} + C\dot{U}_{t+\Delta t} + KU_{t+\Delta t} = F_{t+\Delta t} \tag{6.19}$$

This equation can be solved by iteration; however Equations (6.17), (6.18) and (6.19) 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

$$K^* U_{t+\Delta t} = F^* \tag{6.20}$$

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 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 oscillations which often develop in linear and non-linear 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 6.1. With $\theta = 1$ the approach is the standard Newmark method. An unconditionally stable method with large damping in the higher modes is produced with $\delta = \frac{1}{2}$, $\alpha = \frac{1}{6}$ and $\theta = 1{\cdot}4$.[12]

6.2.3 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

Table 6.1 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

$$\delta \geq 0{\cdot}50; \qquad \alpha \geq 0{\cdot}25(0{\cdot}5+\delta)^2; \qquad \theta \geq 1{\cdot}0$$

4. Calculate integration constants:

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

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

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

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

5. Form effective stiffness matrix: $K^* = K + a_0 M + a_1 C$
6. Triangularize K^*: $K^* = LDL^T$

B. FOR EACH TIME STEP:

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

$$F^* = F_{t+\tau} + M(a_0 U_t + a_2 \dot{U}_t + a_3 \ddot{U}_t) + C(a_1 U_t + a_4 \dot{U}_t + a_5 \ddot{U}_t)$$

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

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

3. Calculate accelerations, velocities and displacement at $t+\Delta t$:

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

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

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

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

harmonic functions. One can use the standard Fourier Series in which the loads $F(t)$ are expanded in a series of the form

$$F(t) = \sum_{n=0}^{\infty} A_n \cos\frac{n\pi}{d}t + \sum_{n=0}^{\infty} B_n \sin\frac{n\pi}{d}t \qquad (6.21)$$

in which d is the duration of the loading. The Fourier Coefficients can be evaluated and exact solutions found for the harmonic functions $A_n \cos(n\pi/d)t$

and $B_n \sin (n\pi/d)t$. 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 straightforward 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

$$F(t) = \int_0^\infty (A(\omega) \cos \omega t + B(\omega) \sin \omega t)\, d\omega \tag{6.22}$$

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

$$A(\omega) = \frac{2}{\pi} \int_0^d F(t) \cos \omega t \, dt \tag{6.23}$$

$$B(\omega) = \frac{2}{\pi} \int_0^d F(t) \sin \omega t \, dt \tag{6.24}$$

Also the Fourier integral can be written in complex form as

$$F(t) = \int_{-\infty}^{\infty} \bar{F}(\omega)\, e^{i\omega t}\, d\omega \tag{6.25}$$

in which

$$\bar{F}(\omega) = \tfrac{1}{2}[A(\omega) - iB(\omega)] \tag{6.26}$$

$$\bar{F}(-\omega) = \tfrac{1}{2}[A(\omega) + iB(\omega)] \tag{6.27}$$

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

$$M\ddot{U} + C\dot{U} + KU = \int_{-\infty}^{\infty} \bar{F}(\omega)\, e^{i\omega t}\, d\omega \tag{6.28}$$

The solution is assumed to be of the form

$$U(t) = \int_{-\infty}^{\infty} Y(\omega)\, e^{i\omega t}\, d\omega \tag{6.29}$$

therefore

$$\dot{U}(t) = \int_{-\infty}^{\infty} i\omega Y(\omega)\, e^{i\omega t}\, d\omega \tag{6.30}$$

$$\ddot{U}(t) = \int_{-\infty}^{\infty} -\omega^2 Y(\omega) e^{i\omega t} \tag{6.31}$$

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

$$(K + i\omega C - \omega^2 M) Y(\omega) = \bar{F}(\omega) \tag{6.32}$$

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 [Equation (6.67)]. 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 (6.29).

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.

6.2.4 Numerical evaluation of mode shapes and frequencies

For large structural systems one of the most time consuming phases of a dynamic analysis may be the evaluation of eigenvalues and eigenvectors of the $N \times N$ matrix equation

$$MU + \bar{\omega}^2 KU = 0 \tag{6.33}$$

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

$$(K - \bar{\omega}_n^2 M)\phi_n = 0 \tag{6.34}$$

(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 (6.34) is rewritten as

$$\begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix} \begin{bmatrix} \phi_a \\ \phi_b \end{bmatrix} = \bar{\omega}^2 \begin{bmatrix} 0 & 0 \\ 0 & M_b \end{bmatrix} \begin{bmatrix} \phi_a \\ \phi_b \end{bmatrix} \tag{6.35}$$

