

AD-753 120

FEABL (FINITE ELEMENT ANALYSIS BASIC
LIBRARY) USER'S GUIDE

Oscar Orringer, et al

Massachusetts Institute of Technology

Prepared for:

Air Force Office of Scientific Research

August 1972

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

AFOSR TR 72-2228

ASRL TR 162-3

AD753120

FEABL (FINITE ELEMENT ANALYSIS BASIC LIBRARY) USER'S GUIDE

Oscar Orringer
Susan E. French

AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

AUGUST 1972

Prepared for

**AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNITED STATES AIR FORCE**

Under

CONTRACT NO F44620 70 C 0020

Approved for public release,
distribution unlimited.

NATIONAL TECHNICAL
INFORMATION SERVICE

ACCESSION for	
NTIS	White Section ☒
DDC	Blue Section ☐
UNCLASSIFIED	☐
DISTRIBUTION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist	Avail. and/or SPECIAL
A	

NOTICES

Qualified requestors may obtain additional copies from the Defense Documentation Center, all others should apply to the National Technical Information Service.

Reproduction, translation, publication, use, and disposal in whole or in part by or for the United States Government is permitted.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) MASSACHUSETTS INSTITUTE OF TECHNOLOGY AEROELASTIC & STRUCTURES RESEARCH LABORATORY CAMBRIDGE, MASSACHUSETTS 02139		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP	
3. REPORT TITLE FEABL (FINITE ELEMENT ANALYSIS BASIC LIBRARY) USER'S GUIDE			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Scientific Interim			
5. AUTHOR(S) (First name, middle initial, last name) OSCAR ORRINGER SUSAN E FRENCH			
6. REPORT DATE Aug 1972		7a. TOTAL NO. OF PAGES 105	7b. NO. OF REFS 5
8a. CONTRACT OR GRANT NO. F44620-70-C-0020		9a. ORIGINATOR'S REPORT NUMBER(S) ASRL-TR-162-3	
b. PROJECT NO. 9782-02			
c. 61102F			
d. 681307		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned, this report) AFOSR-TR-72-2228	
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited			
11. SUPPLEMENTARY NOTES TECH, OTHER		12. SPONSORING MILITARY ACTIVITY AF Office of Scientific Research (NAM) 1400 Wilson Blvd. Arlington, Va. 22209	
13. ABSTRACT This guide contains complete instructions for the use of FEABL, a basic software system developed for finite element analysis at the MIT Aeroelastic and Structures Research Laboratory. FEABL has been designed primarily for (but is not limited to) the specialized type of continuum analysis problem encountered in the materials laboratory. FEABL has also been used successfully as an educational tool in the finite element analysis course given by the MIT Department of Aeronautics and Astronautics. The software is modular, and is written in machine-independent FORTRAN IV. A complete program listing is contained in Appendix C. Element subroutines which can be used either independently or in conjunction with FEABL will be presented in future publications.			

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
FINITE ELEMENT ANALYSIS						
STRUCTURAL ANALYSIS						
NUMERICAL METHODS						
FINITE ELEMENT SOFTWARE						
MATRIX METHODS						

IB

UNCLASSIFIED

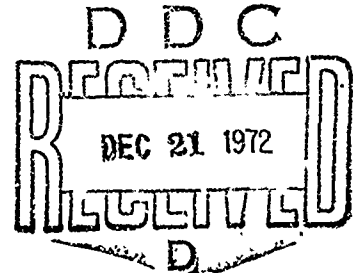
Security Classification

AFOSR TR
ASRL TR 162-3

FEABL
(FINITE ELEMENT ANALYSIS BASIC LIBRARY)
USER'S GUIDE

Oscar Orringer
Susan E. French

August 1972



Prepared for
Aeromechanics Division
Air Force Office of Scientific Research
Arlington, Virginia 22209
Under
CONTRACT NO. F44620-70-C-0020.

Approved for public release; distribution unlimited.

Aeroelastic and Structures Research Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

FOREWORD

The developments documented in this report were carried out at the Aeroelastic and Structures Research Laboratory, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts under Contract No. F44620-70-C-0020 from the Office of Scientific Research, U.S. Air Force. The software system originated in connection with Dr. Orringer's need to use finite-element analysis for problems in the compressive behavior of fiber composites that were being investigated under the Contract. Dr. Jacob Pomerantz of the Aeromechanics Division, AFOSR served as technical monitor.

The authors wish to express their appreciation to Prof. Pin Tong of the Department of Aeronautics and Astronautics for the numerous helpful suggestions he made regarding the design of the FEABL software system. The authors are also indebted to all of the students in the Departments 1971-72 finite element analysis course who, under Prof. Tong's guidance, acted as guinea pigs while the software was still being tested. For his review of this report and for attendant constructive suggestions, the authors are indebted to Professor Emmett A. Witmer of the MIT-ASRL.

The computations and testing of the program were carried out at the MIT Information Processing Center.

ABSTRACT

This guide contains complete instructions for the use of FEABL, a basic software system developed for finite element analysis at the MIT Aeroelastic and Structures Research Laboratory. FEABL has been designed primarily for (but is not limited to) the specialized type of continuum analysis problem encountered in the materials laboratory. FEABL has also been used successfully as an educational tool in the finite element analysis course given by the MIT Department of Aeronautics and Astronautics. The software is modular, and is written in machine-independent FORTRAN IV. A complete program listing is contained in Appendix C. Element subroutines which can be used either independently or in conjunction with FEABL will be presented in future publications.

CONTENTS

<u>Section</u>	<u>Page</u>
1 INTRODUCTION	1
1.1 Purpose and Scope	1
1.2 General System Concepts	4
1.3 General System Organization	4
2 THE FEABL DATA STORAGE SYSTEM	7
2.1 Accessory Data	7
2.1.1 Input-Output Control Parameters	7
2.1.2 Problem Size Parameters	7
2.1.3 Address Index Parameters	8
2.2 The /DATA/ Vector	8
2.2.1 Constraint Vector	9
2.2.2 Address Count Vector	10
2.2.3 Leading Non-Zero Entry Vector	10
2.2.4 Master Assembly List	11
2.2.5 Force/Displacement Vector	15
2.2.6 Master Stiffness Matrix	17
2.3 Modification of the FEABL Data Storage System	19
3 FEABL PROCESS CONTROL	21
3.1 General	21
3.2 Description of the FEABL Software	22
3.2.1 Housekeeping Setup Subroutine for /DATA/ Vector (SETUP)	24
3.2.2 Subroutine for Detailed Organization of the Master Stiffness Matrix Block (ØRK)	25
3.2.3 Element Assembly Subroutine (ASEMBL)	26
3.2.4 Rotation Transformation Subroutine (RØTATE)	28
3.2.5 Boundary Constraint Subroutine (BCØN)	32

CONTENTS (Continued)

<u>Section</u>		<u>Page</u>
	3.2.6 Stiffness Matrix Factoring Subroutine (FACTPD/FACTSD)	34
	3.2.7 Subroutine for Solution of Simultaneous Equations (SIMULQ)	36
	3.2.8 Subroutine for Extraction of Element Displacements from the Global Displace- ment Vector (XTRACT)	37
4	A SAMPLE FEABL PROGRAM	39
5	ADVANCED TECHNIQUES WITH FEABL	46
	5.1 Efficient Programming for Large Problems	46
	5.2 A Numbering Strategy for Planar Problems	48
	5.3 A Short Note on Automatic Data Generation	51
	5.4 A Handy Trick	52
	5.5 Time-Saving Techniques for Design Studies	53
	5.6 Substructuring with FEABL	57
	5.7 Error Estimation Methods	59
	5.7.1 Irons' Energy Variance Criterion	59
	5.7.2 The Residual Force Method	60
	5.7.3 Re-Solution for Residual Displacements	63
	5.7.4 The Method of Rigid Body Modes	64
	REFERENCES	66
	APPENDICES	
A	COMMON Areas Data Requirements	67
B	Applying Shohorn and Stopwatch	68
	B.1 Estimation of Core Storage Requirement	68
	B.2 Estimation of CPU Time Requirement	69
C	FORTTRAN IV Listing of FEABL Software	70

SECTION 1

INTRODUCTION

1.1 Purpose and Scope

Finite element analysis programs developed in the past fall into two general categories: the "one-shot" program optimized for solution of a particular problem, or the "systems" program designed for the non-optimum solution of any problem. The structural engineer who has a new analysis to perform must spend a great amount of time either writing his own "one-shot" program or learning how to use one of the general "systems". FEABL attempts to relieve the analyst of these burdens by providing a basic library which handles the numerical operations common to all finite element analysis, and which can be understood and used after a few hours of study. FEABL's current capabilities are limited to static analysis. Other options will be added in the future, as the need arises, and according to their usefulness. This guide is written for the analyst who has some acquaintance with the theory of finite element analysis and with IBM FORTRAN IV.

THE FEABL software system has been designed primarily for "materials laboratory" stress analysis. The analysis problems encountered in this application area are characterized by geometry which, although not simple, is mathematically describable, and by stress solution accuracy requirements which can usually be met with models of 1,000 or fewer degrees of freedom. Stress concentration analysis for two-material systems and for composite-to-metal step-lap joints are two examples of the materials laboratory type of problem. Another feature of many of these problems is a requirement for nonstandard solution methods. For instance, the boundary conditions may include an undetermined prescribed displacement, the value of which depends upon satisfaction of an

auxiliary condition on a boundary stress integral; a specific example is given in Ref. 1. The main features of FEABL, all-in-core computation and control of the solution process by the user, are well adapted to the needs of such problems.

Experience in the MIT Department of Aeronautics and Astronautics has also shown that FEABL is a valuable educational tool. A budget of \$50 to \$70 per man on an IBM 370/155 machine allows each student to solve one or two problems, each having 200 to 300 degrees of freedom. In the approach taken at MIT, the student is required to program his structure geometry, element interconnections, and subroutines for generating stiffnesses and stresses. He must also achieve a general understanding of the principles upon which the FEABL subroutines are based. Thus, each student is able to gain some practical experience, as well as theoretical knowledge in one semester of a finite element analysis course. This approach is felt to be superior to the traditional method of having the student learn only how to code the input data for a particular, complete "systems" program.

The flexibility and economy of the FEABL software system has been amply demonstrated by various applications in which real problems were solved during the test/exercise development phase. The following are examples abstracted from these applications:

Problem Description	Total DOF	CPU Time (Min)	Total Cost (\$)
Displacement analysis of two large rigid frame structures (6 load cases each)	216	0.42	5.37
	426	2.49	13.17
Continuum stress concentration analysis with boundary condition determined by auxiliary stress integral (3 complete solutions required for each case)	338	1.50	4.50
	442	2.70	12.00
Stress analysis of a large frame in an oil tanker, using rectangle and triangle continuum elements and flange elements (6 loading cases)	1,266	3.0 (Approx.)	14.00
Pressure distribution analysis for viscous flow around a corpuscle in a blood vessel	160	No data available	
Stress analysis, including solution for stress intensity factor of a sharp crack, using hybrid element at the crack	160	0.13	2.54

The above examples were run on the IBM 370/155 at the MIT Information Processing Center. The costs presented are for production runs, exclusive of debugging.

1.2 General System Concepts

Development of the FEABL software has been based on three general concepts. First, FØRTRAN IV was selected as the program language, and all FØRTRAN syntax rules were followed rigorously. This makes FEABL as machine-independent as possible, to allow for use at installations having other than IBM equipment. The only restrictions are that the FØRTRAN compiler software must permit logical IF statements, variable dimensioning, six-character variable names, and labelled CØMMØN declarations. Second, the software is modular in character. The finite element analysis process has been decomposed into a series of distinct steps, and each step which is independent of structure geometry has been programmed as a separate subroutine. FEABL is thus analogous, in a sense, to the IBM Scientific Subroutine Package.

The third concept deals with data storage techniques. In order to minimize wasted storage space, a vector approach has been adopted. All of the input data for a problem, certain "housekeeping" data, and internally generated problem data such as the master stiffness matrix are stored in a single vector, referred to below as the /DATA/ vector. In addition, control parameters which locate entries in the /DATA/ vector or manage input-output operations are organized in four CØMMØN groups.

1.3 General System Organization

The analyst must provide one or more programs which interface with the FEABL software:

1. A MAIN program which inputs the required problem data and controls the execution of all subprograms.
2. One or more subroutines which generate an element stiffness matrix (and, if required, equivalent nodal forces for thermal stress, gravity, etc.) for each type of element.*

* Subroutines from ASRL EGL (Element Generator Library) may be used.

3. One or more subroutines which generate element stresses from the element's nodal displacements.*

Data may be input by reading cards, disk or tape files, by automatic generation techniques, or by a combination of the two methods. The details of a particular problem will determine which approach is more efficient. For very simple types of problems, in which the structure is divided into only a few different element types and shapes, stiffness matrix and stress generation may be done within the MAIN program. However, if separate subroutines are required for these tasks, pre-existing ones may be interfaced with FEABL, since the FEABL software package has been written in a manner such that pre-existing FORTRAN subroutines may be adapted to run with FEABL with little or no re-programming.

The interface between user-written programs and FEABL consists of two parts: data location and process control. Figure 1 illustrates the general features of the data location interface. The labelled COMMON areas mentioned in the previous section must appear in the user's MAIN program. This is accomplished simply by including identical sets of DIMENSION, COMMON and EQUIVALENCE declarations in MAIN and in the FEABL software. There is no requirement for the element generator subroutines to communicate with FEABL in this manner; these subroutines need only be interfaced with user's MAIN by an identical DIMENSION declaration for the element stiffness matrix and "force/displacement" vector. The latter is a dual-purpose vector in which element equivalent nodal forces are generated, and which serves later as a storage area for the solution displacement vector of the element. The interface between the generator routines and FEABL is achieved automatically by variable DIMENSION declarations contained in the FEABL software. The data location interface is discussed in detail in Section 2.

*Subroutines from ASRL EGL (Element Generator Library) may be used.

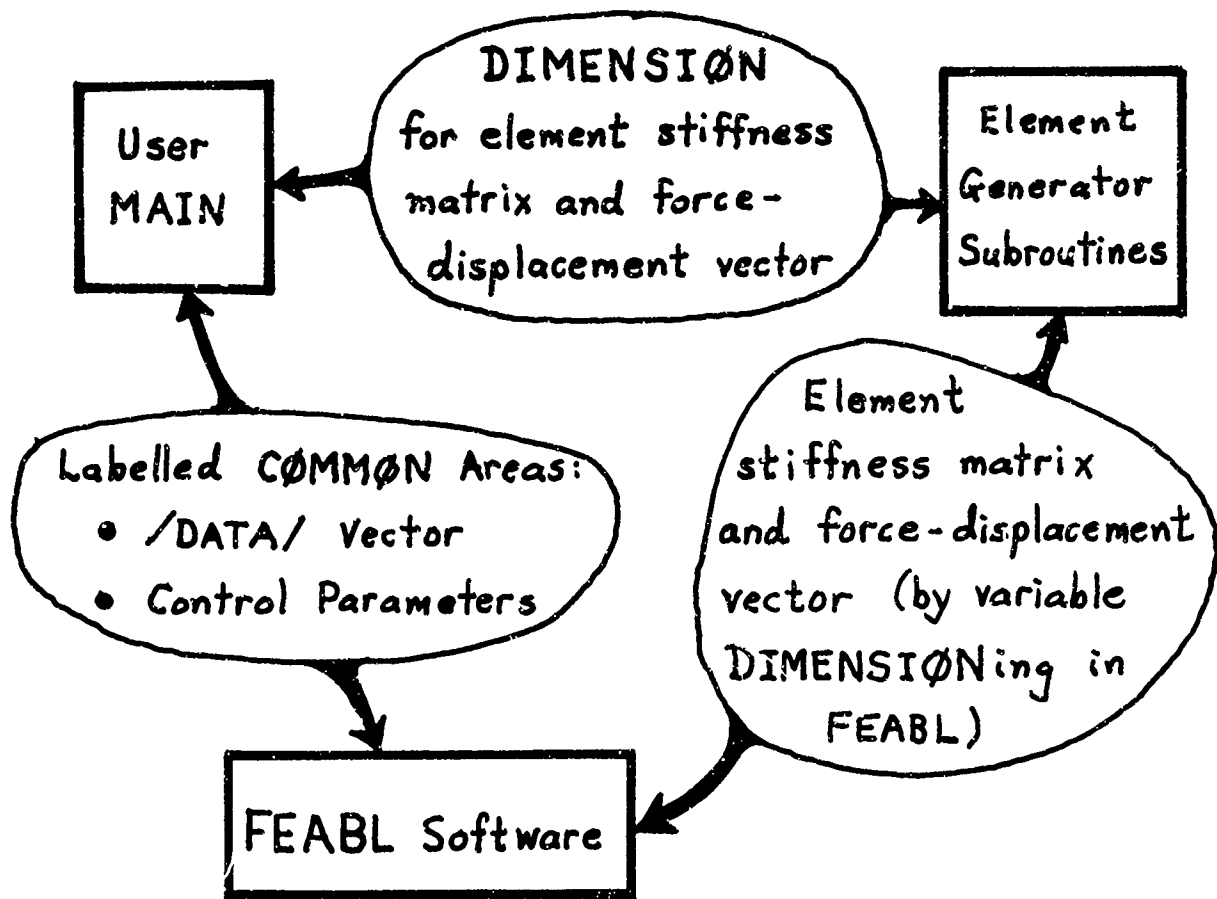


FIGURE 1

The process control interface is simply a recognition that input of problem data and operations on the data must be performed in a certain sequence. This requires a definite series of programming steps in user's MAIN program. Section 3 discusses the process control interface in detail.

SECTION 2

THE FEABL DATA STORAGE SYSTEM

2.1 Accessory Data

The accessory data is divided into four control parameter CØMMØN groups. The CØMMØN declaration statements listed below for the four control parameter groups must appear in all user-written MAIN programs.

2.1.1 Input-Output Control Parameters

CØMMØN /IØ/ KR, KW, KP, KT1, KT2, KT3

The six parameters in the /IØ/ group are hardware device codes, defined as follows:

KR - card reader

KW - printer

KP - card punch

KT1, KT2, KT3 are extra device codes which the user may define optionally as he pleases. For example, KT1 might refer to a system direct access disk, KT2 to an external tape unit, etc. Only the printer device code KW is used directly by the FEABL software presented in this guide. The input-output control parameters are used in FØRTRAN READ and WRITE statements, e.g.:

```
      READ (KR,1) NET, NDT
1  FØRMAT (2I6)
      WRITE (KW,2) NET, NDT
2  FØRMAT (16HOTØTAL ELEMENTS=, I6, 10X, 10HTØTAL DØF=, I6)
```

The correct values of KW and any other device codes employed by the user must be established at the beginning of the MAIN program.

2.1.2 Problem Size Parameters

CØMMØN /SIZE/ NET, NDT

These parameters define the problem size in terms of the total number of elements (NET) and the total number of degrees of freedom (NDT) in the whole structure. NDT includes both

constrained and unconstrained degrees. In standard solutions these parameters are established at the beginning of the MAIN program and remain fixed.

2.1.3 Address Index Parameters

```
CØMMØN /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
CØMMØN /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
```

These parameters control the begin and end locations of data sub-blocks in the /DATA/ vector. The user establishes correct values for these parameters by calling FEABL subroutine SETUP after he has established the /SIZE/ block. Use of the address index parameters to locate entries in the /DATA/ vector is discussed in Subsection 2.2.

2.2 The /DATA/ Vector

```
DIMENSION REAL(XXXX), INTGR(XXXX)
CØMMØN /DATA/ REAL
EQUIVALENCE (REAL(1), INTGR(1))
```

The above declarations serve to define the /DATA/ vector as a one-dimensional array occupying a CØMMØN area labelled /DATA/ and having two reference names: REAL for floating point entries and INTGR for integer entries. These declarations must appear in the user-written MAIN program. The user chooses the dimension integer "XXXX" to suit his particular needs. The DIMENSION declaration with user's value for the length of the /DATA/ vector must be inserted in each FEABL subprogram, immediately following the subroutine name declaration, e.g.:

```
SUBROUTINE ASEMBL (LNUM, NDE, ELK, ELQ)
DIMENSION REAL(1000), INTGR(1000)
```

Some compilers restrict the length of any vector to 32,768 words (8,000_{HEX}). If this restriction is encountered, and the user desires a longer /DATA/ vector, say 50,000 words, the following form of the declaration statements can be employed:

```
DIMENSION REAL(25000), INTGR(25000), DUMMY(25000)
CØMMØN /DATA/ REAL, DUMMY
EQUIVALENCE (REAL(1), INTGR(1))
```

The COMMON /DATA/ declaration must then be changed to the above form in each FEABL subprogram.