The first submatrix equation is

$$K_{aa}\phi_a + K_{ab}\phi_b = 0 \tag{6.36}$$

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

$$\phi_a = T\phi_b \tag{6.37}$$

where

$$T = -K_{aa}^{-1}K_{ab} \tag{6.38}$$

The resulting eigenvalue problem is of the form

$$K_{bb}^{*}\phi_b = \bar{\omega}^2 M_b \phi_b \tag{6.39}$$

in which

$$K_{bb}^{*} = K_{bb} + K_{ba}T \tag{6.40}$$

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 (6.34)] 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 behaviour of the structure is known. Static load patterns are selected and corresponding displacement vectors are calculated. Or

$$KR = P \tag{6.41}$$

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

$$\phi = RX = R_1X_1 + R_2X_2 + R_3X_3 + \cdots R_LX_L \tag{6.42}$$

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 (6.42) into Equation (6.34) and premultiplication by R^T. Or,

$$K^*X - MX\Omega = 0 \tag{6.43}$$

where

$$K^* = R^T K R = R^T P \tag{6.44}$$

$$M^* = R^T M R \tag{6.45}$$

$$\Omega = \text{diag}\,(\omega_i^2) \tag{6.46}$$

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 (6.41). Therefore a large number of structural elements can be used to model the basic structural behaviour. 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 6.3 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 lateral load pattern shown the resulting displacement is complex and involves movements in all directions in addition to torsional behaviour.

For a structure of this type one would expect the lateral behaviour 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 behaviour are easily established and are shown in Figure 6.3a. Of course the first six vibrational mode shapes for this structure will be composed of a linear combination of the resulting displacement patterns. Figure 6.3b illustrates six other load patterns which could have been used to produce identical results.

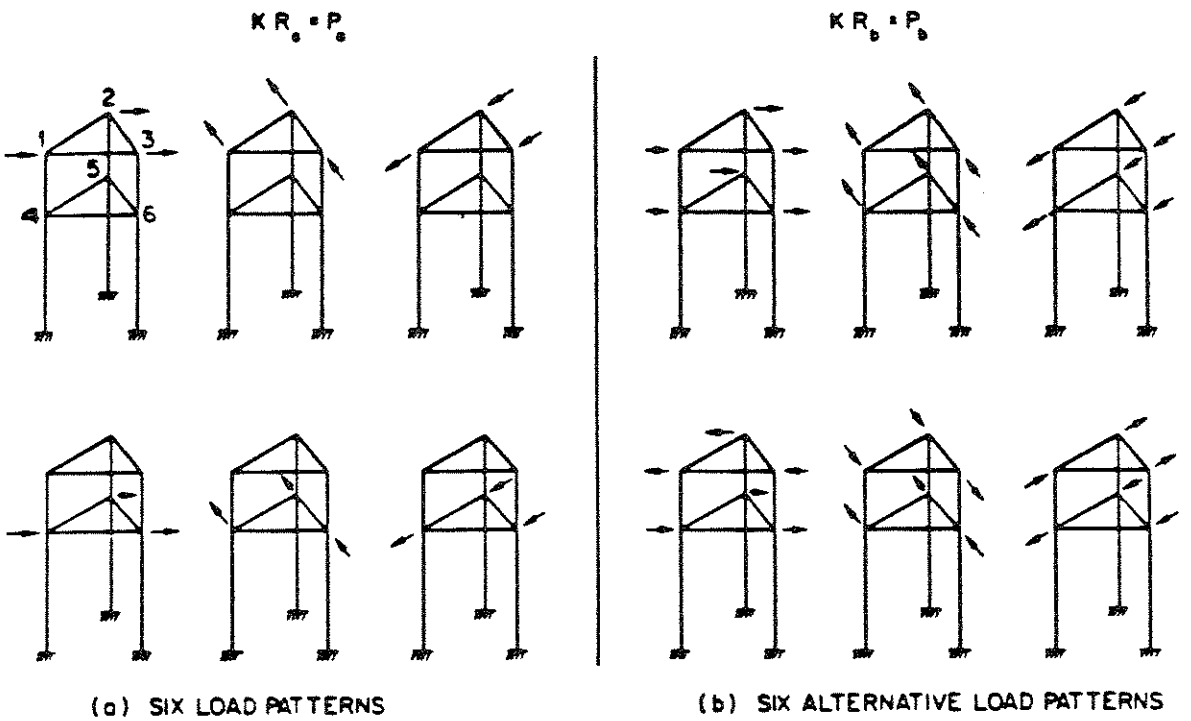


Figure 6.3 Example of selection of different load patterns for tower structure

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.[16,17,18] It is the only modern computer method for very large systems (over 5000 degrees of freedom) which will converge to the exact eigenvalues.

Table 6.2 summarizes the subspace iteration method. The computer implementation of the method and a FORTRAN listing is given in Reference 2. From Table 6.2 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

$$\phi^{(1)} = R_1 X_1 \tag{6.47}$$

an improved set of load patterns can be calculated from

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

Table 6.2 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:

$$K = LDL^T \qquad (N \times N)$$

3. Specify initial load patterns:

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

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

1. Solve for Ritz vectors R_k:

$$LDL^T R_k = P_k \qquad (N \times L)$$

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:

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

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.

It is assumed that P_2 is an improved estimation of the inertia forces associated with the basic mode shapes. From Table 6.2 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:

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

(ii) 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.

(iii) The size of the subspace L should be approximately 30 per cent more vectors more than the number of accurate eigenvalues required.

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

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

(vi) 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.

(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.

(i) Inverse iteration

If only one load pattern is used in the iteration procedure given in Table 6.2 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

$$\lambda_1 = \omega_1^2 \simeq K^{(k)}/M^{(k)} \quad . \tag{6.49}$$

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.

(ii) Determinant search

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

$$[K - \lambda M]X = 0 \tag{6.50}$$

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

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

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 6.4. This function $p(\lambda)$, which in this case is evaluated numerically, is a plot of the characteristic polynomial. $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \ldots \lambda_N$, are the eigenvalues of the system and correspond to zero values of the det $(K-\lambda M)$.

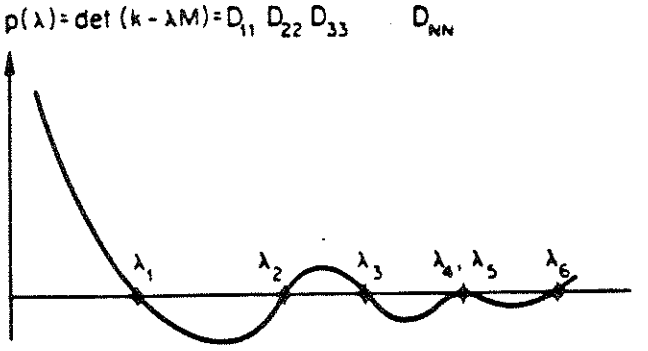


Figure 6.4 Characteristic polynomial $[K-\lambda M]$

(*iii*) *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 6.5. 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

$$\lambda_s = \mu_k + \frac{\mu_k - \mu_{k+1}}{p_1(\mu_{k+1}) - p_1(\mu_k)} p_1(\mu_k) \tag{6.53}$$

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]

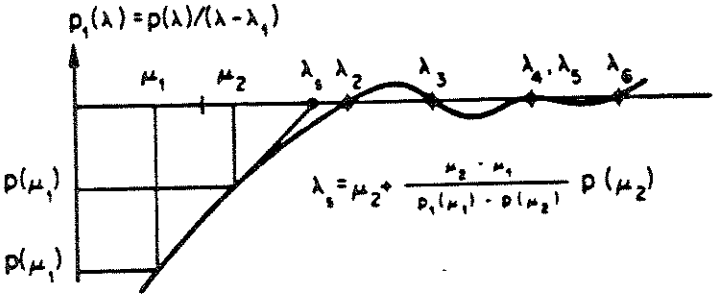


Figure 6.5 Deflated polynomial with λ_1 suppressed

(*iv*) *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

$$\lambda = \lambda_s + \rho \tag{6.54}$$

Therefore Equation (6.50) can be written as

$$[K_s - \rho M]X = 0 \tag{6.55}$$

where

$$K_s = K - \lambda_s M \tag{6.56}$$

If λ_s is closer to λ_2 than to λ_1 the inverse iteration method when applied to Equation (6.55) will converge to λ_2. This follows from the standard power method proof.[1,2]

(*v*) *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

$$\det(K - \lambda_s M) = D_{11}D_{22}D_{33}\cdots D_{NN} \tag{6.57}$$

For a given value of λ the basic properties of this sequence of numbers are illustrated in Figure 6.6. 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

$$\lambda = 1{\cdot}001\lambda_n \tag{6.58}$$

If all eigenvalues have been calculated the Sturm sequence should have n negative terms.[2] This assumes a desired accuracy of 0·001.

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.

6.2.5 Transformation to uncoupled modal equations

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

$$M^* = \phi^T M \phi = I \tag{6.59}$$

$$K^* = \phi^T K \phi = \text{diag}\,(\bar{\omega}_n^2) \tag{6.60}$$

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

$$\phi_n^T M \phi_n = 1 \tag{6.61}$$

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

$$U(t) = \sum_{n=1}^{M} \phi_n X_n(t) = \phi X(t) \tag{6.62}$$

Therefore the velocities and accelerations are

$$\dot{U}(t) = \phi \dot{X}(t) \tag{6.63}$$

$$\ddot{U}(t) = \phi \ddot{X}(t) \tag{6.64}$$

If Equations (6.62), (6.63) and (6.64) are substituted into Equation (6.2) and the resulting matrix equation premultiplied by ϕ^T we obtain

$$M^* \ddot{X}(t) + C^* \dot{X}(t) + K^* X(t) = P(t) \tag{6.65}$$

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:

$$C^* \simeq \text{diag}\,(2\xi_n \bar{\omega}_n) \tag{6.66}$$

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

$$\ddot{X}_n(t) + 2\xi_n \bar{\omega}_n X_n(t) + \bar{\omega}_n^2 X_n(t) = p_n(t) = c_n f(t) \tag{6.67}$$

After the modal equations are evaluated the time dependent displacements are calculated from Equation (6.62).