The dual nature of the /DATA/ vector must be recognized clearly. For example, the 100th entry in the vector may be treated as either a floating point or an integer quantity in arithmetic or logical instructions by operating, respectively, with REAL(100) or INTGR(100). This property will lead to compilation errors if the user attempts to save storage space by declaring INTGR to be an array of 2-BYTE words (half-words), while REAL is left as an array of normal 4-BYTE words (full words), or if the user attempts to run in double precision mode for floating point arithmetic.

The /DATA/ vector is organized into six blocks, the limits of which are determined by the values stored in the address index parameters (Subsection 2.1.3). In normal usage, the first four blocks contain integer data and the last two contain floating point data. Storage conventions are detailed in the following sections, in the order in which the blocks appear in the /DATA/ vector.

2.2.1 Constraint Vector

This is an integer block which contains the global number of each degree of freedom at which a displacement is to be prescribed. Entries in the constraint vector are referred to by:

INTGR(I) where $ICØN \leq I \leq LCØN$

The following example illustrates user action involving the constraint vector

```
·  
·  
INTGR(ICØN)=2  
INTGR(ICØN+1)=25  
INTGR(LCØN)=6  
·  
·
```

By means of the above instructions the user has specified that global displacements 2, 25, and 6 will be prescribed. The prescribed displacements need not be listed in any particular order. Any excess space in the constraint vector is filled with zeros by FEABL.

2.2.2 Address Count Vector

This is an integer block which contains information relating to the absolute address in the /DATA/ vector of each diagonal entry of the master (global) stiffness matrix. Entries in the address count vector are referred to by:

INTGR(I) where $IK\emptyset UNT < I < LK\emptyset UNT$

To obtain the relevant data for the Nth row, the correct subscript is:

$$J = IK\emptyset UNT + N - 1$$

INTGR(J) contains address information for row N

Normally, the user has no direct communication with the address count vector; its contents are generated internally by FEABL. The absolute address of each diagonal entry is output for use in debugging. After the output, the data in the address count vector are modified as follows:

$$\text{Contents} = (\text{Absolute Address of } K_{11}) - 1$$

Subsequently, any stored* entry K_{ij} can be obtained by referring to:

$$\text{REAL}(\text{KADR})$$

where the address is given by:

$$\text{KADR} = \text{INTGR}(IK\emptyset UNT + I - 1) + J$$

2.2.3 Leading Non-Zero Entry Vector

The LNZ vector is an integer block containing the number j of the column in which appears the leading non-zero entry K_{ij}

* Many entries of the master stiffness matrix are not stored in the /DATA/ vector. See Subsection 2.2.6 for details.

of the master stiffness matrix for each row i . Entries in the LNZ vector are referred to by:

INTGR(I) where $ILNZ \leq I \leq LLNZ$

INTGR(ILNZ+N-1) refers to row N.

Normally, the user has no direct communication with the LNZ vector; its contents are generated internally by FEABL, and are output for use in debugging and evaluating displacement numbering strategies.

2.2.4 Master Assembly List

This is an integer block containing all of the information which relates the user's local degree of freedom numbers to the user's global numbering system. The master assembly list is, in effect, a Boolean logical transformation between the element displacement vectors, which together form a linearly dependent set, and the global displacement vector, which is an independent set. Entries in the master assembly list are referred to by:

INTGR(I) where $IMASTR \leq I \leq LMASTR$

The master assembly list need not be filled, but if it is not, a zero must be stored immediately following the last active position.

The master assembly list is subdivided into two sections, as indicated in Figure 2.

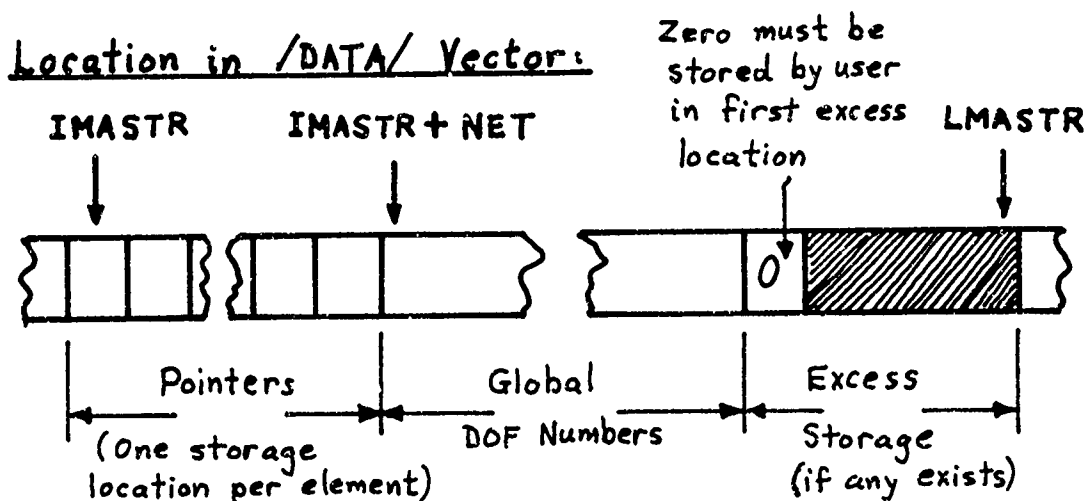


FIGURE 2

The pointer section consists of one location per element, from $\text{INTGR}(\text{IMASTR})$ to $\text{INTGR}(\text{IMASTR}+\text{NET}-1)$, with the convention that these locations correspond to the user's elements in ascending order 1, 2, ..., NET. Each pointer contains the absolute address in the /DATA/ vector of the location where the assembly list for an element starts, as indicated in Figure 3.

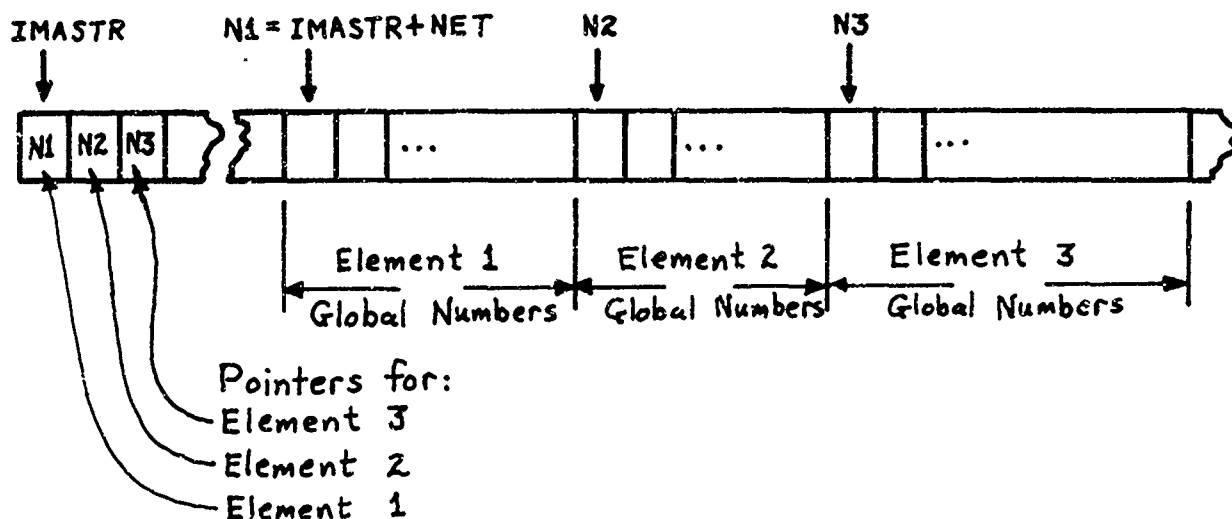
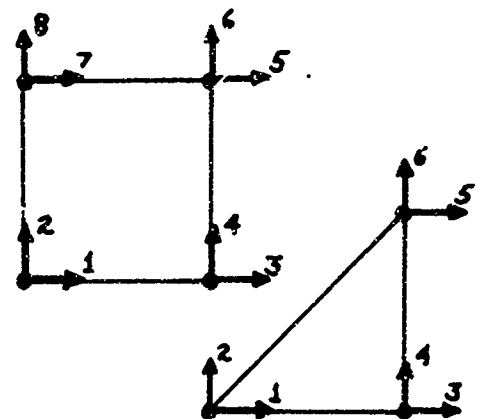
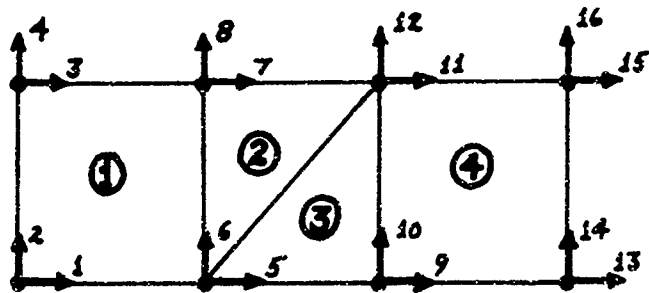


FIGURE 3

The remainder of the master list contains the user's element-by-element sequence of global displacement numbers. As an illustrative example, consider the set of elements shown in Figure 4, consisting of two 4-node rectangles and two 3-node triangles, each with two degrees of freedom per node. The total length required for the master assembly list is given by:

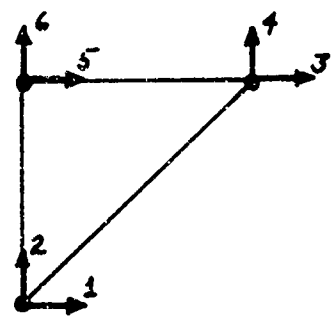
Pointers:	1 per element-----	4
Rectangles:	8 DOF/element-----	16
Triangles:	6 DOF/element-----	12
<u>TOTAL-----</u>		<u>32</u>

Suppose that $\text{IMASTR}=101$. Then the assembly information, correctly stored, would appear as shown in Figure 5.



① = Element Global Number

i = Global DOF Numbers



Local Numbering
Conventions for DOF

FIGURE 4

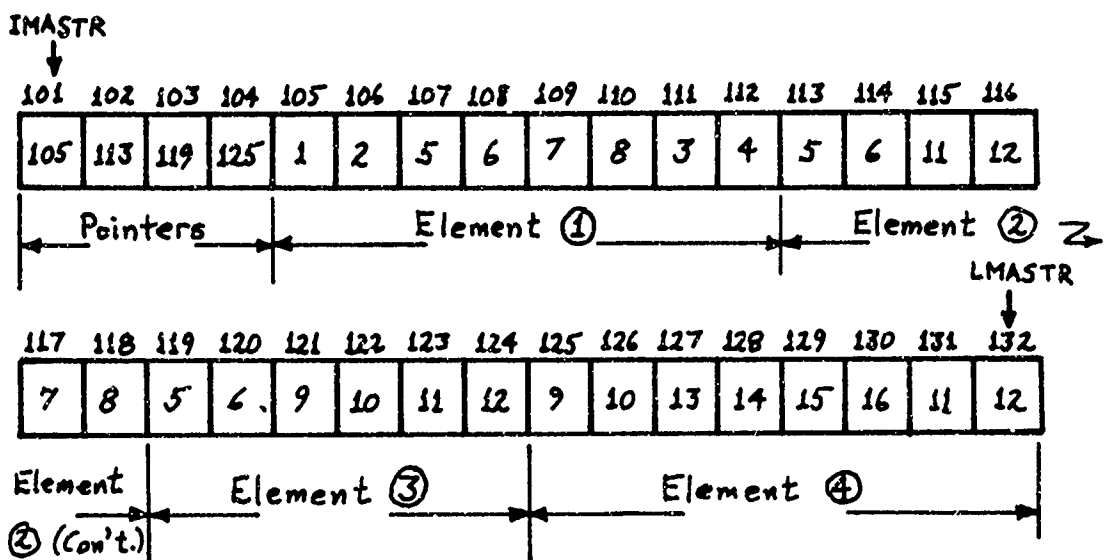


FIGURE 5

The following crude set of FØRTRAN instructions might be used to store the assembly information shown above:

```
      NEXT=IMASTR+4
      INTGR(IMASTR)=NEXT    (establishes pointer for 1st element)
      INTGR(NEXT)=1
      INTGR(NEXT+1)=2
      .
      .
      INTGR(NEXT+7)=4
      NEXT=NEXT+8
      INTGR(IMASTR+1)=NEXT (establishes pointer for 2nd element)
      INTGR(NEXT)=5
      INTGR(NEXT+1)=6
      .
      .
      INTGR(NEXT+5)=8
      NEXT=NEXT+6
      INTGR(IMASTR+2)=NEXT (establishes pointer for 3rd element)
      .
```

The use of pointers in the master list allows assemblies involving as many different types of elements with different total numbers of degrees of freedom as desired. Also the use of DOF numbers rather than node numbers permits assembly of elements having different numbers of displacements at various nodes without requiring any special programming or conventions. The user may choose any element and displacement global numbering schemes and any local displacement numbering conventions he desires. The only restrictions are that:

1. There must be no gaps in the master assembly list.
2. The array must be filled or, if excess storage exists, a zero must be stored after the last displacement number.
3. All element numbers and degree of freedom numbers must be positive.
4. The lowest element number and the lowest global degree of freedom number must each be unity.

2.2.5 Force/Displacement Vector

This is a floating point block which contains all of the force and displacement information required for analysis of a structure. Entries in the force/displacement vector are referred to by:

REAL(I) where $IQ \leq I \leq LQ$

The Nth entry in the vector is the force or displacement associated with the Nth global degree of freedom.

Various types of information are overlayed in the force/displacement vector. First, if the structure being analyzed is loaded by continuum body forces (e.g., gravity) or is in a thermal environment, the resulting element equivalent nodal forces must be assembled along with the element stiffness matrices. FEABL subroutine ASEMBL uses the force/displacement vector as a storage area for this purpose. Second, the user must introduce his global concentrated nodal forces and prescribed displacements into the force/displacement vector. Finally, the FEABL solution subprograms store the displacement solution in the force/displacement vector by overwriting the prescribed quantities.

The following simple algorithm enables the user to communicate with the Nth global degree of freedom:

```
NN=IQ+N-1
REAL(NN)=...
or
...=f(REAL(NN))
```

For example, suppose the structure in Figure 4 is to be given the boundary conditions shown in Figure 6. Global degrees of freedom 4 and 8 have concentrated forces A and B applied, respectively, while degrees 12 and 15 have displacements C and D prescribed, respectively. Displacements, 1, 2, 5, 6, 13, and 14 are prescribed to be zero. The following set of FORTRAN instructions specify these boundary conditions:

(a) Establishment of the constraint vector

```

INTGR(ICØN)=1
INTGR(ICØN+1)=2
INTGR(ICØN+2)=5
INTGR(ICØN+3)=6
INTGR(ICØN+4)=12
INTGR(ICØN+5)=13
INTGR(ICØN+6)=14
INTGR(ICØN+7)=15

```

(b) Input of the prescribed displacements:

```

REAL(IQ)=0.
REAL(IQ+1)=0.
REAL(IQ+4)=0.
:
REAL(IQ+13)=0.
REAL(IQ+11)=C
REAL(IQ+14)=D

```

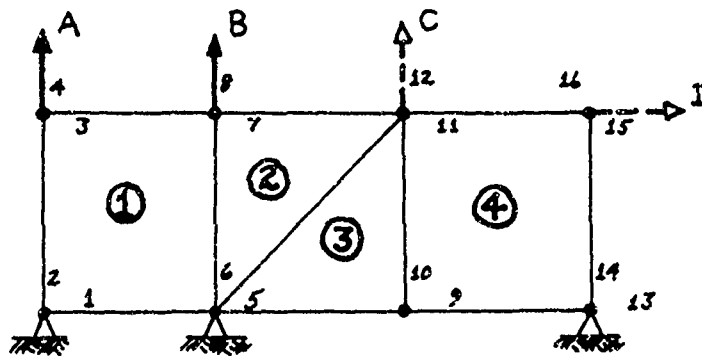
(c) Input of the prescribed forces:

```

REAL(IQ+3)=REAL(IQ+3)+A
REAL(IQ+7)=REAL(IQ+7)+B

```

The instructions (b) and (c) have been written assuming that element equivalent nodal forces have been assembled into the force/displacement vector. Thus, all degrees at which displacements are prescribed must be set to their correct values, while nonzero concentrated global forces must be added to the pre-existing assembled element forces.



A, B are global concentrated forces
C, D are global prescribed displacements

FIGURE 6

2.2.6 Master Stiffness Matrix

This floating point block contains the essential entries of the master stiffness matrix, stored one row after the other. Entries in the master stiffness matrix are referred to by:

REAL(I) where $IK \leq I \leq LK$

Normally, the user is not required to communicate directly with the master stiffness matrix, but an understanding of its detailed organization will be helpful in debugging.

Since a stiffness matrix is always symmetric, only its lower triangle need be kept in storage. Thus, the general organization of the master stiffness array in the /DATA/ vector may be represented by the diagram shown in Figure 7.

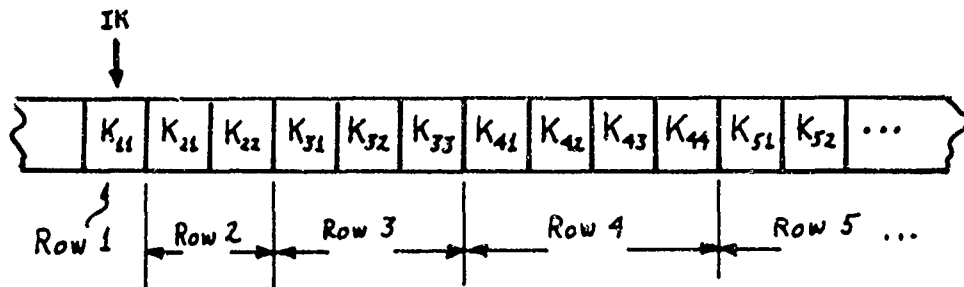


FIGURE 7

Significant additional savings in storage may be realized for problems with many degrees of freedom by taking advantage of the fact that a master stiffness matrix is normally banded and sparsely populated. The boundary line of the shaded area in Figure 8 represents the leading non-zero entry locations in a hypothetical stiffness matrix. FEABL subprograms which operate on the master stiffness matrix incorporate logic instructions which cause the operation to be skipped if the entry K_{ij} lies in the unshaded area of Figure 8. Thus, the leading zero entries for each row are not stored, and this is where the address count vector and the LN2 vector come into play. The actual organization of the master stiffness array consists of a sequence of

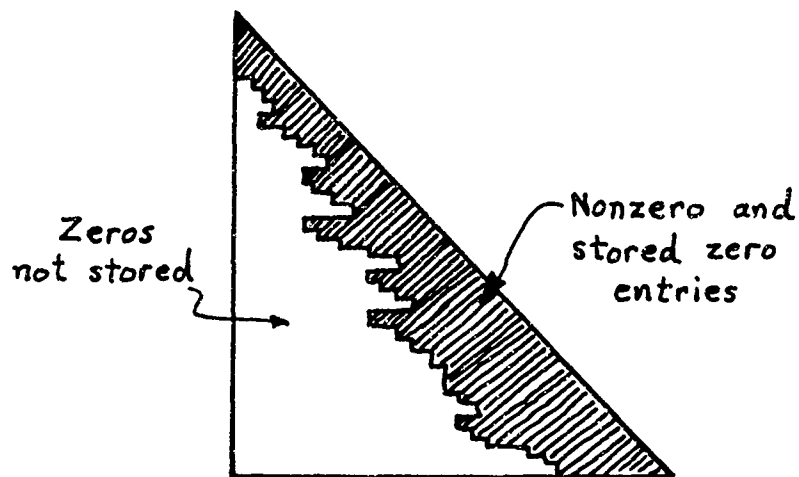


FIGURE 8

variable-length sections for the rows of the stiffness matrix. Figure 9 illustrates a sample sequence.

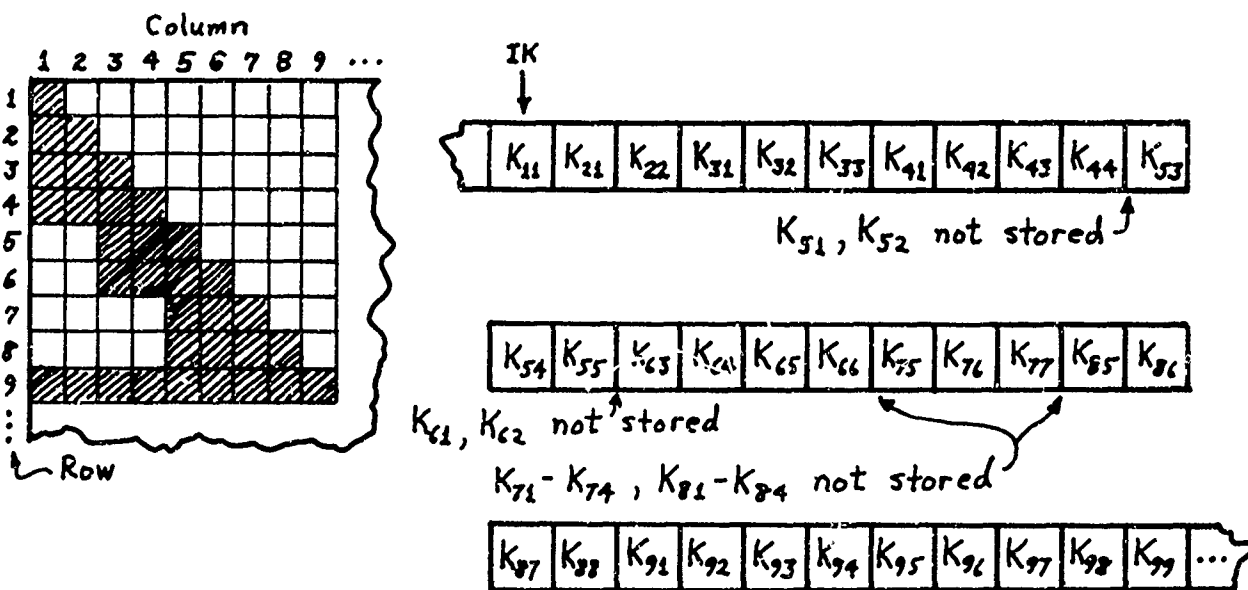


FIGURE 9

The algorithms for operating on the master stiffness matrix, based on the contents of the address count and LN_Z vectors are quite simple:

```

      (For an operation on  $K_{ij}$ : row I, column J,  $J \leq I$ )
      .
      .
      INDEX=ILNZ+I-1
      IF (J .LT. INTGR(INDEX)) GO TO 5
      KADR=IKOUNT+I-1
      KADR=INTGR(KADR)+J
      (To define address of  $K_{ij}$  in master stiffness array)
      (Operate using REAL(KADR) for  $K_{ij}$ )
      .
5     (Skip operation if  $J < LN_Z$  column number for the row)

```

2.3 Modification of the FEABL Data Storage System

The data storage system presented above has been designed for "production" computing. It has been assumed implicitly that the user will be conducting a great number of studies involving similar stress analysis problems and employing nearly the same number of degrees of freedom. Thus, the /DATA/ vector need be DIMENSIONed only once, after which production object decks of the FEABL software may be made.

However, the FEABL software may also be placed in on-line storage in a form which will handle problems of widely varying size, with only minor modifications. These modifications are as follows:

1. Delete the COMMON /DATA/ REAL declaration from all programs and subprograms.
2. Delete the EQUIVALENCE (REAL(1), INTGR(1)) declaration from all subprograms. (Retain this declaration in MAIN.)
3. Use the standard declaration:
 DIMENSION REAL(2), INTGR(2)
 in all FEABL subroutines. (DIMENSION the /DATA/ vector properly in MAIN.)
4. Add the array names REAL and INTGR as arguments of

all FEABL subroutines, e.g.:

```
SUBROUTINE ØRK(LENGTH, REAL, INTGR)
SUBROUTINE FACTPD(REAL, INTGR)
```

When the on-line version of FEABL is used, only the DIMENSION declaration for the /DATA/ vector in the user's MAIN program need be changed to perform analyses requiring different amounts of data storage.

SECTION 3

FEABL PROCESS CONTROL

3.1 General

For present purposes the finite element analysis of a structure will be divided into eight programming stages:

1. Establishment of input/output device codes, problem size, and address index parameters.
2. Input of the master assembly list and organization of the master stiffness matrix into corresponding segments.
3. Generation and assembly of element stiffnesses (and nodal equivalent forces, if any) in global coordinates.
4. Application of rotation transformations to the master stiffness matrix (and assembled nodal equivalent forces) at any nodes at which the boundary conditions are to be given in special coordinate systems.
5. Input of the prescribed quantities, i.e., global numbers at which displacements are to be prescribed, values of prescribed displacements, and accumulation of values of any nonzero global concentrated forces. Application of constraints to the master stiffness matrix and force/displacement vector.
6. Solution for the master displacement vector.
7. Application of inverse rotation transformations to the master displacement vector, at any nodes where a rotation was applied in stage 4, to produce a master displacement vector entirely in the global coordinate system.
8. Extraction of element displacement vectors from the global vector; calculation of element stresses from the element displacement vector.

Specific FEABL subprograms are associated with each of the above stages, according to the following table:

Stage	FEABL Subroutines	Stage	FEABL Subroutines
1	SETUP	5	BCØN
2	ØRK	6	FACTPD/FACTSD, SIMULQ
3	ASEMBL	7	RØTATE
4	RØTATE	8	XTRACT

Figure 10 illustrates the standard FEABL process sequence in terms of the eight stages described above.

Analysts who are just beginning to work with FEABL are advised to observe strictly the standard process sequence outlined above. This sequence will be described in full detail in Section 4. More experienced analysts may find it convenient to depart occasionally from the standard process in order to reduce program length.

3.2 Description of the FEABL Software

The FEABL package consists of the eight subroutines listed in the table in Subsection 3.1. Each subroutine is associated with a particular stage in the standard process sequence (except RØTATE, which is associated with both stages 4 and 7). All FEABL subprograms are ready to use for in-core finite element analysis, once the DIMENSION declaration for the /DATA/ vector has been inserted (see Subsections 2.2 and 2.3). The following subsections demonstrate how each of the FEABL subroutines is called, define the subroutine arguments, and outline briefly what the subroutine does. A summary table of the CØMMØN area information required by each FEABL subroutine is given in Appendix A. Listings appear in Appendix C.

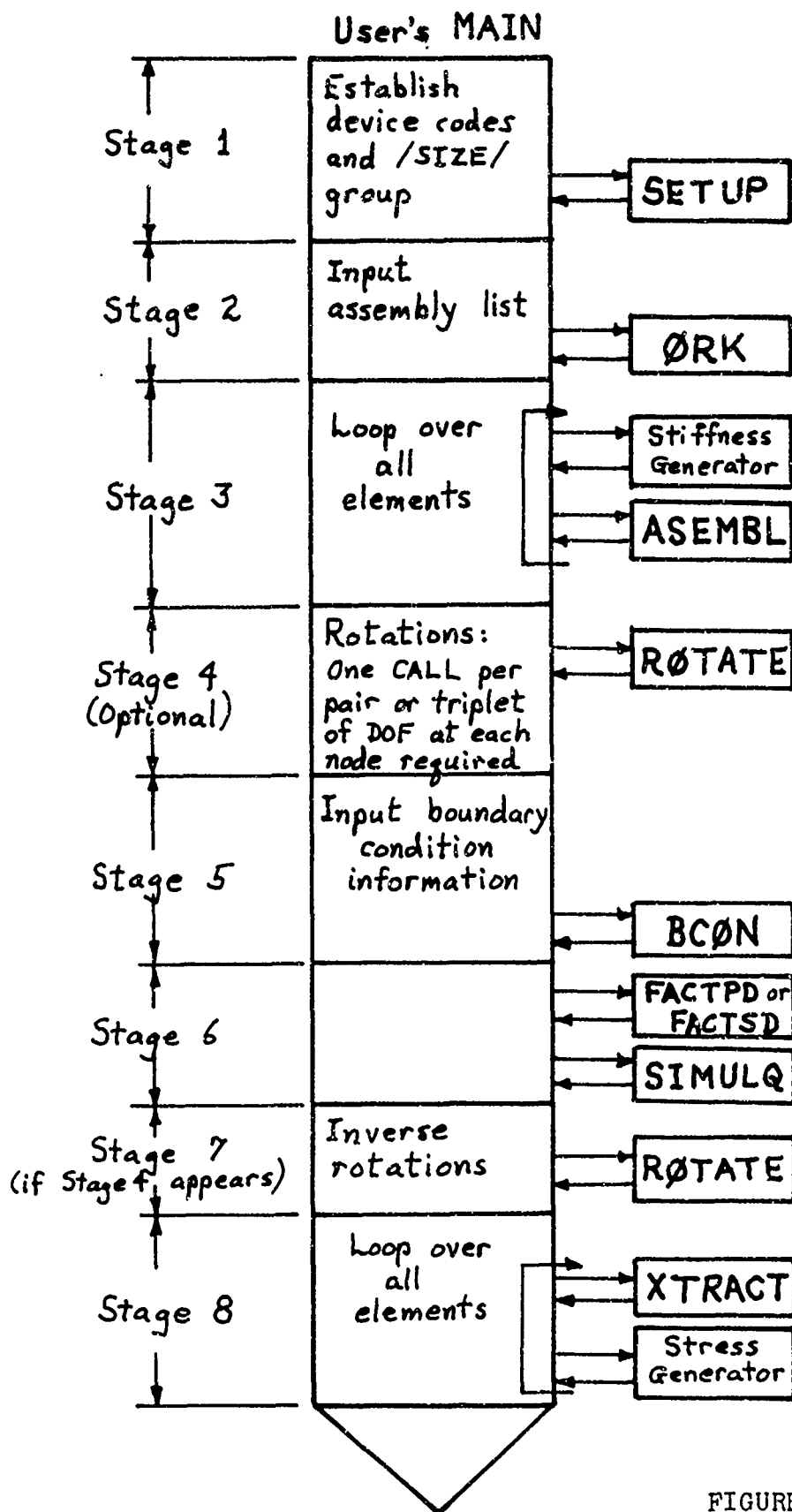


FIGURE 10

3.2.1 Housekeeping Setup Subroutine for /DATA/ Vector (SETUP)

CALL SETUP(LENGTH, NCØN, MASTRL)

LENGTH - A scalar integer numerically equal to the dimension which the user has assigned to the /DATA/ vector.

NCØN - A scalar integer greater than or equal to the total number of degrees of freedom, in the assembled structure, at which displacements are to be prescribed.

MASTRL - A scalar integer greater than or equal to the total number of words required for the master assembly list.

Based on NCØN, MASTRL and the total number of degrees of freedom in the entire assembled structure (NDT, in the /SIZE/ group), subroutine SETUP organizes the /DATA/ vector by calculating the address index parameters in the /BEGIN/ and /END/ groups, except for the index LK which defines the end of the block reserved for the master stiffness matrix. SETUP uses the argument LENGTH to test whether the user's /DATA/ vector has at least enough storage available to accommodate the first four data blocks (constraint vector, address count vector, LNZ vector and master assembly list). If the /DATA/ vector is too short, SETUP estimates the total length required for all six data blocks, based on a reasonable population density for the lower triangle of the master stiffness matrix, prints the estimate and aborts the run. If the first four blocks can be accommodated, the constraint vector is filled with zeros and control is returned to MAIN.

3.2.2 Subroutine for Detailed Organization of the Master Stiffness Matrix Block (ØRK)

CALL ORK(LENGTH)

LENGTH - A scalar integer numerically equal to the dimension which the user has assigned to the /DATA/ vector.

Subroutine ØRK produces, in essence, a map for the master stiffness matrix like that of Figure 9, using the information contained in the master assembly list to calculate the correct values of the entries in the LNZ vector. This is accomplished by first setting each LNZ column number equal to its row number (diagonal matrix), and then examining the assembly information element by element to re-set the LNZ column numbers, according to the following algorithm:

1. The smallest global number N for the element is found.
2. The element's global numbers are then treated as row numbers. If the LNZ column number corresponding to a row (global number) is greater than N, its value is re-set to N.

Once the LNZ vector has been established, subroutine ØRK uses an accumulation process to calculate the absolute address of the diagonal entry of each row. By convention, the diagonal is the only entry stored for the first row and is therefore located by the address index parameter IK:

INTGR(IKØUNT)=IK

Subsequent entries are located by the algorithm:

INTGR(IKØUNT+M-1)=INTGR(IKØUNT+M-2)+M+1-INTGR(ILNZ+M-1)
(Diag Addr for Mth row)=(Diag Addr for row M-1)+(Total no. of nonzero entries in Mth row)

Conveniently, the address of the diagonal for the last row in the master stiffness matrix is also the correct value of LK. Subroutine ØRK now tests the /DATA/ vector by means of the argument LENGTH. If the /DATA/ vector is too short for the problem data, an exact calculation of its required length is

output and the run is aborted. If sufficient storage is available, the stiffness matrix map is output and the address count vector is modified by:

$$\text{INTGR}(\text{IKØUNT}+M-1)=\text{INTGR}(\text{IKØUNT}+M-1)-M; 1 \leq M \leq \text{NDT}$$

for all rows, M. This saves repeated subtraction of the row number in later subprograms. The correct algorithm for locating K_{MN} in the /DATA/ vector is now:

$$\text{KADR}=\text{INTGR}(\text{IKØUNT}+M-1)+N; M > N$$

K_{MN} is assigned to $\text{REAL}(\text{KADR})$

Since the next stage of the analysis will involve the accumulation of data in the master stiffness matrix (and perhaps in the force/displacement vector), ØRK's last action before returning control to MAIN is to fill these two data blocks with floating point zeros.

3.2.3 Element Assembly Subroutine (ASEMBL)

CALL ASEMBL(LNUM, NDE, ELK, ELQ)

LNUM - A positive scalar integer = user's global element number

NDE - A scalar integer = total number of degrees of freedom possessed by the element which is about to be assembled

ELK - A floating point, two-dimensional square array which contains the stiffness matrix of the element about to be assembled.

ELQ - A floating point vector which contains the equivalent nodal forces for the element about to be assembled, or which contains floating point zeros if there are no equivalent nodal forces.

Since ELK and ELQ are variably DIMENSIONed in this subprogram, elements having different numbers of degrees of freedom can be handled automatically. However, these arguments must be DIMENSIONed explicitly in the user's MAIN program. For example, suppose a structure is to be analyzed in plane stress with a

combination of 3-node triangle elements (6 degrees of freedom) and 4-node rectangle elements (8 degrees of freedom). Then the declaration:

```
DIMENSION TRIK(6,6), TRIQ(6), RECK(8,8), RECQ(8)
```

might appear in MAIN. A triangle element would be assembled by the instruction:

```
CALL ASEMBL(LNUM, 6, TRIK, TRIQ)
```

while a rectangle element would be assembled by:

```
CALL ASEMBL (LNUM, 8, RECK, RECQ)
```

Subroutine ASEMBL can handle elements having as many as one hundred degrees of freedom (not a serious restriction). If the assembly of larger elements is attempted, ASEMBL will abort the run and tell the user to change the DIMENSION of one of its internal parameters.

ASEMBL examines the section of the master assembly list belonging to element number LNUM and records the values of the global displacement numbers in an internal vector called MNUM. Then, each entry K_{ij} of the lower triangle of the element stiffness matrix ($i \geq j$), is accumulated into its proper place in the master stiffness matrix, according to the algorithm:

```
I=MNUM(i)
```

```
J=MNUM(j)
```

```
 $K_{ij} \rightarrow K_{IJ}$  for  $I \geq J$ 
```

```
 $K_{ij} \rightarrow K_{JI}$  for  $I < J$ 
```

(See Subsection 3.2.1 for address algorithm for K_{IJ} .) ASEMBL does not use the upper triangle ($i < j$) of the element stiffness matrix; the user may omit calculating these entries to save time. The vector ELQ of element equivalent nodal forces is accumulated in the same manner into the proper locations in the master force vector (fifth block of the /DATA/ vector).

3.2.4 Rotation Transformation Subroutine (RØTATE)

CALL RØTATE(NØDE, IRØW, JRØW, KRØW, ZANGLE, YANGLE, XANGLE)

NØDE - A scalar integer "convenience" number for the user.

May be positive or negative but not zero. (See explanation below.)

IRØW } Three scalar integers equal to the global numbers
JRØW } of the degrees of freedom at the node at which
KRØW } the rotation is to be performed.

ZANGLE } Floating point values of the three Euler rotation
YANGLE } angles in degree measure.
XANGLE }

The argument NØDE does not enter directly into the transformation calculations, but is printed out in the heading of the information supplied by RØTATE. Normally, the user assigns either his node numbers or the series 1, 2, 3,... to a set of rotations. A positive value of NØDE causes the full subprogram to execute, and is used for stage 4 in the process sequence. In stage 7, only the solution displacement vector requires transformation; a negative value of NØDE will cause execution of the now unwanted transformation of the stiffness matrix to be skipped.

The global numbers of the degrees of freedom at the node being rotated may be contiguous (e.g., 13, 14, 15) or separated (e.g., 5, 10, 15) depending upon the global numbering scheme the user has adopted. Either form is acceptable to this subprogram. If only two degrees of freedom are to participate in the rotation (as, for example, in plane stress problems), an integer zero must be specified for the third global number. If six degrees of freedom are to participate (e.g., shell elements), two separate rotations are required, i.e., one CALL for the translational and one for the rotational degrees of freedom. Errors in the global number arguments (e.g., repeating a number or too many zeros) will cause an abort.

The three Euler angle arguments must be specified according to the conventions illustrated in Figure 11. Let XYZ be the user's global cartesian axis system, with respect to which the element stiffness and equivalent nodal forces have been generated and assembled. Let $\tilde{XYZ}$ be a local coordinate system with respect to which the displacement constraints are to be specified.

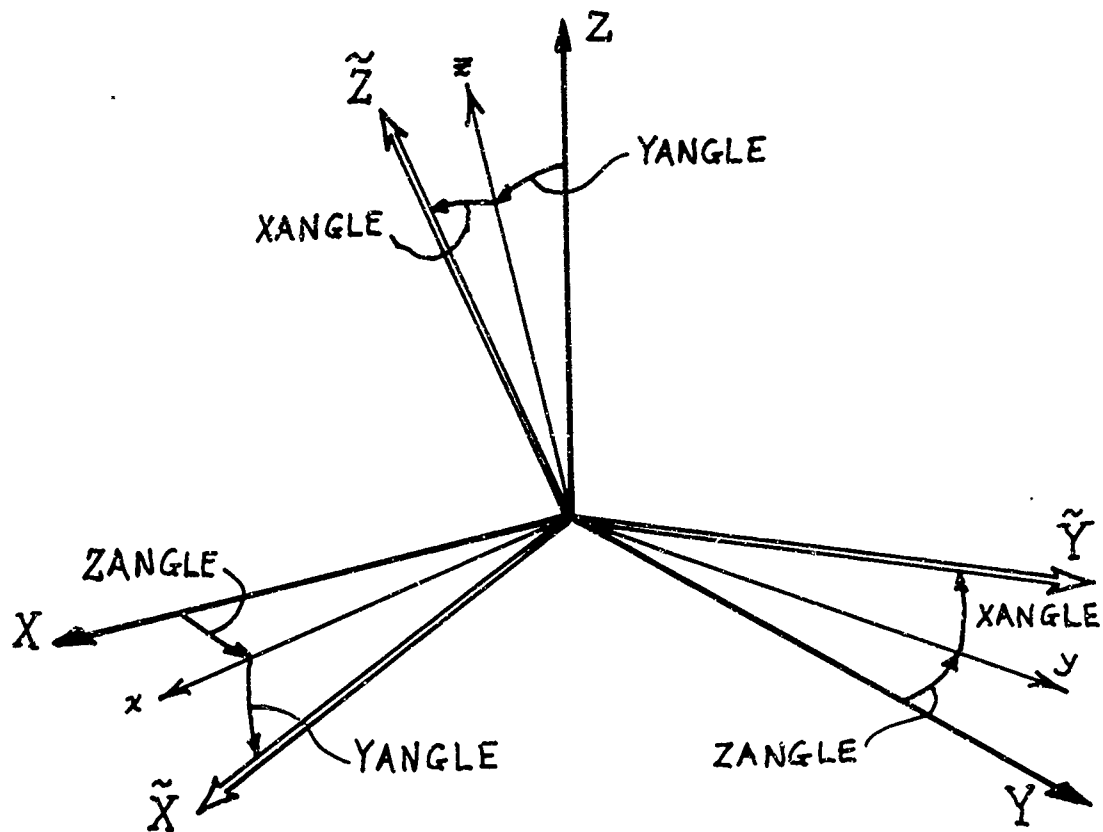


FIGURE 11

The following conventions have been adopted in subroutine ROTATE:

1. Axes X, Y are first rotated through angle ZANGLE about axis Z to the intermediate positions x, y.
2. Axes x, Z are then rotated through angle YANGLE about axis y, x to the final position $\tilde{X}$ and Z to an intermediate position z.
3. Axes y, z are then rotated through angle XANGLE about axis $\tilde{X}$ to their final positions $\tilde{Y}$, $\tilde{Z}$.
4. Positive angles obey the "right hand rule" of vector analysis.