The same technique can be used to uncouple Equation (6.32) in order to avoid the solution of a large number of complex linear equations. For this case the following transformation is introduced

$$Y(\omega) = \phi Z(\omega) \tag{6.68}$$

The substitution of Equation (6.68) into Equation (6.32) yields a typical modal equation in the frequency domain.

$$[\bar{\omega}_n^2 + 2i\omega\bar{\omega}_n\xi_n - \omega^2]Z_n(\omega) = \phi_n\bar{F}(\omega) \tag{6.69}$$

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

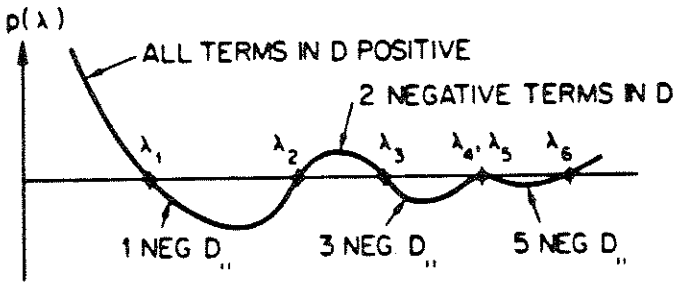


Figure 6.6 Sturm sequence properties of polynomial

6.2.5 Solution of modal equations

The solution to the single-degree of freedom modal equation given by Equation (6.67) 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 6.1 is numerically very efficient when applied to Equation (6.67).

(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 (6.69). 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 straight line loading subjected to displacement and velocity initial conditions. Therefore exact numerical values at any convenient time interval can be calculated. Table 6.3 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 equation 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.

$$\ddot{v}(t)+2\omega\xi\dot{v}(t)+\omega^2 v(t)=f(t) \tag{6.70}$$

A typical plot of the maximum, $v(\max)$, for specified ω and damping ratios ξ is shown in Figure 6.1.

The typical modal equation is of the form

$$\ddot{X}_n+2\bar{\omega}_n\xi_n\dot{X}_n+\bar{\omega}_n^2 X_n=\phi_n F(t)=c_n f(t) \tag{6.71}$$

Therefore the maximum modal response can be calculated from

$$X_n(\max)=c_n v_n(\max) \tag{6.72}$$

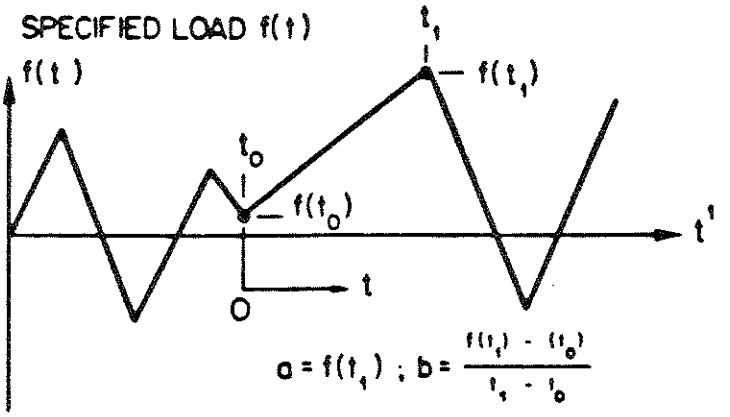
where $v_n(\max)$ is the value obtained from Figure 6.1 for values of ω_n and ξ_n.

Table 6.3 Piecewise exact solution method

BASIC EQUATION

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

SPECIFIED LOAD f(t)



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

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

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

$$A_2 = X(t_0) - A_0$$

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

EXACT SOLUTION FOR VELOCITY

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

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

$$U(\max)_n = \phi_n X_n(\max) \tag{6.73}$$

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

$$u = \sqrt{(u_1^2 + u_2^2 + u_3^2 + \cdots u_m^2)} \tag{6.74}$$

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

$$\sigma(\max) = TU(\max)_n \tag{6.75}$$

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

$$\sigma = \sqrt{(\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \cdots \sigma_M^2)} \tag{6.76}$$

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

6.3 NON-LINEAR ANALYSIS

For offshore fixed structures several different types of non-linear behaviour for both static and dynamic loads are possible. Non-linear behaviour 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 non-linearities 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 non-linear 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 non-linear behaviour is due to non-linear 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 non-linear under low stress levels. Under dynamic loads soil non-linearities are often approximated by an effective damping factor to be used in a linear dynamic analysis. Also during earthquake or large sea conditions some non-linear material behaviour can be tolerated without the collapse of the structure.

One of the most unlikely types of non-linear behaviour to be expected in a practical structure is the existence of large strains which require an alternate

definition of stress. This type of non-linearity 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 particle velocities the wave forces are non-linear and may be expressed as

$$F(t) = F(U(t), \dot{U}(t)) \tag{6.77}$$

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

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta t\, \ddot{U}_t \tag{6.78}$$

and a good estimate of the non-linear 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 6.1 can be used for this form of non-linearity.

A general formulation for the non-linear analysis of a structural system can be developed if Equation (6.1) is rewritten at time

$$(F_t^I + \Delta F_t^I) + (F_t^d + \Delta F_t^d) + (F_t^s + \Delta F_t^s) = F_{t+\Delta t} \tag{6.79}$$

in which the force changes are given by

$$\Delta F_t^I = M_t\, \Delta \ddot{U}_t, \quad \Delta F_t^d = C_t\, \Delta \dot{U}_t, \quad \Delta F_t^s = K_t U_t \tag{6.80}$$

where M_t, C_t and K_t are the approximate mass, damping and stiffness matrices at time t. Therefore Equation (6.79) can be rewritten as

$$M_t\, \Delta \ddot{U} + C_t\, \Delta \dot{U}_t + K_t\, \Delta U_t = F^* \tag{6.81}$$

where

$$F^* = F_{t+\Delta t} - F_t^I - F_t^d - F_t^S \tag{6.82}$$

A direct step-by-step method, as presented for linear systems, can be used for the evaluation of $\Delta \ddot{U}_t$, $\Delta \dot{U}_t$ 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:

(a) From the displacement U_t calculate in member m the strain ε_m.

(b) From the specified non-linear stress strain behaviour calculate the stress τ_m.

(c) 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.

(d) Calculate the total structural forces at time t from

$$F_t^s = \sum F_m$$

For structures with slight non-linearities this type of analysis may be accurate. However for very non-linear behaviour 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 non-linear analysis methods the reader is referred to References 12, 20, 21, 22, 23, 24.

6.4 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 offshore structural systems.[24,25,26] General purpose programs for non-linear analysis have been developed based on the techniques presented.[22,24] However for non-linear analysis of complex offshore structures the development of special purpose computer programs is justifiable because of the unique nature of the structures and their loading.

REFERENCES

1. Clough, R. W. and Penzien, J. (1975). *Dynamics of Structures*, McGraw-Hill Book Company, New York, NY.
2. Bathe, K. J. and Wilson, E. L. (1976). *Numerical Methods in Finite Element Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ.
3. Biggs, J. M. (1964). *Introduction to Structural Dynamics*, McGraw-Hill Book Company, New York, NY.
4. Hurty, W. C. and Rubinstein, M. F. (1964). *Dynamics of Structures*, Prentice-Hall, Inc., Englewood Cliffs, NJ.
5. Collatz, L. (1966). *The Numerical Treatment of Differential Equations*, Springer-Verlag, New York, NY, 116.
6. Crandall, S. H. (1956). *Engineering Analysis*, McGraw-Hill Book Company, New York, NY.
7. Forberg, C. E. (1969). *Introduction to Numerical Analysis*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.
8. Newmark, N. M. (1959). 'A method of computation for struetural dynamics', ASCE, *Journal of Engineering Mechanics Division*, **85**, 67–94.
9. Houbolt, J. C. (1959). 'A recurrence matrix solution for the dynamic response elastic aircraft', *Journal of Aeronautical Science*, **17**, 540–550.