Subroutine ROTATE forms the matrix of direction cosines:

$$\mathbf{D} = \begin{pmatrix} \cos(\tilde{X}, X) & \cos(\tilde{X}, Y) & \cos(\tilde{X}, Z) \\ \cos(\tilde{Y}, X) & \cos(\tilde{Y}, Y) & \cos(\tilde{Y}, Z) \\ \cos(\tilde{Z}, X) & \cos(\tilde{Z}, Y) & \cos(\tilde{Z}, Z) \end{pmatrix}$$

from the values of the Euler angles supplied in the argument list. Then, if argument NØDE is positive, ROTATE applies the following transformations to portions of the master stiffness matrix $\mathbf{K}$ where i, j, k are the global numbers specified by the user:

$$\mathbf{D} \begin{pmatrix} K_{ii} & K_{ij} & K_{ik} \\ K_{ji} & K_{jj} & K_{jk} \\ K_{ki} & K_{kj} & K_{kk} \end{pmatrix} \mathbf{D}^T$$

$$\mathbf{D} \begin{pmatrix} K_{i1} & K_{i2} \dots K_{i,i-1} & K_{i,i+1} \dots K_{i,j-1} & K_{i,j+1} \dots K_{i,k-1} & K_{i,k+1} \dots \\ K_{j1} & K_{j2} \dots K_{j,i-1} \dots \\ K_{k1} & K_{k2} \dots \end{pmatrix}$$

$$\begin{pmatrix}
 K_{1i} & K_{1j} & K_{1k} \\
 K_{2i} & K_{2j} & K_{2k} \\
 \vdots & \vdots & \vdots \\
 K_{i-1,i} & K_{i-1,j} & \\
 \vdots & \vdots & \\
 K_{i+1,i} & & \\
 \vdots & & \\
 K_{j-1,i} & & \\
 K_{j+1,i} & & \\
 \vdots & & \\
 K_{k-1,i} & & \\
 K_{k+1,i} & & \\
 \vdots & &
 \end{pmatrix} \mathbf{D}^T$$

Finally, the corresponding entries of the force/displacement vector are transformed according to:

$$\mathbf{D} \begin{Bmatrix} Q_i \\ Q_j \\ Q_k \end{Bmatrix}$$

A few examples will illustrate the proper use of subroutine RØTATE. First, suppose that global degrees of freedom 1 and 2 are to be rotated by 45 degrees at node 1 in a plane elasticity problem. The correct instruction is:

CALL RØTATE(1, 1, 2, 0, 45., 0., 0.)

To return the force/displacement vector to the user's global axis system after the displacement solution has been obtained:

CALL RØTATE(-1, 1, 2, 0, -45., 0., 0.)

Note that only the first angular argument ZANGLE is used in two-dimensional problems. In a three-dimensional problem, to rotate

degrees 4, 5, 6 at node 2 through angles of 15, 30 and 45 degrees about the Z, y and $\tilde{X}$ axes:

```
CALL ROTATE(2, 4, 5, 6, 15., 30., 45.)
```

and after the displacement solution has been obtained:

```
CALL ROTATE(-2, 4, 5, 6, -45., -30., -15.)
```

Note that not only the signs, but also the order of application of the angles is reversed. However, some care is required in three dimensions. Suppose degrees 7, 8, 9 at node 3 were rotated only about axes Z and y:

```
CALL ROTATE(3, 7, 8, 9, 20., 40., 0.)
```

Then to reverse the rotation after obtaining the displacement solution:

```
CALL ROTATE(-3, 7, 8, 9, -40., -20., 0.)
```

Note that the unused XANGLE does not participate in the reversal. The instruction:

```
CALL ROTATE(-3, 7, 8, 9, 0., -40., -20.)
```

would be incorrect.

3.2.5 Boundary Constraint Subroutine (BCØN)

```
CALL BCØN
```

Let u , Q represent respectively the global* displacement vector and the global force vector. The result of the first four program stages has been to supply a right-hand side and the stiffness coefficients K_{ij} for the force-displacement relations:

$$Ku = Q$$

However, the value of u is known at some degrees of freedom, with the corresponding Q unknown, while the value of Q is known

*The term "global" should be taken in a more general sense here. It refers to the final set of coordinate systems $\tilde{XYZ}$ in which the user will prescribe his boundary conditions.

at the other degrees.

Conceptually, the set of all degrees of freedom in the structure may be divided into two subsets: Those at which forces are prescribed (F) and those at which displacements are prescribed (D). When the user specifies the values of the prescribed quantities, Q becomes a "force/displacement" vector in fact:

$$Q = \begin{Bmatrix} \hat{Q}_F \\ \hat{u}_D \end{Bmatrix} \quad (\text{"^"} \text{ means a prescribed quantity})$$

and:

$$\begin{aligned} u_F &= \text{unknown displacements} \\ Q_D &= \text{unknown reaction forces} \end{aligned}$$

The force-displacement relations may be partitioned in a similar manner:

$$\begin{Bmatrix} K_{FF} & K_{FD} \\ K_{FD}^T & K_{DD} \end{Bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F \\ Q_D \end{Bmatrix}$$

Subroutine BCØN transforms the force-displacement relations from the above form to:

$$\begin{Bmatrix} K_{FF} & 0 \\ 0 & I \end{Bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F - K_{FD} \hat{u}_D \\ \hat{u}_D \end{Bmatrix}$$

With the above "right hand side" in the force/displacement vector, standard equation-solving techniques may be used to produce the solution displacement vector. Before making the transformation, BCØN arranges the entries of the constraint vector (Subsection 2.2.1)

in ascending order and checks for the presence of global numbers (positive integers). The run is aborted if no global numbers are found.

3.2.6 Stiffness Matrix Factoring Subroutine (FACTPD/FACTSD)

CALL FACTPD

or

CALL FACTSD

The force-displacement relations are solved by Choleski's direct method, which consists of two programming steps:

1. The master stiffness matrix is factored into a triple product.
2. The displacements are solved for sequentially, in three sub-steps.

Subroutine FACTPD/FACTSD accomplishes the first programming step. The master stiffness matrix **K** is factored into the form:

$$\mathbf{K} = \mathbf{L} \mathbf{D} \mathbf{L}^T$$

where **L** is a lower triangular matrix:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \dots \\ L_{21} & 1 & 0 & 0 \dots \\ L_{31} & L_{32} & 1 & 0 \dots \\ L_{41} & L_{42} & L_{43} & 1 \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

and **D** = $\begin{bmatrix} D_1 & & \\ & D_2 & \\ & & \dots \end{bmatrix}$ is a diagonal matrix. The factoring algorithms are:

$$\left. \begin{aligned} L_{mn} &= \frac{1}{D_n} \left[K_{mn} - \sum_{j=J(n)}^{n-1} (D_j L_{mj} L_{nj}) \right] ; n = J(n), \dots, m-1 \\ D_m &= K_{mm} - \sum_{j=J(m)}^{m-1} (D_j L_{mj}^2) \end{aligned} \right\} m = 1, 2, \dots, NDT$$

where:

$J(n)$ = Leading Nonzero Entry Column Number for Row n
 $J(m)$ = Leading Nonzero Entry Column Number for Row m
 $J(m,n)$ = $\max(J(n), J(m))$

The master stiffness matrix is destroyed and replaced by the entries of **L** and **D** as the factoring process is executed. The entries L_{mn} ($m > n$) and D_m are stored respectively at K_{mn} ($m > n$) and K_{mm} . The unit diagonal entries of **L** are not stored.

FACTPD/FACTSD tests the entries of **D** for nonsingularity and positive-definiteness as they are created. **K** is positive-definite if all $D_m > 0$. Rows for which $D_m < 0$ are reported. If any D_m is found to equal zero exactly, **K** is singular; the row in which the singularity was discovered is reported and the run is aborted.

The names FACTPD and FACTSD refer to different entry points in this subprogram. If **K** is supposed to be positive-definite (as in the case of a structure analyzed by compatible displacement elements), FACTPD should be called. If errors were made in the assembly or constraint stages of the program, they will appear now as $D_m < 0$ in one or more rows, and FACTPD will abort the run. If **K** is not necessarily positive-definite (as in the case of a structure analyzed by hybrid stress-displacement elements), FACTSD should be called. FACTSD continues execution even if there exist $D_m < 0$. If the FEABL software is to be converted to the on-line storage version (Section 2.3), be sure to make the array names REAL, INTGR arguments of both entry points.

FACTPD/FACTSD also makes a rough estimate of the conditioning of $\mathbf{K}$ by calculating the so-called rounding error parameter (see Ref. 2, pg. 81):

$$E = \min (|D_m| / |K_{mm}|)$$

E is a measure of how many significant figures of information have been lost in the diagonal entries, as a result of the factoring algorithm. The run is aborted if $E < 10^{-5}$. The user must keep in mind the fact that the rounding error parameter is an imperfect conditioning measure. If FACTPD/FACTSD reports that no significant figures have been lost, it does not necessarily follow that there is no error in the displacement solution. Advanced techniques for realistic solution error estimates are discussed in Section 5.

3.2.7 Subroutine for Solution of Simultaneous Equations (SIMULQ)

CALL SIMULQ(ENERGY)

ENERGY - A scalar floating point variable, the value of which is undefined when SIMULQ is called.

With the stiffness matrix in factored form $\mathbf{K} = \mathbf{L}\mathbf{D}\mathbf{L}^T$, let $\mathbf{P} = \mathbf{L}^T \mathbf{u}$ and $\mathbf{R} = \mathbf{D}\mathbf{P}$. Then:

$$\mathbf{L}\mathbf{R} = \mathbf{Q}$$

can be solved sequentially for $R_1, R_2, \dots, R_{NDT}$ via the algorithm:

$$R_1 = Q_1$$

$$R_m = Q_m - \sum_{j=J(m)}^{m-1} (L_{mj} R_j) ; m = 2, 3, \dots, NDT$$

Then:

$$P_m = R_m / D_m ; m = 1, 2, \dots, NDT$$

Finally, the displacement solution is obtained by solving $L^T u = P$ sequentially for u_{NDT} , u_{NDT-1} , ..., u_1 according to:

$$u_{NDT} = P_{NDT}$$

$$u_m = P_m - \sum_{j=m+1}^{NDT} (L_{jm} u_j) ; m = NDT-1, NDT-2, \dots, 1$$

As SIMULQ carries out these three sub-steps, the prescribed vector Q is first replaced by R , then R is replaced by P and finally P is replaced by u . Q and u are printed out by SIMULQ.

Since the approximate value of the strain energy in the structure is often useful to the analyst, this quantity is calculated by SIMULQ during execution of the solution steps. If the factored form of K is introduced into the strain energy expression, there results:

$$\text{Strain Energy} = \frac{1}{2} u^T K u = \frac{1}{2} (u^T L) D (L u) = \frac{1}{2} P^T D P$$

Thus, the straightforward algorithm:

$$\text{Strain Energy} = \frac{1}{2} \sum_{j=1}^{NDT} D_j (P_j)^2$$

can be used. The value of the strain energy thus calculated is printed out. At the end of execution of subroutine SIMULQ, the force/displacement vector (fifth block in the /DATA/ vector) contains the master displacement solution vector, and the strain energy value has been assigned to the argument ENERGY.

3.2.8 Subroutine for Extraction of Element Displacements from the Global Displacement Vector (XTRACT)

CALL XTRACT(LNUM, NDE, ELQ)

LNUM - A positive scalar integer equal to user's global element number

NDE - A scalar integer equal to the total number of degrees of freedom in the element

ELQ - A floating point vector at least NDE words long. The entries of ELQ are undefined when XTRACT is called.

If the analyst desires to calculate stress or strain distributions in his structure, he commonly uses transformations between stress or strain and nodal displacements:

$$\epsilon = B u_{el} \quad \sigma = E \epsilon$$

where ϵ, σ are respectively vectors of strain and stress components at selected points in the element domain, and where u_{el} is the vector of element nodal displacements, i.e., a subset of the global vector u . Subroutine XTRACT selects the correct subset u_{el} out of u , based upon the information contained in the user's master assembly list. The values of u_{el} are placed in the argument vector ELQ.

SECTION 4

A SAMPLE FEABL PROGRAM

In order to illustrate further the data location and process sequence interfacing, sample user programs will be developed for analysis of the truss structure shown in Figure 12. The structure consists of 16 bars and 18 degrees of freedom in the XY plane.

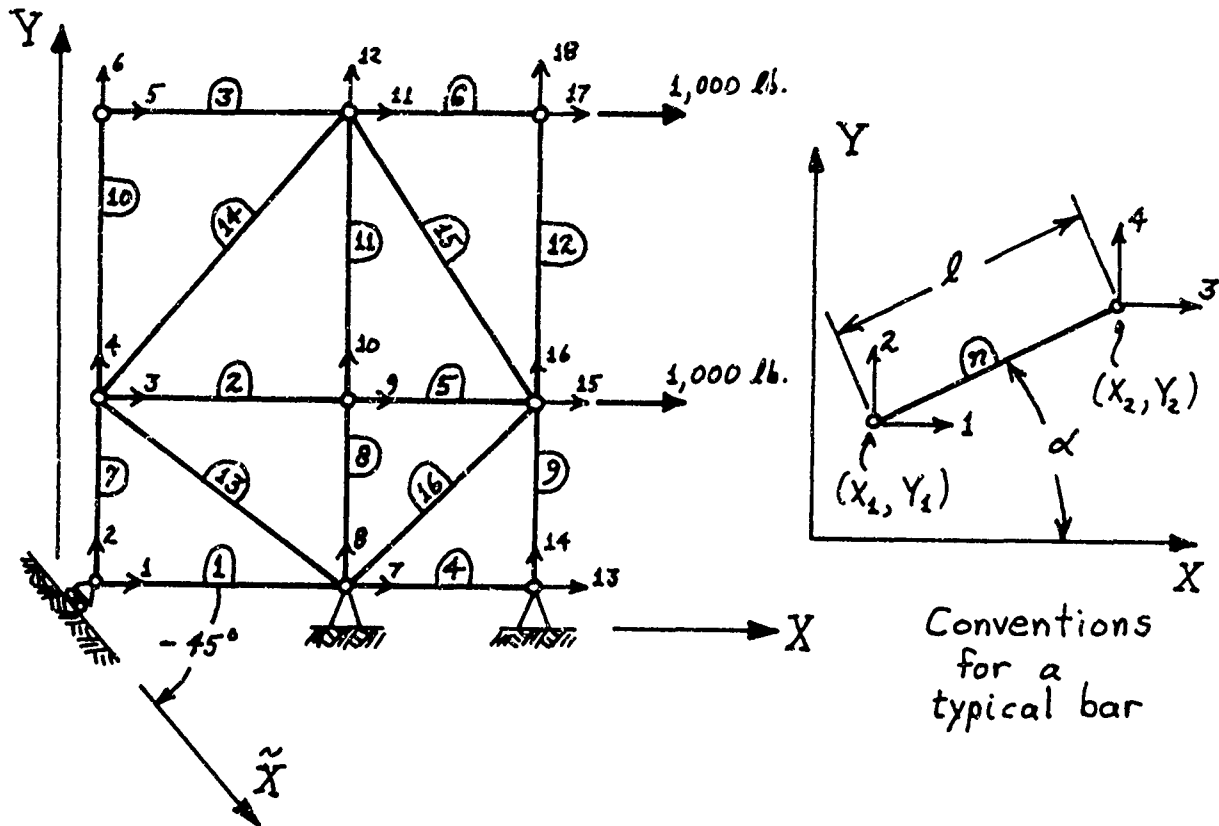


FIGURE 12

Local number conventions for the (n) th typical bar element are shown at the right. With these conventions, the stiffness matrix of the typical bar element is given by:

$$k_{\oplus} = \frac{EA}{l} \begin{bmatrix} \cos^2 \alpha & \sin^2 \alpha & \sin \alpha \cos \alpha & -\sin \alpha \cos \alpha \\ -\cos^2 \alpha & \cos^2 \alpha & -\sin \alpha \cos \alpha & \sin \alpha \cos \alpha \\ -\sin \alpha \cos \alpha & \sin \alpha \cos \alpha & \sin^2 \alpha & -\sin^2 \alpha \end{bmatrix} \quad (\text{Symmetric})$$

where:

$$l = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

$$\cos \alpha = \frac{1}{l} (X_2 - X_1)$$

$$\sin \alpha = \frac{1}{l} (Y_2 - Y_1)$$

and where E, A are the bar's modulus and cross section area. Also, once the displacements u_1, u_2, u_3, u_4 are known for the element, its elongation may be calculated as:

$$\delta = (u_3 - u_1) \cos \alpha + (u_4 - u_2) \sin \alpha$$

and the load in the bar is then given by:

$$P = EA \delta / l$$

where l and α are defined as above.

The user decides to read the properties and nodal coordinates for each element, each time they are required (a rather inefficient procedure). Reading is to be done by the generator subroutine, rather than in MAIN; however, the card reader device code will be established in MAIN. Therefore, the user programs his stiffness matrix and stress generator subroutines as follows:

Stiffness Matrix Generator

```
      SUBROUTINE BARK (ELK, ELQ)
      DIMENSION ELK (4, 4), ELQ (4)
      COMMON /IØ/ KR, KW, KP, KT1, KT2, KT3
C   INTERFACE WITH IØ CONTROL PARAMETERS IS OPTIONAL
91  FORMAT (6E10.3)
      READ (KR, 91) X1, Y1, X2, Y2, E, A
C   CALCULATE BAR LENGTH
      BARL=SQRT ((X2-X1)** 2+(Y2-Y1)** 2)
C   CALCULATE SINE AND COSINE
      S=(Y2-Y1)/BARL
      C=(X2-X1)/BARL
C   CALCULATE ENTRIES IN LOWER TRIANGLE OF ELK-ASEMBL DOES
C   NOT USE UPPER TRIANGLE
      ELK (1, 1)=E*A*C*C/BARL
      ELK (2, 1)=E*A*S*C/BARL
      .
      .
      etc.
      .
      ELK (4, 4)=E*A*S*S/BARL
C   ESTABLISH ZERO ELEMENT EQUIVALENT NODAL FORCES
      DO 10 I=1, 4
10   ELQ (I)=0.
      RETURN
      END
```

Stress Generator

```
      SUBROUTINE BARF (LNUM, ELQ)
      DIMENSION ELQ (4)
      COMMON /IØ/ KR, KW, KP, KT1, KT2, KT3
91  FORMAT (6E10.3)
92  FORMAT (21HOBAR FORCE IN BAR NØ., I4, 2H =, E10.3, 3H LB)
      READ (KR, 91) X1, Y1, X2, Y2, E, A
      BARL = SQRT ((X2-X1)** 2+(Y2-Y1)**2)
      S=(Y2-Y1)/BARL
      C=(X2-X1)/BARL
      FORCE=E*A*(C*(ELQ(3) - ELQ (1)) + S* (ELQ(4)-ELQ (2)))/BARL
      WRITE (KW, 92) LNUM, FORCE
      RETURN
      END
```

Subroutines BARK and BARF are ready to use in conjunction with the FEABL software. The user now begins the construction of his MAIN program, one stage at a time.

Stage 1: Program Heading (Data Location Interface), Device
Code and Problem Size Establishment

The user estimates that a 1000-word /DATA/ vector will be more than adequate for the problem. Required device codes are the card reader (5) and printer (6).

```
C  MAIN PROGRAM FOR SOLUTION OF TRUSS PROBLEM
    DIMENSION REAL(1000), INTGR(1000)
    DIMENSION ELK(4, 4), ELQ(4)
    COMMON /IØ/ KR, KW, KP, KT1, KT2, KT3
    COMMON /SIZE/ NET, NDT
    COMMON /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
    COMMON /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
    EQUIVALENCE (REAL(1), INTGR(1))
    KR=5
    KW=6
    NET=16
    NDT=18
    CALL SETUP (1000, 5, 80)
C  END OF STAGE 1
```

The user has called for space for five constraints: master displacements 2 (after a -45° rotation of 1 and 2), 7, 8, 13 and 14. The assembly list must allow room for 16 elements $\times$ (4 DOF plus 1 pointer per element) = 80 words. The first declaration:

```
DIMENSION REAL(1000), INTGR(1000)
```

is also duplicated and placed in each FEABL subroutine.

Stage 2: Assembly List Input and Organization of **K**

The user recognizes that the pointers will occupy locations IMASTR to IMASTR+15 (see Section 2.2.4). He chooses to write specific assignment instructions for each element:

```
INTGR(IMASTR)=IMASTR+16 (Pointer for 1st element)
INTGR(IMASTR+16)=1
INTGR(IMASTR+17)=2
INTGR(IMASTR+18)=7
INTGR(IMASTR+19)=8
:
:
```

Stage 1: Program Heading (Data Location Interface), Device Code and Problem Size Establishment

The user estimates that a 1000-word /DATA/ vector will be more than adequate for the problem. Required device codes are the card reader (5) and printer (6).

```
C  MAIN PROGRAM FOR SOLUTION OF TRUSS PROBLEM
    DIMENSION REAL(1000), INTGR(1000)
    DIMENSION ELK(4, 4), ELQ(4)
    COMMON /IØ/ KR, KW, KP, KT1, KT2, KT3
    COMMON /SIZE/ NET, NDT
    COMMON /BEGIN/ ICØN, IKØUNT, ILNZ, IMASTR, IQ, IK
    COMMON /END/ LCØN, LKØUNT, LLNZ, LMASTR, LQ, LK
    EQUIVALENCE (REAL(1), INTGR(1))
    KR=5
    KW=6
    NET=16
    NDT=18
    CALL SETUP (1000, 5, 80)
C  END OF STAGE 1
```

The user has called for space for five constraints: master displacements 2 (after a -45° rotation of 1 and 2), 7, 8, 13 and 14. The assembly list must allow room for 16 elements $\times$ (4 DOF plus 1 pointer per element) = 80 words. The first declaration:

```
DIMENSION REAL(1000), INTGR(1000)
```

is also duplicated and placed in each FEABL subroutine.

Stage 2: Assembly List Input and Organization of **K**

The user recognizes that the pointers will occupy locations IMASTR to IMASTR+15 (see Section 2.2.4). He chooses to write specific assignment instructions for each element:

```
INTGR(IMASTR)=IMASTR+16 (Pointer for 1st element)
INTGR(IMASTR+16)=1
INTGR(IMASTR+17)=2
INTGR(IMASTR+18)=7
INTGR(IMASTR+19)=8
:
:
```

```

:
:
INTGR(IMASTR+7)=IMASTR+44 (pointer for 8th element)
INTGR(IMASTR+44)=7
INTGR(IMASTR+45)=8
INTGR(IMASTR+46)=9
INTGR(IMASTR+47)=10
:
:
INTGR(IMASTR+15)=IMASTR+76 (pointer for 16th element)
INTGR(IMASTR+76)=7
INTGR(IMASTR+77)=8
INTGR(IMASTR+78)=15
INTGR(IMASTR+79)=16 (80th location in master assembly list)

```

At this point, it is advisable to dump the assembly list for debugging purposes if there is any doubt about its accuracy.

Dumping may be done by:

```

      WRITE (KW, 5) (INTGR(I), I=IMASTR, LMASTR)
5     FORMAT (1X, 10I10)

```

Finally:

```

      CALL ØRK(1000)
C     END ØF STAGE 2

```

Stage 3: Generation and Assembly of Element Properties

In this stage, the user merely invokes the appropriate subroutines in a loop.

```

      DØ 10 LNUM=1, NET
      CALL BARK (ELK, ELQ)
      CALL ASEMBL (LNUM, 4, ELK, ELQ)
10     CØNTINUE
C     END ØF STAGE 3

```

Stage 4: Rotation Transformations

In the present problem, only the lower left node (Figure 12) requires rotation. Master degrees of freedom number 1 and 2 must be rotated by -45 degrees.

```

      CALL ROTATE(1, 1, 2, 0, -45., 0., 0.)
                        ↑      ↑      ↑
                        Quantities for 3-D problems are not used
C     END ØF STAGE 4

```

Stage 5: Boundary Conditions

Displacements 2, 7, 8, 13, 14 are constrained, and the constraint vector starts in INTGR(1) (ICØN=1). Therefore:

```
INTGR(1)=2
INTGR(2)=7
INTGR(3)=8
INTGR(4)=13
INTGR(5)=14
```

The constrained displacements are all prescribed to be zero. Only the nonzero prescribed forces need be input; these are 1,000 lb. each at the 15th and 17th degrees of freedom. Therefore:

```
C  PRESCRIBED DISPLACEMENTS
    REAL(IQ+1)=0.
    REAL(IQ+6)=0.
    REAL(IQ+7)=0.
    REAL(IQ+12)=0.
    REAL(IQ+13)=0.
C  ACCUMULATE PRESCRIBED FORCES
    REAL(IQ+14)=REAL(IQ+14)+1000.
    REAL(IQ+16)=REAL(IQ+16)+1000.
    CALL BCØN
C  END ØF STAGE 5
```

Stage 6: Choleski Solution

```
    CALL FACTPD
    CALL SIMULQ(ENERGY)
C  END ØF STAGE 6
```

Stage 7: Inverse Rotation to Obtain Global Displacements in Global Cartesian Coordinates

A rotation of -45° was performed at node 1 before the boundary conditions were imposed. Since all element calculations are done with respect to the unrotated XY axis system, this rotation must be reversed before element stresses are calculated:

```
    CALL RØTATE(-1, 1, 2, 0, 45., 0., 0.)
C  END ØF STAGE 7
```

Stage 8: Calculation of Element Stresses

In this case, bar forces are to be calculated. Again, the user merely invokes the proper subroutines:

```

      DØ 20 LNUM=1, NET
      CALL XTRACT (LNUM, 4, ELQ)
      CALL BARF (LNUM, ELQ)
C    20 CØNTINUE
      END ØF PRØG
      STØP
      END

```

If the reader has grasped the material presented up to this point, he now has enough familiarity with FEABL to use it, albeit somewhat inefficiently. However, some additional degree of sophistication is desirable for the instructions which input problem data such as the assembly list. It is apparent that a straightforward set of instructions such as that given in the sample program is quite cumbersome, especially for problems involving more elements with more degrees of freedom per element. Improved programming techniques are discussed in the next section.

SECTION 5

ADVANCED TECHNIQUES WITH FEABL

5.1 Efficient Programming for Large Problems

When a structure is to be analyzed with a large number of elements and many degrees of freedom, there often occurs a definite trend toward regularity in the element set. The analyst should recognize two distinct forms of regularity, and he should be prepared to take advantage of each in writing his MAIN program.

First, consider the truss structure shown in Figure 13, with all bays having the same dimension vertically and horizontally.

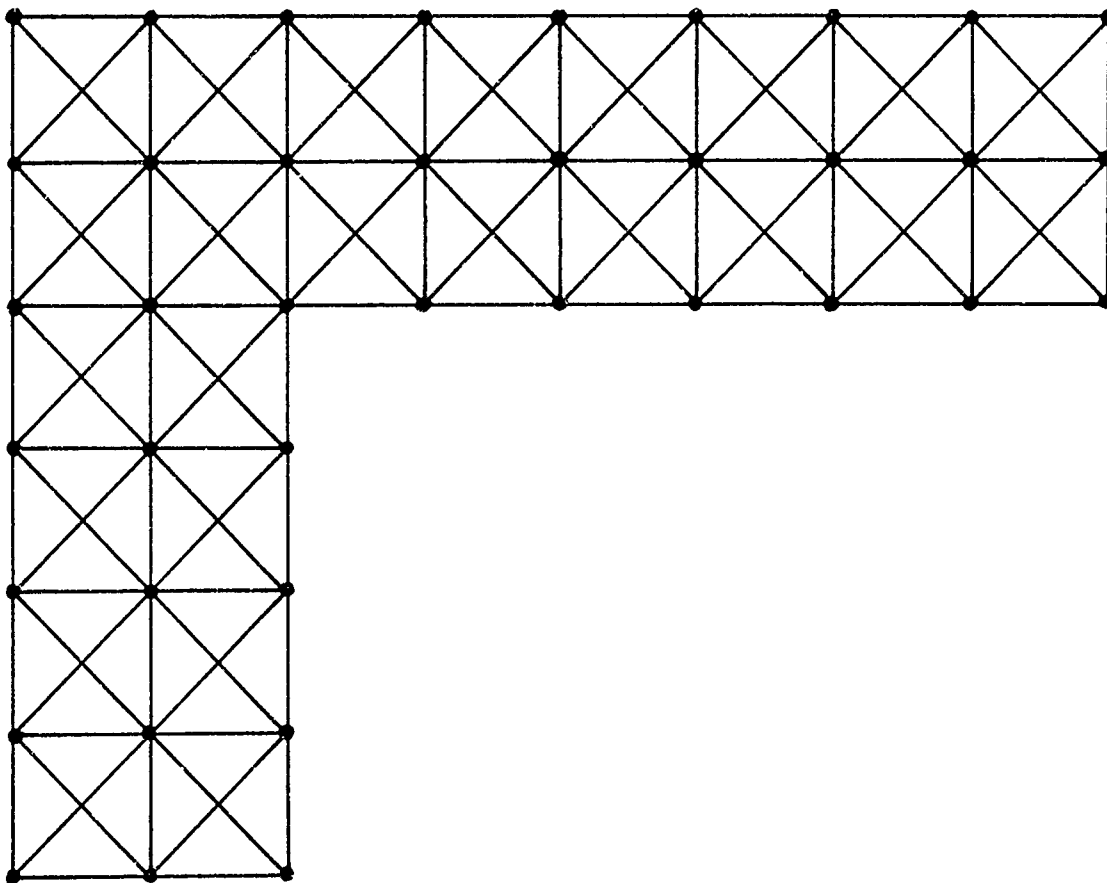


FIGURE 13

The bar elements may be said to possess a degree of regularity, in that there are only four different types of elements in the structure:

1. Horizontal bars
2. Vertical bars.
3. $+45^\circ$ diagonals.
4. -45° diagonals.

Since an element stiffness matrix depends upon nodal coordinates only through differences in the coordinate values (see beginning of Section 4), there will occur only four independent element stiffness matrices for the structure of Figure 13: one for each of the element types listed above. Thus, for example, the element stiffness matrix for a typical horizontal bar may be generated once and assembled repeatedly (32 times for the present case). The technique may be repeated again for the 30 vertical bars, 24 $+45^\circ$ diagonals and 24 -45° diagonals, achieving considerable savings in execution time.

The second form of regularity involves the way in which displacements and elements may be numbered. Often an element set is topologically equivalent to a rectangular or square array, even if it is not geometrically regular. For example, suppose the trapezoidal continuum shown in Figure 14(a) is to be analyzed in plane stress using 32 3-node triangle elements. For numbering purposes the element and displacement sets are topologically equivalent to the square net shown in Figure 14(b). It is then possible to generate the assembly list for the structure by a double DØ loop. Each element is "located" in the structure via an intersection of one of the element strings LX and one of the element strings LY. The element number and its master displacement numbers may be generated from the values of LX and LY. Taking advantage of numbering regularity enables the user to input large sections of the assembly list with relatively few FORTRAN instructions.

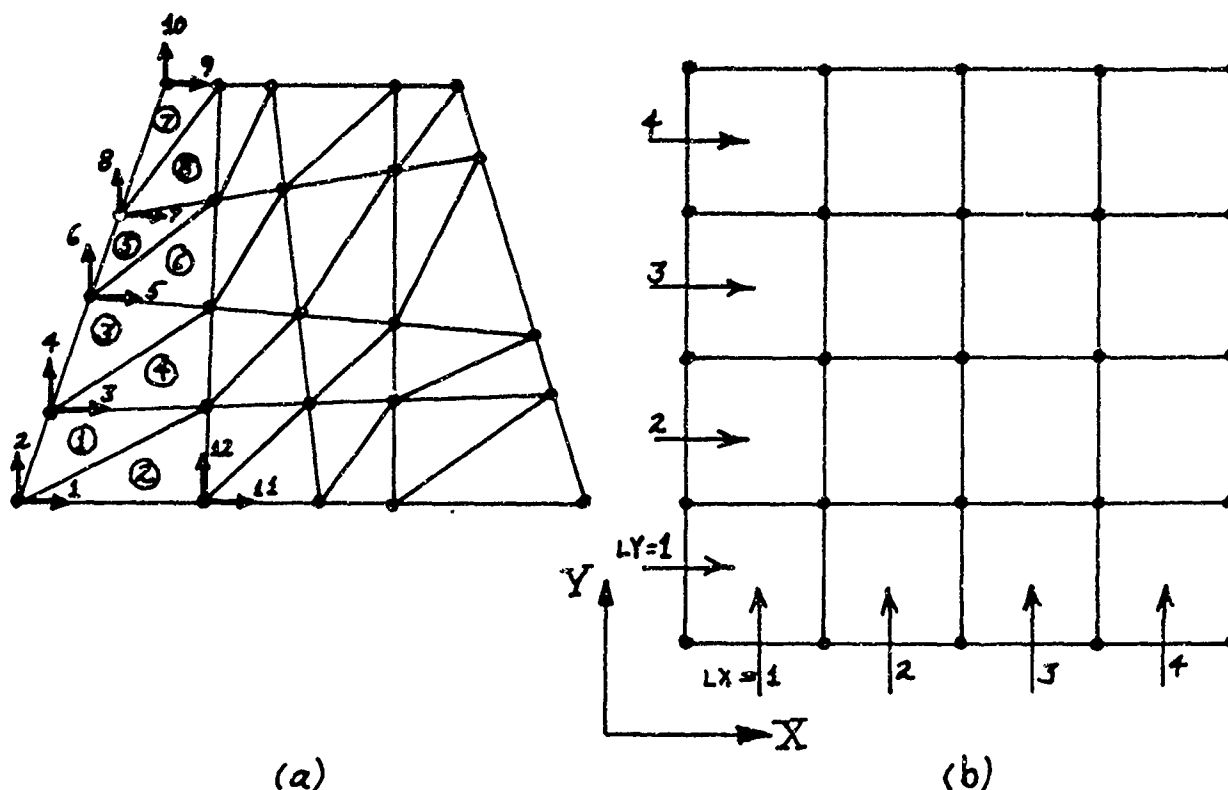


FIGURE 14

5.2 A Numbering Strategy for Planar Problems

Suppose the set of $M \times N$ rectangular plane stress elements shown in Figure 15 is to be assembled, where M , N may be quite large. If the numbering strategy shown in the figure is adopted, beginning with element 1 and displacements 1, 2 at the lower left corner and ending with element MN and displacements $2(M+1)(N+1)-1$, $2(M+1)(N+1)$ at the upper right corner, then each element number can be given as a function of its LY , LX string coordinates:

$$LNUM = LY + N*(LX-1) \text{ where } 1 \leq LY \leq N \text{ and } 1 \leq LX \leq M$$

In order to create similar functions for the element assembly list, a local numbering convention must be adopted. If the

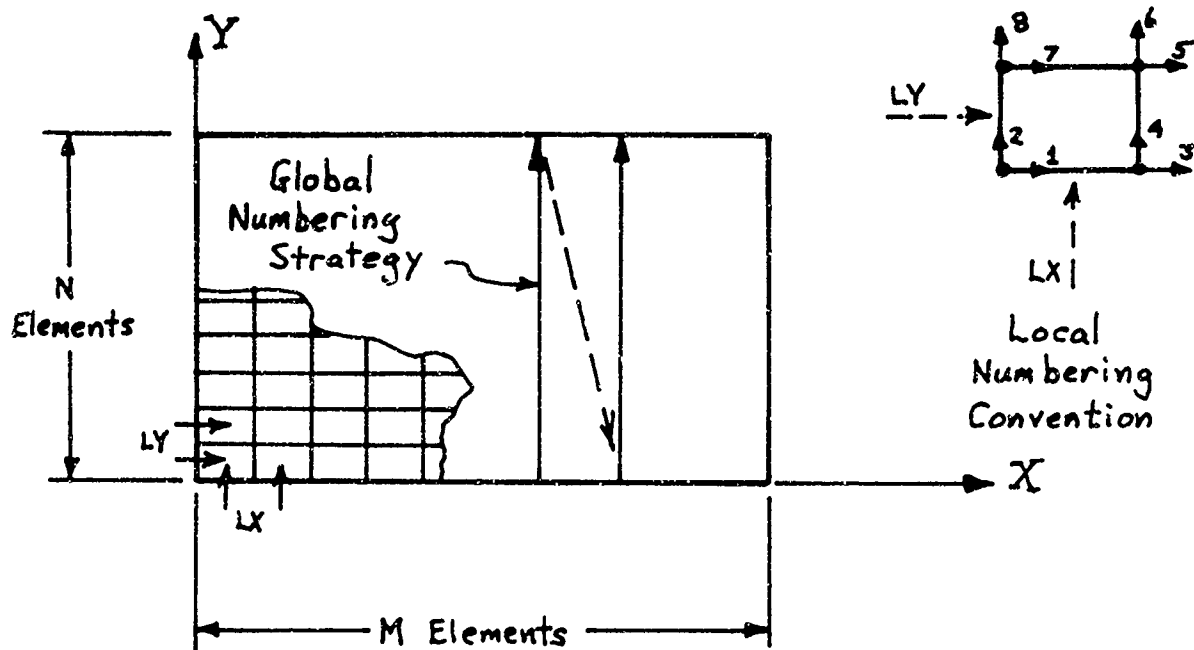


FIGURE 15

convention shown in Figure 15 is used, the global DOF numbers may be calculated as follows: Local displacements 1, 2 can be considered to lie on strings LX, LY; hence:

$$\begin{aligned} (\text{Local \#1}) &= 2*LY - 1 + 2*(N+1)*(LX - 1) \\ (\text{Local \#2}) &= (\text{Local \#1}) + 1 \end{aligned}$$

Local displacements 7, 8 follow directly from the master numbering scheme:

$$\begin{aligned} (\text{Local \#7}) &= (\text{Local \#2}) + 1 = (\text{Local \#1}) + 2 \\ (\text{Local \#8}) &= (\text{Local \#7}) + 1 = (\text{Local \#1}) + 3 \end{aligned}$$

Local displacements 3, 4 are shifted one LX string to the right; hence they are $2*(N+1)$ ahead of 1, 2:

$$\begin{aligned} (\text{Local \#3}) &= (\text{Local \#1}) + 2*(N+1) \\ (\text{Local \#4}) &= (\text{Local \#1}) + 2*(N+1) + 1 = (\text{Local \#2}) + 2*(N+1) \end{aligned}$$

and finally:

$$\begin{aligned} (\text{Local \#5}) &= (\text{Local \#1}) + 2*(N+1) + 2 = (\text{Local \#7}) + 2*(N+1) \\ (\text{Local \#6}) &= (\text{Local \#1}) + 2*(N+1) + 3 = (\text{Local \#8}) + 2*(N+1) \end{aligned}$$

With the algorithms derived above, it is quite easy to develop an efficient procedure to input the entire assembly list into the /DATA/ vector. The reader will recall that the pointer

for element number LNUM is stored in INTGR(IMASTR+LNUM-1). One additional integer variable, called NEXT, is required. NEXT is to be incremented after completion of the input for one element, so that the next value of NEXT is the address of the next available location in the master assembly list, i.e., NEXT= the value of the pointer for the next element. The entire assembly list is then generated and input by the following set of FØRTRAN instructions:

```

      (Values of M and N defined elsewhere in program)
      :
C     INITIALIZE PØINTER VALUE
      NEXT=IMASTR+NET
C     LØØP ØVER ELEMENT STRINGS
      DØ 20 LX=1, M
      DØ 20 LY=1, N
      LNUM=LY+N*(LX-1)
      IPTR=IMASTR+LNUM-1
C     ESTABLISH PØINTER FØR THE ELEMENT
      INTGR(IPTR)=NEXT
C     CALCULATE JD1=1 LESS THAN MASTER NØ. ØF 1ST DØF, JD3=1
C     LESS THAN MASTER NØ. ØF 3RD DØF
      JD1=2*(LY-1)+2*(N+1)*(LX-1)
      JD3=JD1+2*(N+1)
C     ASSIGN DOF MASTER NOS. TO LOCATIONS NEXT, NEXT + 1,..., NEXT + 7
C     IN /DATA/ VECTØR
      J=0
      K=0
      DØ 10 I=1,8
      INDEX=NEXT+I-1
      IF(I .GT. 2 .AND. I .LT. 7) GØ TØ 5
      J=J+1
      JD=JD1
      II=J
      GØ TØ 10
5    JD=JD3
      K=K+1
      II=K
10   INTGR(INDEX)=JD+II
C     INCREMENT NEXT LØCATION
20   NEXT=NEXT+8
C     STØRE A ZERO IN NEXT LØCATION (ASSUMING EXCESS STØRAGE
C     IN MASTER ARRAY)
      IF (NEXT .LE. LMASTR) INTGR(NEXT)=0
      :

```

The last IF statement above is a good form of insurance for cases in which the user may have overestimated the number of words required for his master assembly list.