10. Wilson, E. L. (1960). 'A computer program for the dynamic stress analysis of underground structures', *Report UC SESM 68-1*, Department of Civil Engineering, University of California, Berkeley.
11. Wilson, E. L. (1969). 'Elastic dynamic response of axisymmetric structures', *Report UC SESM 69-2*, Department of Civil Engineering, University of California, Berkeley.
12. Wilson, E. L., Farhoomand, I., and Bathe, K. J. (1973). 'Non-linear dynamic analysis of complex structures', *International Journal of Earthquake Engineering and Structural Dynamics*, **1**, 241–252.
13. Bathe, K. J. and Wilson, E. L. (1973). 'Stability and accuracy analysis of direct integration methods', *International Journal of Earthquake Engineering and Structural Dynamics*, **1**, 283–291.
14. Wilson, E. L. (1974). 'The static condensation algorithm', *International Journal of Numerical Methods in Engineering*, **8**, 199–203.
15. Shubinski, R. P., Wilson, E. L., and Selna, L. G. (1966). 'Dynamic response of deepwater structures', *Civil Engineering in the Oceans*, ASCE Conference, San Francisco.
16. Bathe, K. J. (1971). 'Solution Methods of Large Generalized Eigenvalue Problems in Structural Engineering', *Report UC SESM 71–20*, Civil Engineering Department, University of California, Berkeley.
17. Bathe, K. J. and Wilson, E. L. (1972). 'Large eigenvalue problems in dynamic analysis', ASCE, *Journal of Engineering Mechanics Division*, **98**, 1471–1485.
18. Bathe, K. J. and Wilson, E. L. (1973). 'Eigensolution of large structural systems with small bandwidth', ASCE, *Journal of Engineering Mechanics Division*, **99**, 467–479.
19. Bathe, K. J. and Wilson, E. L. (1973). 'Solution methods for eigenvalue problems in structural mechanics', *International Journal for Numerical Methods in Engineering*, **6**, 213–226.
20. Chopra, A. K. (1974). 'Earthquake analysis of complex structures', *Proceedings ASME Conference, Applied Mechanics in Earthquake Engineering*, November 1974, New York.
21. Felippa, C. A. (1976). 'Procedures for computer analysis of large non-linear structural systems', *International Symposium on Large Engineering Systems*, University of Manitoba, Winnipeg, Canada, August 9–12, 1976.
22. Bathe, K. J., Ozdemir, H., and Wilson, E. L. (1974). 'Static and dynamic geometric and material nonlinear analysis', *Report UC SESM 74-4*, Department of Civil Engineering, University of California.
23. Bathe, K. J. (1976). '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.
24. Bathe, K. J. (1975). '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.
25. Bathe, K. J., Wilson, E. L., and Peterson, F. E. (1973). '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

USE OF A MS-DOS COMPUTER IN STRUCTURAL ENGINEERING ANALYSIS

TABLE OF CONTENTS

1. SUMMARY

The major purpose of this "small document" is to provide the essential information for a new user/engineer to become productive on a DOS microcomputer with a minimum investment of time. It is assumed that the engineers objectives are to use existing programs, develop simple FORTRAN programs and to prepare data input files and technical documentation. In order to simplify the presentation only computers with large capacity fixed disks and a minimum of one removable floppy disk will be considered.

Only a small number of the most important DOS commands will be presented. In addition, a few special purpose DOS programs are given which should help during the initial learning process. The simple easy-to-use open-screen editor, EDED, is presented as an alternative to the cumbersome line editor, EDLIN. This document is not intended to be a replacement for the MS-DOS user's manual, the MS-FORTRAN reference manual or other professional editors or technical word processors.

2. INTRODUCTION TO DOS

MS-DOS (Microsoft Disk Operating System) is the operating system for many computers with Intel 8088 to Intel 80486 series of Central Processing Units. The DOS monitor allows all computer programs rapid access to the DOS file management system. Since certain files are independent of the programming language it is possible to transfer information between programs which have been written in different programming languages such as FORTRAN, BASIC or C.

Files are stored on large capacity fixed disks or removable floppy disks. Each disk has a root "directory" and "subdirectories" which indicate the name, size and location of all files on the disk. System disks have the basic DOS operating system on a reserved section of the disk. When the computer is started or "booted" the DOS system and the COMMAND.COM are read into the computer, control is transferred to the DOS monitor program, and the "C>" (or "A>") prompt is displayed on the console indicating that "C" is the currently logged-in drive and that the DOS system is awaiting a command. The user can then enter any legal DOS operation followed by a carriage return (CR).

A DOS command is either one of the INTERNAL commands, contained in the COMMAND.COM file, or it is a request to execute a program which is stored on any disk drive (and directory) which is connected to the computer system. Also, the user can switch to another "logged disk" by typing the new drive name (A,B,C or D) followed by a colon (:) and terminated by a CR. Also, the user can transfer control to another directory with the change directory, CD, command.

FILE NAMES

A file name consists of two parts: the primary name (1 to 8 characters) and an extension name (1 to 3 characters) separated by a period. For example, the FORTRAN source program for the computer program CAL would be named CAL.FOR; its binary relocatable file would be named CAL.OBJ and its executable (command) file would be named CAL.EXE. The general form of a file name is

"drive":\path\"name"."extension"

If the "drive" is not specified it is taken as the currently logged drive. If the path is not specified it is taken as the current directory. The following symbols are not allowed as characters within a file name:

< > . , ; : = * [ ] / \

An ambiguous file name is used to refer to one or more files on a disk. For example:

CAL.*	References all files on the disk with the first name CAL
*.COM	References all files on the disk with the second name COM
CA*.FOR	References all .FOR files which has CA as the first two letters of its name

SPECIAL DOS CONTROL CHARACTERS

During the execution of DOS programs the following control characters can be entered at the keyboard:

control C	Causes termination of the execution of the program
control P	All subsequent screen output is also directed to the printer until the next control P is typed
control S	Screen output is stopped temporarily in order to view a segment of output. Screen output will continue when any other character is typed.
Prt Sc	Causes the current screen display to be printed

3. INTERNAL DOS COMMANDS

The following commands are built into DOS and are available when the COMMAND.COM program is loaded:

DIRECTORY COMMAND

DIR Causes the names of all files on the currently logged disk to be listed on the console.