A similar scheme for structures consisting of triangle elements (Figure 14) can be developed if the "right-side-up" and "upside-down" elements are treated as separate sets.

The automatic generation technique derived above was developed for a structure with complete topological regularity; however, it may be extended to cover large portions of less regular structures with relatively little additional programming. There is also a hidden advantage: the numbering strategy adopted in Figure 15 is not only easy to produce, but also minimizes the population of the master stiffness matrix if none of the elements have mid-side nodes.

5.3 A Short Note on Automatic Data Generation

The reader will recall that the hypothetical user in Section 4 chose to read in his element coordinates and basic properties from data cards. This procedure can be time-consuming and expensive for problems involving large numbers of elements, especially when the element set geometry is not regular in the sense of Subsection 5.1 and Figure 13. However, it often happens that the element set geometry is regular in the sense that the element coordinates may be calculated from a general algorithm based upon the element string concept discussed in the previous subsection, i.e.:

```

DØ 20 LX=1, M
DØ 20 LY=1, N
LJUM=LY+N*(LX-1)
( $\vec{X} = \vec{f}(LX, LY)$ )
:
:

```

where $\vec{X} = \{X1, Y1, X2, Y2, \dots\}$ is the element coordinate vector and $\vec{f}$ is a floating point function of LX and LY. An algorithm of this type may be used to generate the data base, as it is

needed, for calculation of element stiffness matrices and element stresses. This results in the trade-off of a slight increase in execution time and some decrease in required storage space, since it is not necessary to carry large vectors of global nodal coordinates in core.

5.4 A Handy Trick

There often occur plane elasticity problems in which a large number of degrees of freedom are to have prescribed displacements. For example, suppose a rectangular domain such as the one shown in Figure 15 is to be analyzed, and that all four edges of the domain are clamped. This means that 25 to 30 per cent to the total degrees of freedom will have prescribed displacements.

A significant amount of execution time may be saved by modifying the numbering scheme shown in Figure 15, so that the edge degrees of freedom have the largest global numbers. Let NDT be the total number of degrees and NFT be the total number of unconstrained degrees. Then the modified number scheme assigns:

1, 2, ..., NFT

to the unconstrained degrees and:

NFT+1, NFT+2, ..., NDT

to the degrees along the edges of the structure. Program stages 1 through 5 are completed in standard fashion.

However, just before factoring **K**, the user may fool FEABL by inserting:

ITEMP=NDT
NDT=NFT

in his MAIN subprogram. Subroutines FACTPD/FACTSD and SIMULQ will then solve only for the unknown displacements 1, 2, ..., NFT. At the beginning of Stage 7, when the full displacement vector may be required again, the user inserts:

NDT=ITEMP

The real value of this trick depends on a trade-off between execution time and core storage. The modified numbering scheme will result in a requirement for additional storage space for K , over what is needed by the numbering scheme discussed in Subsection 5.2. The excess requirement is given by:

$$\Delta S \approx (NDT - NFT) (NDT - B)$$

where B is the average semi-bandwidth of the unconstrained part of K .

5.5 Time-Saving Techniques for Design Studies

The "design study" approach to finite element analysis may take any of the following forms:

1. Consideration of a number of loading environments applied to a unique structure with unique displacement boundary conditions.
2. Analysis of a unique structure under various environments in which prescribed displacements are changed, as well as prescribed forces.
3. Consideration of variations in the structure itself to meet a given environment.

The first two categories are self-explanatory. An example of the third might be an analysis of a truss structure in which the compression bars are checked for buckling:

$$P_{bar} < P_{cr} = \pi^2 EI / l^2$$

If any bar were found to be under a compressive load greater than its P_{cr} , the program might incorporate an algorithm for redesigning the bar and re-analyzing the new structure.

The above categories of problems can be studied using FEABL with only minor modifications. No changes are required in the data location interfaces and if a change to the process sequence interface is required, it involves only an intelligent

application of the rules presented above. However, the analyst must be familiar with programming techniques required for communication between his computer's core and external storage devices (system disks, drums, tape units). These techniques vary from installation to installation and will not be discussed in specific terms here. The analyst should consult the operating manuals applicable to the system which he will be using.

Load environment case study is the easiest type of multi-solution problem to handle. Since the displacement boundary conditions are not varied, only one assembly and factoring of the master stiffness matrix need be done. The factored form of $\mathbf{K}$ is held in the /DATA/ vector while each prescribed vector is formed and the displacements are solved for. The quantity:

$$\mathbf{Q}^* = \hat{\mathbf{Q}}_E - \mathbf{K}_{FD} \hat{\mathbf{u}}_D$$

$\hat{\mathbf{Q}}_E$ = Assembled Element Equivalent Nodal Forces
and the prescribed displacements $\hat{\mathbf{u}}_D$ must be saved. These two quantities are found in the Force/Displacement Vector:

$$\mathbf{Q} = \left\{ \begin{matrix} \mathbf{Q}^* \\ \hat{\mathbf{u}}_D \end{matrix} \right\}$$

in the /DATA/ vector just after BCØN is called. (The prescribed external loads $\mathbf{Q}_P$, on the other hand, are not really needed until SIMULQ is called.) Figure 16 illustrates the modified FEABL process sequence which will accomplish this result.

The second category involves essentially the same techniques as the first. However, in this case the data which must be saved in external storage are:

1. $\hat{\mathbf{Q}}_E$ - Assembled element equivalent nodal forces.
2. $\mathbf{K}$ - Assembled master stiffness matrix prior to boundary condition application.

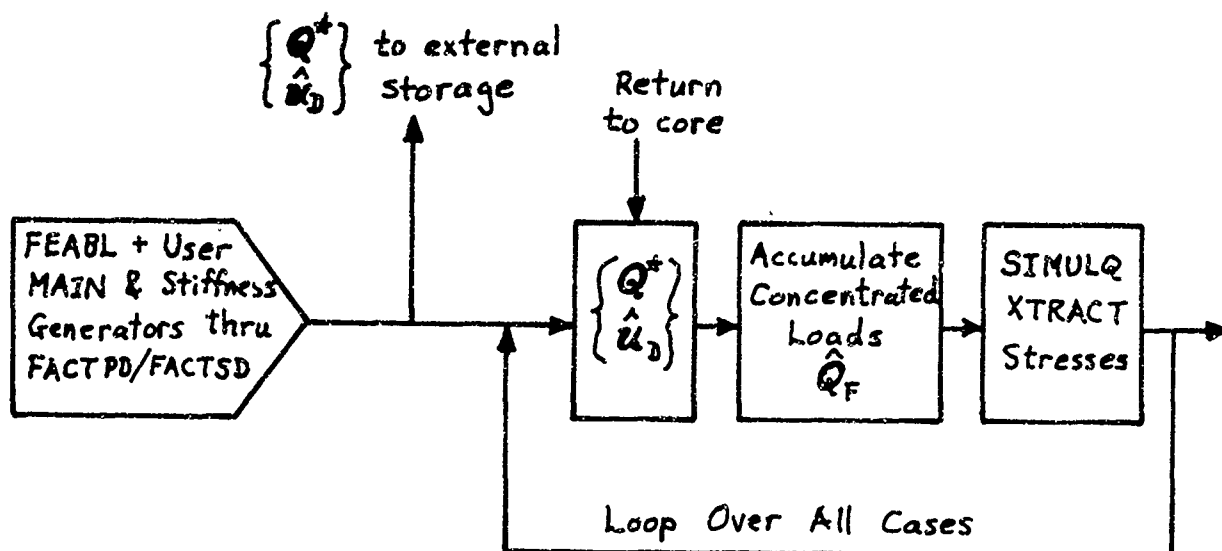


FIGURE 16

For each case the user must input the contents of the constraint vector, fetch $\hat{Q}_E$, accumulate $\hat{Q}_F$ and reset $\hat{u}_D$, fetch K and finally re-enter the standard FEABL process at the point where BCØN is called.

Significant time savings are gained in the third problem category if the design changes involve only a few elements in a structure composed of a large number of elements. Suppose that the displacement solution has been found from the force-displacement relations for the initial structure:

$$Ku = \hat{Q}_F + Q^*$$

During calculation of the element stresses, it is discovered that design changes are required in one or more elements; these changes will appear in the force-displacement relations as a modification ΔK of the master stiffness matrix and (possibly) a modification $\Delta \hat{Q}_E$ of the assembled element nodal force vector.

The new force-displacement relations:

$$(\mathbf{K} + \Delta\mathbf{K}) \mathbf{u}' = \hat{\mathbf{Q}}_F + \mathbf{Q}^* + \Delta\mathbf{Q}^*$$

must then be solved for the modified displacements $\mathbf{u}'$.

To program the design change technique with FEABL, the user must take the following nonstandard actions:

1. Output $\mathbf{K}$ and $\hat{\mathbf{Q}}_F + \hat{\mathbf{Q}}_E$ to external storage after assembly and input of prescribed quantities, but before calling BCØN.
2. After the initial displacement solution $\mathbf{u}$ has been obtained, transfer it to temporary core storage outside the /DATA/ vector. (The Force/Displacement Vector area in the /DATA/ vector will be required for accumulation of $\Delta\hat{\mathbf{Q}}_E$.) Zero the Force/Displacement Vector and the Master Stiffness Matrix Array.
3. Calculate stresses from $\mathbf{u}$, element-by-element. (The user will need his own version of XTRACT to extract the proper $\mathbf{u}_{el}$ from temporary core storage.) When a design change is required, calculate $\Delta\mathbf{k}_{el}$ and $\Delta\hat{\mathbf{Q}}_{el}$ and accumulate them using FEABL subroutine ASEMBL.
4. Apply rotation transformations (if any) to $\Delta\mathbf{K}$ and $\Delta\hat{\mathbf{Q}}_E$, reset entries of $\Delta\hat{\mathbf{Q}}_E$ to zero where displacements are prescribed.
5. Fetch $\mathbf{K}$ and $\hat{\mathbf{Q}}_F + \hat{\mathbf{Q}}_E$ from external storage and accumulate them to $\Delta\mathbf{K}$ and $\Delta\hat{\mathbf{Q}}_E$. Apply boundary conditions and solve for $\mathbf{u}'$.

The above process may be repeated in an iterative design process, with $\mathbf{K} + \Delta\mathbf{K}$, $\hat{\mathbf{Q}}_F + \mathbf{Q}^* + \Delta\mathbf{Q}^*$ at the beginning of each new design step playing the role of the initial values $\mathbf{K}$, $\hat{\mathbf{Q}}_F + \mathbf{Q}^*$. This procedure is referred to as iterative updating, and is also applicable to nonlinear elastic and elastic-plastic analysis of continua. In these types of analysis the "design change"

results from following a material stress-strain curve and/or testing the satisfaction of a yield condition (e.g., the Mises-Hencky criterion).

5.6 Substructuring with FEABL

The FEABL software system has been designed primarily for in-core solutions. Up to 1,500 degrees of freedom can be handled on currently available hardware with 500 KBYTE (125 KWord) memory. FEABL's in-core capability may be extended by means of the substructuring technique outlined briefly here.

Figure 17 illustrates a domain which has been divided into a small number of substructures. The desired stress solution accuracy is on a scale much smaller than the substructure dimension,

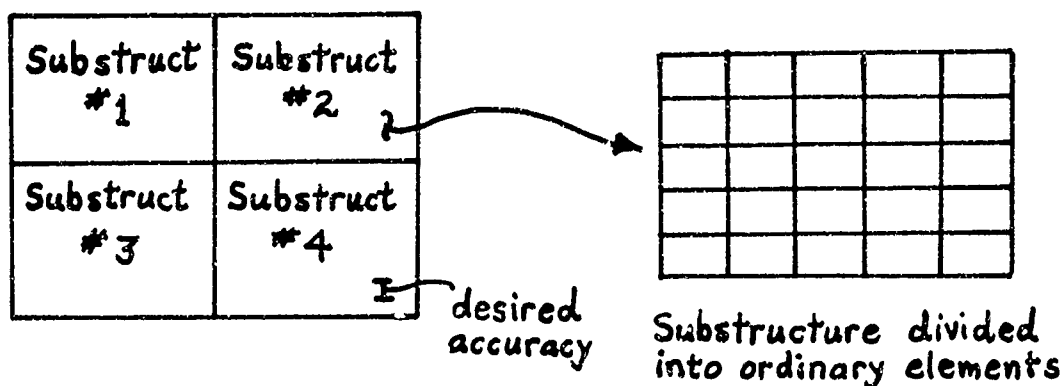


FIGURE 17

so each substructure is subdivided further into ordinary elements. Let the subscripts I and B refer, respectively, to the interior and boundary degrees of freedom in a substructure. Then the

force-displacement relations for the substructure may be partitioned into:

$$\begin{bmatrix} \mathbf{K}_{BB} & \mathbf{K}_{BI} \\ \mathbf{K}_{BI}^T & \mathbf{K}_{II} \end{bmatrix} \begin{Bmatrix} \mathbf{u}_B \\ \mathbf{u}_I \end{Bmatrix} = \begin{Bmatrix} \mathbf{Q}_B \\ \mathbf{Q}_I \end{Bmatrix}$$

The interior degrees of freedom may be eliminated by the process of static condensation, which transforms the substructure force-displacement relations to the form:

$$\mathbf{K}_{BB}^c \mathbf{u}_B = (\mathbf{K}_{BB} - \mathbf{K}_{BI} \mathbf{K}_{II}^{-1} \mathbf{K}_{BI}^T) \mathbf{u}_B = \mathbf{Q}_B - \mathbf{K}_{BI} \mathbf{K}_{II}^{-1} \mathbf{Q}_I = \mathbf{Q}_B^c$$

Each substructure may now be considered as a "superelement" having degrees of freedom only on the interelement boundaries, as shown in Figure 18. The quantities $\mathbf{K}_{BB}^c$ and $\mathbf{Q}_B^c$ are then

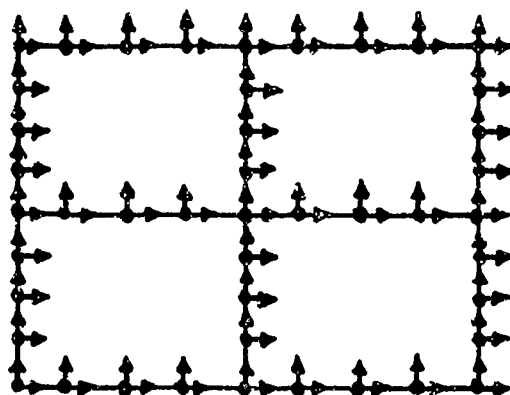


FIGURE 18

assembled in the same manner as the stiffness matrices and equivalent nodal force vectors for ordinary elements. After the displacement solution on the substructure boundaries has been obtained, the original force-displacement relations for each substructure may be used, with $\mathbf{u}_B$ as prescribed displacements, to obtain the interior solution.

In addition to extension of FEABL's problem size capability, the substructuring technique may also be used to reduce roundoff error and to save execution time. All of these improvements depend upon optimization of the relative numbers of substructures and of ordinary elements within the substructures. Use of the substructuring technique with FEABL software will be discussed in detail in a future publication.

5.7 Error Estimation Methods

Rigorous mathematical proofs exist showing that a finite element analysis for the stress and displacement distributions in a structure converges to the exact solution as the number of elements in a given domain is increased, provided only that certain easily satisfied restrictions are obeyed (Ref.3). However, since all calculations done in a digital computer are imprecise, errors due to roundoff will occur. It has been shown (Ref. 4) that roundoff error increases in an extremely complex way as the number of elements is increased. Therefore, it is advisable to resort to some numerical method which will produce a reasonable estimate of the error, and which does not depend upon the details of what types of elements are used or what boundary conditions are applied to the structure. Four methods are presented here.

5.7.1 Irons' Energy Variance Criterion

The energy criterion proposed by Irons (Ref. 5) is the most economical, requiring no more time than the calculation of the rounding error parameter discussed in Subsection 3.2.6. The diagonal entries K_{ii} of the master stiffness matrix must be saved, either in temporary core storage or on an external unit. After the displacement solution has been obtained, the energy variance can be calculated from:

$$C = 2^{-p+1} \sqrt{B} \frac{\sum_{i=1}^{NDT} K_{ii} (u_i)^2}{\sum_{i=1}^{NDT} \sum_{j=1}^{NDT} K_{ij} u_i u_j}$$

where p is the computer precision in decimal places and B is the average semi-bandwidth of K . The value of B can be obtained easily from the address index parameters*:

$$B = \text{FLØAT}(\text{LK}+1-\text{IK}) / \text{FLØAT}(\text{NDT})$$

One half the value of the denominator of the energy variance expression is provided to the user by FEABL subroutine SIMULQ in the argument ENERGY of that subroutine.

Some care must be exercised in applying Irons' energy criterion. The user will obtain unrealistically large values of C in problems in which the structure is constrained very lightly, and in which large amounts of "self-energy" can be stored when a single degree of freedom is displaced. The cantilever beam is a good example of a structure to which the Irons' criterion cannot be applied.

5.7.2 The Residual Force Method

Calculation of residual forces provides a more detailed picture of the distribution of errors through the structure. The master stiffness matrix may be saved in external storage for this purpose immediately after calling BCØN. Approximate values of the reaction forces at degrees where displacements were prescribed may be obtained as well by saving K just prior to calling BCØN.

* FLØAT is an IBM library function which converts integers to floating point numbers.

Let u_F^* be the approximate displacement solution obtained from the constrained force-displacement relations:

$$\begin{bmatrix} K_{FF} & 0 \\ 0 & I \end{bmatrix} \begin{Bmatrix} u_F \\ \hat{u}_D \end{Bmatrix} = \begin{Bmatrix} \hat{Q}_F - K_{FD} \hat{u}_D \\ \hat{u}_D \end{Bmatrix}$$

Then:

$$Q^* = \begin{Bmatrix} Q_F^* \\ Q_D^* \end{Bmatrix} = K \begin{Bmatrix} u_F^* \\ \hat{u}_D \end{Bmatrix}$$

is the approximate force vector. When K has been returned to core after the displacement solution has been obtained, Q^* can be calculated by the following algorithm:

```

DIMENSION R(500)
C VECTOR R MUST ALLOW ENOUGH STORAGE FOR THE FULL
C FORCE VECTOR Q-STAR IF REACTION FORCES ARE
C DESIRED. OTHERWISE, ENOUGH FOR THE UNCONSTRAINED
C DEGREES OF FREEDOM MUST BE ALLOWED
C ALLOCATE EXTERNAL FILES
:
: (Standard FEABL process sequence, except where noted)
:
:
C K MATRIX TO EXTERNAL STORAGE
WRITE (...) (REAL(I), I=IK,LK)
C FORCE/DISPL MUST ALSO BE SAVED FOR COMPARISON LATER
WRITE (...) (REAL(I), I=IQ, LQ)
CALL BC0N
CALL FACTPD
CALL SIMULQ(STRE)
:
: (Stress solution, if desired)
:
:
C RETURN K MATRIX FOR CALCULATION OF Q-STAR
READ (...) (REAL(I), I=IK, LK)
C INITIALIZE CONSTRAINED ROW POINTER AT 1ST NONZERO ROW
DO 50 II=IC0N, LC0N
IF(INTGR(II) .EQ. 0) GO TO 50
NC=II
GO TO 60
50 CONTINUE

```

```

C  INITIALIZE R VECTOR POINTER
60 NR=1
C  LOOP OVER DEGREES OF FREEDOM, SKIPPING CONSTRAINED DEGREES
DØ 70 IRØW=1, NDT
IF(INTGR(NC) .NE. IRØW) GØ TØ 71
C  UPDATE RØW POINTER
NC=NC+1
GØ TØ 70
C  FORM SUMMATION OF K(IRØW, J)* U(J), J=LNZ TØ NDT
71 INIT=ILNZ+IRØW-1
INIT=INTGR(INIT)
KK=IKØUNT+IRØW-1
KK=INTGR(KK)
SUM=0.
DØ 72 J=INIT, IRØW
KADR=KK+J
JJ=IQ+J-1
72 SUM=SUM+REAL(KADR)* REAL(JJ)
C  REMAINDER OF SUM MUST TAKE K ENTRIES FROM CØL IRØW
INIT=IROW+1
IF(INIT .GT. NDT) GØ TØ 74
DØ 73 J=INIT, NDT
C  MAKE SURE RØW J HAS AN ENTRY IN CØL IRØW
JJ=ILNZ+J-1
IF(INTGR(JJ) .GT. IRØW) GØ TØ 73
KADR=IKØUNT+J-1
KADR=INTGR(KADR)+IRØW
JJ=IQ+J-1
SUM=SUM+REAL(KADR)*REAL(JJ)
73 CØNTINUE
C  PLACE SUM IN NEXT AVAIL R LØCATION AND UPDATE R POINTER
74 R(NR)=SUM
NR=NR+1
70 CØNTINUE
C  RETURN ØRIGINAL Q TØ CØRE FØR CØMPARISØN
READ (...) (REAL(I), I=IQ, LQ)
C  CALCULATE RESIDUAL FØRCES Q-(Q-STAR)=Q-R
C  REINITIALIZE NC AND NR
NC=II
NR=1
DØ 80 IRØW=1, NDT
IF(INTGR(NC) .NE. IRØW) GØ TØ 81
NC=NC+1
GØ TØ 80
81 JJ=IQ+IRØW-1
R(NR)=REAL(JJ)-R(NR)
WRITE (KW, 500) IRØW, R(NR)
500 FØRMAT (20H RESIDUAL FØRCE NØ.,I6, 1X, 1H=, E10.3)
NR=NR+1
80 CØNTINUE

```

In the above algorithm, the vector of residual forces:

$$\mathbf{R}_F = \hat{\mathbf{Q}}_F - \mathbf{Q}_F^* = \Delta \mathbf{Q}_F$$

has been calculated. The entries of $\mathbf{R}_F$ provide the detailed picture mentioned at the beginning of this subsection. Another useful parameter for overall error measurement is the force vector magnitude ratio:

$$r_F = \left[\frac{\sum (R_i^2)}{\sum (\hat{Q}_i - \sum K_{ij} u_j)^2} \right]^{1/2} = \frac{|\Delta \mathbf{Q}_F|}{|\hat{\mathbf{Q}}_F|}$$

where the summations extend only over the unconstrained degrees of freedom.

5.7.3 Re-Solution for Residual Displacements

Although the residual force vector is fairly easy to calculate, the interpretation of its meaning is not a trivial task. The displacement solution must certainly be judged acceptable if, for example, there are many residual forces on the order of 1 lb. at degrees where forces of 1,000 lb. were applied originally. However, a common situation in finite element analysis is that the applied force is zero at many degrees of freedom. What does a 1 lb. residual force mean at these points? The averaged measure r_F presented in the previous section relieves this detailed interpretation problem to some degree. However, the averaged measure of greatest interest to the analyst is the displacement vector magnitude ratio:

$$d_F = \frac{|\Delta \mathbf{u}_F|}{|\mathbf{u}_F|}$$

Unfortunately, no simple relation exists between d_F and the force vector magnitude ratio r_F . If K is an ill-conditioned matrix, d_F may in fact be much larger than r_F .

If the analyst is willing to spend some additional computing time, the displacement residuals and an approximate calculation of d_F may be obtained by re-resolution. This technique requires an additional external storage file capable of holding K . Just before K is returned to core in the algorithm of Subsection 5.7.2, its factored form LDL^T is read into this extra file. Then, picking up where the previous algorithm ended, the original displacement vector magnitude is calculated, LDL^T is returned to core and the contents of R are transferred to the proper locations in the force/displacement block of the /DATA/ vector. Re-resolution is now done simply by calling SIMULQ again, after which Δu_F will be found in the force/displacement block. The displacement vector magnitude ratio obtained from this procedure is actually:

$$d_F^* = \frac{|\Delta u_F|}{|u_F^*|}$$

and $d_F \neq d_F^*$ unless $|\Delta u_F| \ll |u_F^*|$.

5.7.4 The Method of Rigid Body Modes

A somewhat less cumbersome technique for error measurement can be employed when a structure is modeled by elements which contain a full set of rigid body modes in their assumed displacement fields. Let

$$r = \begin{Bmatrix} r_F \\ \hat{r}_b \end{Bmatrix}$$

be any rigid body displacement vector for the whole structure (e.g., unit vertical translation at every node). Then if r is introduced into the unconstrained force-displacement relations and the right hand side is calculated, there will result:

$$\mathbf{K} \mathbf{r} = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}$$

to the accuracy of the user's computer. Conversely, solution of the constrained force-displacement relations:

$$\begin{bmatrix} \mathbf{K}_{FF} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{Bmatrix} \mathbf{u}_F \\ \hat{\mathbf{r}}_D \end{Bmatrix} = \begin{Bmatrix} -\mathbf{K}_{FD} \hat{\mathbf{r}}_D \\ \hat{\mathbf{r}}_D \end{Bmatrix}$$

would result in $\mathbf{u}_F = \mathbf{r}_F$ to the accuracy of the computer if there were no roundoff error. Now, since the exact $\mathbf{r}_F$ can be inferred from $\hat{\mathbf{r}}_D$ merely by inspection, the above problem may be solved as an auxiliary to the real problem, and the error measure:

$$d_{RB} = \frac{|\mathbf{r}_F - \mathbf{u}_F|}{|\mathbf{r}_F|}$$

may be calculated. The rigid body mode technique requires the saving only of $\mathbf{K}$ (before constraint). A simple algorithm will serve to calculate $\mathbf{r}_F$ and $\mathbf{r}_F - \mathbf{u}_F$, and the additional execution time required amounts only to calling each of BCØN, FACTPD (or FACTSD) and SIMULQ once extra. Also, the lengthy residual force and re-solution algorithm in the MAIN program is avoided. Recent tests of the method of rigid body modes on a cantilever beam have shown that when $d_F < 0.1$, $d_{RB} \ll d_F$. However, when $d_F > 0.1$ (the region of primary interest) d_{RB} performs as well as d_F^* .

REFERENCES

1. Orringer, O. "The Effect of Cross Section Stress Concentration on the Compressive Strength of a Unidirectional Fiber Composite." ASRL TR 162-4 (in preparation)
2. Mack, E. W., Berg, B. A., and Witmer, E. A. "An Improved Discrete-Element Analysis and Program for the Linear-Elastic Static Analysis of Meridionally-Curved, Variable-Thickness, Branched Thin Shells of Revolution Subjected to General External Mechanical and Thermal Loads, Part 2-The SABOR 4 Program." Massachusetts Institute of Technology, Aeroelastic and Structures Research Laboratory, ASRL TR 146-4, Part 2 (also SAMSO TR 68-310, Part 2), March 1968. (AD 840 614L).
3. Tong, P. and Pian, T. H. H. "The Convengence of Finite Element Method in Solving Linear Elastic Problems." Int'l. J. Solids Structures, Vol. 3, 1967. pp. 865-879.
4. Tong, P. "On the Numerical Problems of the Finite Element Methods." Study No. 5, Symposium on Computer-Aided Engineering, University of Waterloo, Ontario, Canada, 1970.
5. Irons, B. M. and Kan, T. K. Y. "Equation-Solving Algorithms for the Finite-Element Methods." Paper No. V-4-2, University of Illinois Conference on Numerical Methods in Structural Analysis, 1971.

APPENDIX A

COMMON AREAS DATA REQUIREMENTS

The following table summarizes what information is expected by each FEABL subroutine in the four control parameter COMMON areas and in the /DATA/ vector COMMON area.

SUBROUTINE NAME COMMON AREA	ASEMBL	BCON	FACTPD/FACTSD	ØRK	ROTATE	SETUP	SIMULQ	XTRACT
/IO/ (Printer Code KW Only)	✓	✓	✓	✓	✓	✓	✓	
/SIZE/ NET, NDT		✓	✓	✓	✓	✓	✓	
/BEGIN/ Address index parameters	✓	✓	✓	✓	✓		✓	✓
/END/ Address index parameters		✓	✓	✓ Except LK			✓	
/DATA/ Vector:								
1. Constraint Vector		✓	✓			(c)	✓	
2. Address Count Vector	✓	✓	✓		✓		✓	
3. LNZ Vector	✓	✓	✓		✓		✓	
4. Assembly List	✓			✓				✓
5. F/D Vector	(a)	✓			Only $\hat{Q}_E$ (b)		✓	u
6. K Matrix	(a)	✓	✓		✓		LDL ^T	

(a) These blocks are zeroed by ØRK before the first element is assembled.

(b) $\hat{Q}_E$ = Vector of assembled element equivalent nodal forces.

(c) Block zeroed by SETUP

APPENDIX B

APPLYING SHOEHORN AND STOPWATCH

This appendix contains data from which the user may estimate the total amount of core storage required by a FEABL-based program and the approximate execution time (CPU time) the run will take. Such estimates will prove useful aids in making trade-off decisions. The data given in this appendix is based on runs done on an IBM 370/155 using the FØRTPAN G compiler. The numbers will vary somewhat from one machine or compiler to another.

B.1 Estimation of Core Storage Requirement

The following table gives the length of each FEABL subroutine in BYTES (as compiled in FØRTRAN G) and words, and the deck size. On IBM 360 and 370 series hardware, core storage calculations are normally done in terms of BYTES, while words are used on many

Subroutine Name	Words	BYTES	No. of Cards in Deck*
ASEMBL	410	1,642	58
BCØN	599	2,396	125
FACTPD/FACTSD	669	2,674	126
ØRK	517	2,070	106
RØTATE	1,858	7,430	316
SETUP	726	2,904	94
SIMULQ	600	2,398	127
XTRACT	130	522	23
FEABL Software-Total	5,509	22,036	975

other hardware systems. The total storage requirement for programs plus data can be estimated as follows:

* Includes all comment cards

Item:	KWords	KBYTES
1. FEABL Software	5.5	22.1
2. User programs (a)	2.5	10.0
3. System library subprograms (b)	5.1	22.0
4. /DATA/ Vector (c)	(L)	(4L)
5. Four control parameter /COMMON/ areas	--	0.1
Totals	13.1+(L)	53.2+(4L)

- (a) Estimate for a typical analysis with a MAIN program and two generator subroutines.
- (b) Includes library functions such as SIN, COS, SQRT and systems management subroutines.
- (c) L=the dimension of the /DATA/ vector in KWords (1000 words).

B.2 Estimation of CPU Time Requirement

The process of factoring the master stiffness matrix into its triple product:

$$K = LDL^T$$

is the primary time consumer in any finite element analysis. Some study of the algorithms in FEABL subroutine FACTPD/FACTSD will convince the reader that the CPU time consumed is proportional to NB^2 , where N is the total number of unconstrained DOF in the assembled structure and B is the average semi-bandwidth of the master stiffness matrix. B may be calculated approximately from N and the population density of **K** :

$$B = \frac{(N+1)P}{2}$$

where

$$P = \text{Population Density} = \frac{\text{Total No. of Stored Entries}}{\text{Total Entries in Full Lower Triangle}}$$

Experience on the IBM 370/155 indicates that the required CPU time is given approximately by:

$$\text{time} = NB^2 \times 10^{-6} \text{ minutes}$$

APPENDIX C

FØRTRAN IV LISTING OF FEABL SOFTWARE

The eight subroutines of the FEABL software system are listed in alphabetical order in this appendix:

Subroutine	Page
ASEMBL	71
BCØN	73
FACTPD/FACTSD	77
ØRK	81
RØTATE	84
SETUP	93
SIMULQ	96
XTRACT	100

The code is for FEABL Version 1 Release 1, with a 10,000-word /DATA/ vector. The following actions will convert the program to Version 2 ("On-Line") as explained in Section 2.3.

SUBROUTINE	Change DIMENSIONs of REAL, INTGR to 2	Delete Declaration
ASEMBL	Card No. 0007	Card Nos. 0017 and 0020
BCØN	0007	0012 and 0015
FACTPD/FACTSD	0008	0013 and 0016
ØRK	0015	0020 and 0023
RØTATE	0007	0013 and 0017
SETUP	0007	0013 and 0015
SIMULQ	0007	0012 and 0015
XTRACT	0007	0011 and 0013

```

ASEM0001
SUBROUTINE ASEMBL(LNUM,NDE,ELK,ELQ)
C*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AERODELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C*****
DIMENSION REAL(10000),INTGR(10000)
DIMENSION ELK(NDE,NDE), ELQ(NDE)
C*****
C CHANGE THIS DIMENSION STATEMENT IF YOU HAVE RECEIVED AN ERROR MESSAGE
C INDICATING LENGTH OF MNUM VECTOR HAS BEEN EXCEEDED. ALSO CHANGE
C INSTRUCTION INDICATED BELOW
DIMENSION MNUM(100)
C*****
COMMON /I0/ KR, KW, KP, KI1, KI2, KI3
COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK
COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
EQUIVALENCE (REAL(1), INTGR(1))
C PRINT CONTROL
901 FORMAT(1H0,110,27H DOF/ELEMENT EXCEEDS LENGTH,14,26H SPECIFIED FORASEM0022
1 MNUM VECTOR,/,1X,44HERROR OCCURED DURING ASSEMBLY OF ELEMENT NO.,ASEM0023
2 110,/,1X,93HUSER MUST CORRECT CALL ARGUMENTS IN MAIN OR CHANGE LEASEM0024
3NGTH OF MNUM VECTOR IN SUBROUTINE ASEMBL,/,1X,20HEXECUTION TERMINAASEM0025
4TED)
IKOUM1 = IKOUNT-1
IQM1 = IQ-1
C*****
C VALUE OF LENGTH MUST = DIMENSION OF MNUM
LENGTH = 100
C*****
IF (NDE.LE. LENGTH) GO TO 1
WRITE (KW,901) NDE, LENGTH, LNUM
STOP
C GET MASTER KJW/CUL NOS FOR ELEMENT AND STORE IN MNUM
ASEM0002
ASEM0003
ASEM0004
ASEM0005
ASEM0006
ASEM0007
ASEM0008
ASEM0009
ASEM0010
ASEM0011
ASEM0012
ASEM0013
ASEM0014
ASEM0015
ASEM0016
ASEM0017
ASEM0018
ASEM0019
ASEM0020
ASEM0021
ASEM0022
ASEM0023
ASEM0024
ASEM0025
ASEM0026
ASEM0027
ASEM0028
ASEM0029
ASEM0030
ASEM0031
ASEM0032
ASEM0033
ASEM0034
ASEM0035
ASEM0036

```

ASEM00037
 ASEM00038
 ASEM00039
 ASEM00040
 ASEM00041
 ASEM00042
 ASEM00043
 ASEM00044
 ASEM00045
 ASEM00046
 ASEM00047
 ASEM00048
 ASEM00049
 ASEM00050
 ASEM00051
 ASEM00052
 ASEM00053
 ASEM00054
 ASEM00055
 ASEM00056
 ASEM00057
 ASEM00058

```

1 INDEX = IMASTR+LNUM-1
  DO 2 I = 1,NDE
    J = INTGR(INDEX)+I-1
    2 MNUM(I) = INTGR(J)
  C LOOP OVER ROWS OF ELQ AND OVER LOWER TRIANGLE OF ELEMENT K MATRIX
    DO 4 LROW = 1,NDE
      INDEX = IQM1+MNUM(LROW)
      C ASSEMBLE ELEMENT EQUIVALENT NODAL FORCE INTO Q VECTOR
        REAL(INDEX) = REAL(INDEX)+ELQ(LROW)
        DO 4 LCUL = 1,LROW
          MROW = MNUM(LROW)
          MCOL = MNUM(LCUL)
          IF (MROW .GE. MCOL) GO TO 3
          MROW = MNUM(LCUL)
          MCOL = MNUM(LRJM)
        C CALCULATE ABSOLUTE ADDRESS OF K (MROW,MCCL)
          3 INDEX = IKJUM1+MROW
          KADR = INTGR(INDEX)+MCUL
        C ASSEMBLE STIFFNESS COEFFICIENT
          4 REAL(KADR) = REAL(KADR)+ELK(LROW,LCCL)
          RETURN
        END
  
```

```

SUBROUTINE BCUN
C*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AEROElastic AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C*****
DIMENSION REAL(10000),INTGR(10000)
COMMON /IO/ KZ, KW, KF, KT1, KT2, KT3
COMMON /SIZE/ NET, NDT
COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK
COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK
COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
EQUIVALENCE (REAL(1), INTGR(1))
C PRINT CONTROL
901 FORMAT('9HDISPLACEMENT CONSTRAINTS HAVE BEEN APPLIED TO THE FOLLOWING DEGREES OF FREEDOM,/5X,7HDCF NO.,2X,12HDISPLACEMENT')
902 FORMAT ('2X,11O,2X,E10.3')
903 FORMAT('1X,72H*****ABOVE DOF NUMBER APPEARS TWICE*****')
1 IN CONSTRAINT LIST,/1X,85HEXCUTION TERMINATED IN SUBROUTINE BCUN
2N DUE TO POSSIBILITY OF BOUNCARY CONDITION ERROR)
904 FORMAT('62HYOUR STRUCTURE IS FLYING FREE. PLEASE CONSTRAIN IT NEXT TIME./,1X,28HEXCUTION TERMINATED IN BCUN')
IKOUNT = IKOUNT-1
ILNZM1 = ILNZ-1
IQM1 = IQ-1
C PRINT ENTRY MESSAGE
WRITE (KW,901)
C ORDER THE CONSTRAINT ROW NUMBERS IN ASCENDING SEQUENCE
LAST = LCON-1
IF (LCUN .GT. 0) GO TO 1
WRITE (KW,904)
STOP
1 IFLAG = 0
DO 2 I = ICUN,LAST