DIR A:*.FOR Causes the names of all FORTRAN files on the disk mounted on drive A to be listed on the console.

The D program, given on page 4, can be used in place of DIR.

DELETE COMMAND

DEL XX.EE The file named XX.EE is removed from the current directory and its disk space is no longer reserved.

DEL A:*.BAKAll files in the current directory on the disk mounted on drive A with the second name BAK are removed.

DEL *.* All files are removed from the currently logged disk.

RENAME COMMAND

The name of a file on a disk may be changed by the REN command. For example:

REN ZQY.AA XX Causes the file ZQY.AA to be renamed to XX.

TYPE COMMAND

The TYPE B:XX.YY command causes the contents of the file named XX.YY on drive B to be displayed on the console. Control S will cause the display to be halted temporarily. The TYPE XX.YY|MORE} will cause the display to be halted ever 24 lines.

COPY COMMAND

The COPY program is one of the most important programs which operates under the DOS system. It allows files to be duplicated and copied to other directories, disks or output devices. As examples:

```
COPY  CAL.FOR  CAL.BAK
```

The above command will create a new file CAL.BAK, within the current directory, which is identical to the file CAL.FOR.

```
COPY  *.*  A:\ZZ\
```

The above command will copy all files from the current directory to directory ZZ on drive A. The new files on A will have the same names.

```
COPY  *.FOR  A:
```

The above command will copy all files with extensions .FOR from the current directory to drive A. The files copied to drive A will replace files which previously existed with the same name.

DIRECTORY OPERATIONS

It is possible to use a DOS computer system without defining subdirectories. For such a system all programs and data files would exist in the "root directory" and when the DIR command is given a list of all files would be displayed. In addition, a potential problem with conflicting file names exists if several individuals use such a system. Also, systems without subdirectories are cumbersome to use and to maintain. It is very convenient to create separate directories for special functions. For other problems it is useful to create temporary work areas containing files which may be easily deleted after the problem is completed.

For most systems only one level of subdirectories is required. This approach greatly simplifies the use of "path names" and the general use and maintenance of a system. However, a good approach is to retain a minimum number of files in the root directory and to allow access to the important programs by creating a PATH to all directories which contain these programs with the use of the SET PATH command.

The DOS command MD new will create a new subdirectory, or work area, on the current disk named new which will be able to store data and DOS programs.

After a directory is made it is possible for the user to transfer control to that subdirectory by executing the command CD \new. To return to the root directory the command CD \ is executed.

If all files are deleted from a subdirectory and control is in the root directory the execution of RD new will delete the directory new.

4. ADDITIONAL DOS PROGRAMS

All computer programs which can be executed on a DOS system are stored as files of the form NAME.COM or NAME.EXE and are executed by typing the command NAME. If the program is on a drive other than the logged drive it is executed by the specification of the drive name. For example: A:xxxx will cause the program xxxx to be loaded from drive A and executed. The following is a partial list of some of the most useful programs (DOS commands).

FORMAT PROGRAM

This program must be used to initialize the density and record size for every new diskette. The standard international interchange format is the 360k double sided double density 5 1/4 inch disk. However, for maximum efficiency and speed, disks which are to be used locally on one computer system should be initialized at maximum capacity. The FORMAT operation defines the density of a disk which may be different than what the maker of the disk writes on the label.

DISKCOPY and DISKCOMP PROGRAM

The DISKCOPY is used to copy all files on one disk to another disk and FORMATS the new disk during the process. The DISKCOMP program is used to compare the contents of two disks.

PRINT and PSET PROGRAMS

The PRINT program can be executed at the same time as other DOS commands or programs are being executed. Therefore, files can be printed in a "background mode" and the user can perform other tasks without delay. In addition, the printing of several files can be initiated at the same time as indicated in the following:

PRINT file1 file2 file3 - - -

The command PRINT/T can be used to terminate printing after the completion of the current file. The PSET program can be used with dot matrix printers to select print type and skip perforation options prior to the use of the PRINT program.

D PROGRAM

The D program performs the same general function as the DIR program; however, the names of the files are presented in alphabetical order and in a more compact form on the screen.

WHEREIS PROGRAM

The execution of WHEREIS "name" command will locate in which subdirectories the file "name" exists. This program is very useful in locating duplicate copies of files and programs.

5. DOS BATCH CAPABILITY

This basic DOS function allows a series of DOS operations and user programs to be executed in sequence without the requirement that the user type the series of DOS commands. In order to utilize this option the series of commands must be stored in a "submit file" which is prepared by an editor. The second name of the batch file must be ".BAT". To illustrate this option let us assume that the series of programs SAP, FRAME, SOLVE and FRAMEF are to be executed in sequence. First: prepare a "batch file" with the name "SERIES.BAT" which contains the following information:

SAP
FRAME
SOLVE
FRAMEF

Next: type the DOS command SERIES and the above list of programs will be executed in sequence without the computer stopping after each program.

Execution of the batch operation can be terminated by typing a CONTROL C. The batch operation is extremely useful in "linking together" typically used DOS operations.

The more general form of the batch operation program is one which has several parameters (arguments) which are specified as variables within the .BAT file as indicated below:

"batch file name" P1 P2 - - - Pn

The variables %1 %2 - - within the batch file are replaced by names P1 P2 - - during the execution of the BATCH operation. An example of the use of this form of the batch operation is to erase a series of files which are generated by the program SAP without deleting the input data file which has the same first name and no extension. First, the following file named ER.BAT is prepared:

```
DEL TEMP
REN %1 TEMP
DEL %1.*
REN TEMP %1
```

If the input data file is named FRMEX, then all files with the first name FRMEX must be removed except the data file itself. Therefore, the command

ER FRMEX

will perform this function and eliminate the need to type four DOS commands. The practicality of the batch operation is only limited by the creativity of the user.

6. CONFIG.SYS, COMMAND.COM and AUTOEXEC.BAT

When a DOS computer system is booted (started), or manually rebooted, DOS looks for the file CONFIG.SYS in the root directory. If the file is found, DOS interprets the information within the file in order to configure the DOS system parameters or to select a different COMMAND processor (SHELL). If the CONFIG.SYS file is not found the default values, shown below within [ ], are used.

If a different COMMAND processor is not defined the file COMMAND.COM is loaded from the root directory. Then, DOS automatically searches the root directory for the file AUTOEXEC.BAT and, if found, executes the DOS commands contained within the file.

The files CONFIG.SYS and AUTOEXEC.BAT are optional files; however, their existence is very important if the computer system is to be configured and initialized for professional use. Both files can be prepared or modified by the editor EDED.

A typical CONFIG.SYS file, which is shown below, contains the five different options which are possible:

```
BREAK=ON          [OFF] (allows CTRL/BREAK to terminate DOS operations)
BUFFERS=40        [10]  (RAM buffers reserved for I/O operations)
FILES=15          [12]  (files which can be opened concurrently)
DEVICE=ANSI.SYS         (defines extended screen and keyboard functions)
SHELL=COMMAND.COM       (defines standard COMMAND processor)
```

A typical AUTOEXEC.BAT file is shown below:

```
VERIFY ON                          (all COPY operations will be verified)
SET PATH=C:\;C:\DOS;C:\SAP;C:\CAL;      (sets path)
SET PROMPT=$36$P$36$G              (DOS prompt will display directory name)
GRAPHICS                           (allows screen print of graphics)
VER                                (displays version of DOS)
```

The PATH command is one of the most useful commands in the DOS system. If a DOS program is executed, when operating within a directory on any disk, and the program is not stored within the directory disk space DOS will automatically search for the program in other directories in the order defined by the SET PATH operation. Therefore, there is no reason why duplicate copies of programs should exist in different directories.

7. FILE Preparation- The EDED Editor

The program which is used most often by an engineer at a microcomputer workstation is the "editor". This is normally a highly interactive program which allows the user to prepare input data files and to examine or modify any existing printable file. Therefore, the capability of this program is very important if the user/engineer is to be productive.

The editor "EDLIN", which is supplied with the DOS system, is a very difficult program to use. An engineer with a minimum of computer experience requires approximately two weeks to learn to effectively use this line oriented editor. In addition, the possibility of data input errors is greatly increased if EDLIN is used.

There is a large number of commercial editors available for DOS systems which can be purchased for $20 to $500. In most cases they have a large number of useful options; however, a significant amount of time is required to learn to use them productively. In addition, many of these editors create files with embedded "non-print" characters which are incompatible with standard FORTRAN input files.

The editor EDED, presented here, is designed to be used with a minimum investment of time. It is a simple open-screen editor with a limited number of commands. This editor was written by E. C. Gillott and is distributed by ShareWare and users are encouraged to support his future developments. A complete documentation of the editor can be obtained by PRINTings the EDEDDOCS.TXT file. Or this file exists in the current directory it can be read online within EDED with the F4 key.

The EDED program cannot create or edit files larger than 32k bytes.