```

```

IF (INTGR(I+1) .GE. INTGR(I)) GO TO 2
J = INTGR(I)
INTGR(I) = INTGR(I+1)
INTGR(I+1) = J
IFLAG = 1
2 CONTINUE
IF (IFLAG .EQ. 1) GO TO 1
C CHECK TO SEE IF ANY ROW NUMBERS HAVE BEEN ENTERED IN CONSTRAINT
C VECTOR - ABORT THE RUN IF NONE HAVE BEEN
J = 0
DO 100 I = ICUN,LCON
100 J = J+INTGR(I)
IF (J .GT. 0) GO TO 200
WRITE (KW,904)
STOP
C OUTPUT CONSTRAINT LIST
200 DO 4 I = ICUN,LCON
4 IF (INTGR(I) .EQ. 0) GO TO 4
C CHECK FOR REPEATED DOF AFTER 1ST ONE
IF (I .EQ. ICUN) GO TO 3
IF (INTGR(I) .NE. INTGR(I-1)) GO TO 3
WRITE (KW,903)
STOP
3 J = ICM1+INTGR(I)
WRITE (KW,902) INTGR(I), REAL(J)
4 CONTINUE
C LOOP OVER CONSTRAINED DOF FOR MODIFICATION OF COMPLETELY ASSEMBLED
C FORCE VECTOR
DO 71 I = ICUN,LCON
71 IF (INTGR(I) .EQ. 0) GO TO 71
C CHECK IF PRESCRIBED DISPLACEMENT = 0 --- IF IT DOES, SKIP FORCE VECTOR
MRUW = INTGR(I)
M = ICM1+MRUW
IF (REAL(M) .EQ. 0.) GO TO 71
C DISPL .NE. 0 --- LOOP OVER ALL ROWS TO MODIFY FORCE VECTOR --- SKIP
C CONSTRAINED KUWS (CONTROLLED BY VALUE OF NEXT)

```

```

BCON0037
BCON0038
BCON0039
BCON0040
BCCN0041
BCCN0042
BCON0043
BCON0044
BCON0045
BCON0046
BCON0047
BCCN0048
BCON0049
BCON0050
BCCN0051
BCON0052
BCCN0053
BCON0054
BCON0055
BCON0056
BCON0057
BCCN0058
BCON0059
BCON0060
BCON0061
BCON0062
BCON0063
BCON0064
BCON0065
BCON0066
BCON0067
BCON0068
BCON0069
BCON0070
BCON0071
BCON0072

```

BC0N0073
 BC0N0074
 BC0N0075
 BC0N0076
 BC0N0077
 BC0N0078
 BC0N0079
 BC0N0080
 BC0N0081
 BC0N0082
 BC0N0083
 BC0N0084
 BC0N0085
 BC0N0086
 BC0N0087
 BC0N0088
 BC0N0089
 BC0N0090
 BC0N0091
 BC0N0092
 BC0N0093
 BC0N0094
 BC0N0095
 BC0N0096
 BC0N0097
 BC0N0098
 BC0N0099
 BC0N0100
 BC0N0101
 BC0N0102
 BC0N0103
 BC0N0104
 BC0N0105
 BC0N0106
 BC0N0107
 BC0N0108

```

NEXT = ICUN
DO 7 NROW = 1,NDT
  IF (NROW .NE. INTGR(NEXT)) GO TO 5
  NEXT = NEXT+1
  GO TO 7
5 IF (MROW .GT. VROW) GO TO 51
C CHECK FOR COUPLING OF ROW NROW WITH COL MROW
J = ILN2M1+NROW
IF (INTGR(J) .GT. MROW) GO TO 7
IROW = NROW
ICOL = MROW
GO TO 6
C CHECK FOR COUPLING OF ROW MROW WITH COL NROW
51 J = ILN2M1+MROW
IF (INTGR(J) .GT. NROW) GO TO 7
IROW = MROW
ICOL = NROW
C SUBTRACT K*(PRESCK DISPL) FROM FORCE VECTOR
6 KADR = IKDUM1+IROW
KADR = INTGR(KADR)+ICOL
N = IQM1+NROW
REAL(N) = REAL(N)-REAL(KADR)*REAL(M)
7 CONTINUE
71 CONTINUE
C LOOP OVER CONSTRAINED ROWS TO DECOUPLE THEM FROM REST OF K MATRIX
DO 11 I = ICUN,LCCN
  IF (INTGR(I) .EQ. 0) GO TO 11
  MROW = INTGR(I)
  INIT = ILN2M1+MROW
  INIT = INTGR(INIT)
  C SET ROK = 0
  M = IKDUM1+MROW
  M = INTGR(M)
  DO 8 MCUL = INIT,MROW
    KADR = M+MCUL
    8 REAL(KADR) = 0.
  
```

```

C SET COLUMN = J IN ROWS WHOSE LNZE COL NC IS .LE. MROW -- SKIP THIS
C SECTION IF MKUM IS THE LAST ROW
  IF (MRJW .EQ. NDT) GO TO 10
  INIT = MRUM+1
  DO 9 NROW = INIT, NDT
    N = ILNZN1+MKUM
    IF (INTGR(N) .GT. MROW) GO TO 9
    KADR = IKUUM1+NROW
    KADR = INTGR(KADR)+MROW
    REAL(KADR) = 0.
  9 CONTINUE
C SET DIAGONAL ENTRY = 1
  10 KADR = M+MRUM
    REAL(KADR) = 1.
  11 CONTINUE
  RETURN
  END
BCON0109
BCON0110
BCON0111
BCON0112
BCON0113
BCON0114
BCON0115
BCON0116
BCON0117
BCON0118
BCON0119
BCON0120
BCON0121
BCON0122
BCON0123
BCON0124
BCON0125

```

```

SUBROUTINE FACISD
*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AERUELASTIC AND STRUCURES RESEARCH LABCRATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
*****
C THIS IS ALSO SUBROUTINE FACIPD - SEE ENTRY POINT BELOW
*****
      DIMENSION REAL(10000), INTR(10000)
      COMMON /IO/ KR, KW, KP, K11, KT2, KT3
      COMMON /SIZE/ NET, NOT
      COMMON /BEGIN/ ICON, IKOUNT, ILNZ, IMASTR, IQ, IK
      COMMON /END/ LCON, LKOUNT, LLNZ, LMASTR, LQ, LK
      COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
      EQUIVALENCE (REAL(1), INTR(1))
C PRINT CONTROL
901 FORMAT (1H0,47X,25HTRIPL FFACTOR ENTRY POINT,/,53H K MATRIX NOT POFACT0018
      2SITIVE-DEFINITE IN THE FOLLOWING ROWS )
902 FORMAT (40X,112)
903 FORMAT (48X,4HNONE )
904 FORMAT (58H USER SPECIFIED SEMI-DEFINITE MATRIX. EXECUTION CONTINFACT0022
      2UES )
905 FORMAT (40H USER SPECIFIED POSITIVE-DEFINITE MATRIX,/,21H EXECUTIOFACT0024
      2N TERMINATES )
906 FORMAT (25HOK MATRIX SINGULAR IN ROW,112,/,21H EXECUTION TERMINATEFACT0026
      2S )
907 FORMAT (61HULARGEST ROUNDING ERROR IN DIAGONAL FACTORING OCCURRED FACT0028
      2IN ROW,112,/,43H NUMBER OF LOWEST SIGNIFICANT FIGURES LOST=,13) FACT0029
908 FORMAT (42H ROUNDING ERROR EXCEEDS ACCEPTABLE MAXIMUM,/,21H EXECUTFACT0030
      2ION TERMINATES )
C SET ENTRY FLAG
      NPD = 0
      GO TO 1
      ENTRY FACIPD
      NPD = 1

```

```

C PRINT ENTRY MESSAGE
1 WRITE (KW,9J1)
   IKOUN1 = IKOUN1-1
   ILNZM1 = ILNZ-1
C INITIALIZE AT 1ST NONZERO ENTRY IN CONSTRAINT VECTOR
DO 2 I = 1,ICJN,LCON
  IF (INTGR(I) .EQ. 0) GO TO 2
  GO TO 3
2 CONTINUE
3 NEXT = 1
  IF (INTGR(NEXT) .EQ. 1) NEXT = NEXT+1
C INITIALIZE ERROR PARAMETERS
  JROW = 1
  MPD = 0
  TEST = 1.
C DO FIRST ROW AS SPECIAL CASE
  IF (REAL(IK) .EQ. 0.) GO TO 14
  IF (REAL(IK) .GT. 0.) GO TO 4
C NPD MESSAGE AND FLAG
  WRITE (KW,9J2) JROW
  MPD = 1
C LOOP OVER REMAINING ROWS
  4 DO 13 MROW = 2,NDT
C CHECK FOR CONSTRAINED ROW - SKIP IF FOUND AND RESET FOR THE NEXT ONE
  IF (MROW .NE. INTGR(NEXT)) GO TO 5
  NEXT = NEXT+1
  GO TO 13
C FREE ROW - FACTOR FROM LN2 CUL NO TO ROW NO
  5 M = ILNZM1+MROW
  M = INTGR(M)
  MM = IKJUM1+MROW
  MM = INTGR(MM)
  DO 12 MCUL = M,MROW
    SUM = 0.
    NN = IKJUM1+MCUL
    NN = INTGR(NN)

```

FACT0037
 FACT0038
 FACT0039
 FACT0040
 FACT0041
 FACT0042
 FACT0043
 FACT0044
 FACT0045
 FACT0046
 FACT0047
 FACT0048
 FACT0049
 FACT0050
 FACT0051
 FACT0052
 FACT0053
 FACT0054
 FACT0055
 FACT0056
 FACT0057
 FACT0058
 FACT0059
 FACT0060
 FACT0061
 FACT0062
 FACT0063
 FACT0064
 FACT0065
 FACT0066
 FACT0067
 FACT0068
 FACT0069
 FACT0070
 FACT0071
 FACT0072

```

C LNZE IS A SPECIAL CASE - NU SUM REQUIRED
  IF (MCOL.EQ. M) GO TO 7
C START SUM FROM GREATEST OF MRUW OR ROW 'MCGL' LN2 COL NOS
  INIT = M
  N = ILN2M1+MCUL
  IF (INTGR(N) .GT. M) INIT = INTGR(N)
C NO SUM IF ROW 'MCUL' HAS LEADING ZERCS UP TO THE DIAGONAL
  IF (INIT.EQ. MCOL) GO TO 7
C ACCUMULATE THE SUM
  LAST = MCOL-1
  DO 6 J = INIT, LAST
    KADR = MM+J
    KADR = NN+J
    JJ = IKUUM1+J
    JJ = INTGR(JJ)+J
    6 SUM = SUM+REAL(JJ)*REAL(KADR)*REAL(KADR)
C BRANCH TO SPECIAL ALGORITHM FOR DIAGONAL ENTRIES
  7 IF (MCOL.EQ. MRUW) GO TO 8
C FOR OFF-DIAGONAL ENTRIES:
  KADR = MM+MCUL
  NN = NN+MCUL
  REAL(KADR) = (REAL(KADR)-SUM)/REAL(NN)
  GO TO 12
C DIAGONAL ENTRY - TEST FOR SINGULARITY AND SEMI-DEFINITENESS
  8 MM = MM+MRUW
  IF (REAL(MM)-SUM .NE. 0.) GO TO 9
  JRJW = MRUW
  GO TO 14
  9 IF (REAL(MM)-SUM .GT. 0.) GO TO 10
  WRITE (KW,9)2) MRCW
  MPD = 1
C CALCULATE ROUNDING ERROR
  10 TESTR = ABS((REAL(MM)-SUM)/REAL(MM))
  IF (TESTR .GE. TEST) GO TO 11
  TEST = TESTR
  IKUW = MRUW

```

FACT0073
 FACT0074
 FACT0075
 FACT0076
 FACT0077
 FACT0078
 FACT0079
 FACT0080
 FACT0081
 FACT0082
 FACT0083
 FACT0084
 FACT0085
 FACT0086
 FACT0087
 FACT0088
 FACT0089
 FACT0090
 FACT0091
 FACT0092
 FACT0093
 FACT0094
 FACT0095
 FACT0096
 FACT0097
 FACT0098
 FACT0099
 FACT0100
 FACT0101
 FACT0102
 FACT0103
 FACT0104
 FACT0105
 FACT0106
 FACT0107
 FACT0108

FACT0109
 FACT0110
 FACT0111
 FACT0112
 FACT0113
 FACT0114
 FACT0115
 FACT0116
 FACT0117
 FACT0118
 FACT0119
 FACT0120
 FACT0121
 FACT0122
 FACT0123
 FACT0124
 FACT0125
 FACT0126

```

C EVALUATE DIAGONAL ENTRY
 11 REAL(MM) = REAL(MM)-SUM
 12 CONTINUE
 13 CONTINUE
C SEMI-DEFINITENESS CHECKS AND ROUNDING ERROR OUTPUT
  IF (MPD .EQ. 0) WRITE (KW,903)
  IF (MPD .EQ. 1 .AND. NPD .EQ. 1) WRITE (KW,905)
  IF (MPD .EQ. 1 .AND. NPD .EQ. 0) WRITE (KW,904)
  IERR = -1.00001*ALOG10(ABS(MPD))
  WRITE (KW,907) IROW,IERR
  IF (IERR .GT. 5) WRITE (KW,908)
  IF (MPD .EQ. 1 .AND. NPD .EQ. 1) STOP
  IF (IERR .GT. 5) STOP
  RETURN
C SINGULAR MATRIX
 14 WRITE (KW,906) JROW
  STOP
  END

```

```

SUBROUTINE URK(LENGTH)
C*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C*****
C THIS SUBROUTINE CREATES
C
C 1) THE LN2 VECTOR WHICH HOLDS THE COLUMN NUMBER
C    OF THE LEADING NON-ZERO ENTRY IN EACH ROW OF
C    THE ASSEMBLED "K" MATRIX
C
C 2) THE ADDRESS COUNT VECTOR WHICH HOLDS THE ABSOLUTE
C    ADDRESS OF THE DIAGONAL ENTRY FOR EACH ROW OF THE
C    ASSEMBLED K MATRIX, MINUS THE ROW NUMBER, AND
C    CHECKS THAT K FITS IN THE /DATA/ VECTOR
C
C    DIMENSION REAL(10000),INTGR(10000)
C    COMMON /IO/ KR, KW, KP, KT1, KT2, KT3
C    COMMON /SIZE/ NET, NDT
C    COMMON /BEGIN/ ICCN,IKUNT,ILNZ,IMASTR,IQ,IK
C    COMMON /END/ LCON,LKUUNT,LLNZ,LMASTR,LQ,LK
C    COMMON /DATA/ REAL
C
C VERSION 1 RELEASE 1 AUGUST 1972
C
C EQUIVALENCE (REAL(1),INTGR(1))
C PRINT CUNT,KJL
100 FORMAT(49H)THE LENGTH OF THE "DATA" VECTOR FOR THIS CASE IS,110,140RK 0001
1H WHICH EXCEEDS,110,49H =THE MAXIMUM ALLOWED IN THE DIMENSION STATORK 0002
2EMENT./39H EXECUTION TERMINATED IN SUBROUTINE ORK 0003
200 FORMAT(5X,3HROW,2X,13HLNZE CCL. NO.,2X,18HABS. ADR. OF DIAG.) ORK 0004
300 FORMAT(19,110,112) ORK 0005
400 FORMAT(10H)THERE ARE,11J,68H NON-ZERO ENTRIES IN "K". IF "K" WERE ORK 0006
IFULLY POPULATED THERE WOULD BE,110, 9H ENTRIES.,/,20X,15HTHE DENSIORK 0007
2TY IS ,E15.6) ORK 0008
ILNZM1=ILNZ-1 ORK 0009
IMSTM1=IMASTR-1 ORK 0010
IKUUM1=IKUUNT-1 ORK 0011
NETM1=NET-1 ORK 0012
OR 0013
OR 0014
OR 0015
OR 0016
OR 0017
OR 0018
OR 0019
OR 0020
CRK 0021
OR 0022
OR 0023
OR 0024
OR 0025
OR 0026
OR 0027
OR 0028
OR 0029
OR 0030
OR 0031
OR 0032
OR 0033
OR 0034
OR 0035
OR 0036

```

```

C SET EACH LN2 COLUMN NC = RJW NU (DIAGONAL MATRIX)
DO 30 IROW=1,NDT
  MSUB=ILNZM1+IRUW
  3) INTGR(MSUB)=IRUW
C EXAMINE MASTER ASSEMBLY LIST, ONE ELEMENT AT A TIME, TO CREATE
C THE LN2 VECTOR
DO 20 LNUM=1,NET
  MADDR = IMSTM1+LNUM
  MADDR = INTGR(MADDR)-1
C CALCULATE NU. DUF IN THE ELEMENT BY DIFFERENCING POINTERS, OR ...
  I = IMASTR+LNUM
  IF(LNUM.EQ.NET) GO TO 3
  NDE = INTGR(I)-INTGR(I-1)
  GO TO 4
C ... BEGIN BY ASSUMING THE LIST IS FILLED, FOR LAST ELEMENT
  3 NDE = LMASTR-INTGR(I-1)+1
C INITIALIZE SMALLEST DOF NO. AT LARGEST POSSIBLE VALUE
  4 ISMALL=NDT
C FIND SMALLEST MASTR NUMBER FOR THIS ELEMENT
  DO 5 JDUF=1,NDE
    INDEX=MADDR+JDUF
    IF(INTGR(INDEX).GT.ISMALL) GO TO 5
C DISCONTINUE SEARCH IF A ZERO IS FOUND, INDICATING EXCESS STORAGE
C AND PREMATURE END OF LIST FOR LAST ELEMENT
    IF(INTGR(INDEX).EQ.0) GO TO 6
    ISMALL=INTGR(INDEX)
  5 CONTINUE
C FIND COLUMN NUMBER OF LEADING NON-ZERO ENTRY IN ROW
  6 DO 10 JDUF=1,NDE
    INDEX=MADDR+JDUF
    INDEX=ILNZM1+INTGR(INDEX)
C CHANGE LN2 COL NU ONLY IF NEW ONE IS LESS THAN OLD ONE
    IF(INTGR(INDEX).LT.ISMALL) GO TO 8
    INTGR(INDEX)=ISMALL
  10 GO TO 10

```

```

DRK 0037
DRK 0038
DRK 0039
DRK 0040
DRK 0041
DRK 0042
DRK 0043
DRK 0044
DRK 0045
DRK 0046
DRK 0047
DRK 0048
DRK 0049
DRK 0050
DRK 0051
DRK 0052
DRK 0053
DRK 0054
DRK 0055
DRK 0056
DRK 0057
DRK 0058
DRK 0059
DRK 0060
DRK 0061
DRK 0062
DRK 0063
DRK 0064
DRK 0065
DRK 0066
DRK 0067
DRK 0068
DRK 0069
DRK 0070
DRK 0071
DRK 0072

```

```

C DISCONTINUE OPERATION IF EXCESS STORAGE IS DISCOVERED
8 IF (INTGR(INDEX).EQ.0) GO TO 20
10 CONTINUE
20 CONTINUE
C CREATE ADDRESS COUNT VECTOR
  INTGR(LKOUNT) = IK
  INDEX=LKOUNT
  DO 40 IRCW=2,NDT
    I = ILNZM1+IRCW
    INTGR(INDEX+1) = INTGR(INDEX)+IRCW+1-INTGR(I)
    INDEX=INDEX+1
40
C ADDRESS COUNT VECTOR NOW CONTAINS ABSOLUTE ADDRESS ONLY FOR THE
C DIAGONAL ENTRIES, AND THUS INTGR(LKOUNT) = LK EXACTLY
  IF (INTGR(LKOUNT) .LE. LENGTH) GO TO 50
  WRITE (KW,100) INTGR(LKOUNT), LENGTH
  STOP
50 LK = INTGR(LKOUNT)
  WRITE(KW,200)
  DO 60 IRCW=1,NDT
    I = ILNZM1+IRCW
    J = IKUUM1+IRCW
    WRITE (KW,300) IRCW, INTGR(I), INTGR(J)
    C REPLACE THE ABS. ADDRESS OF DIAG. BY (ABS. ADDRESS - ROW NO.)
    INTGR(J) = INTGR(J)-IRCW
    CONTINUE
60 NENTRY = INTGR(LKOUNT)+NDT-IK+1
  INDEX = (NDT*(NDT+1))/2
  DENS = FLOAT(NENTRY)/FLOAT(INDEX)
  WRITE (KW,400) NENTRY, INDEX, DENS
  C ZERO THE FORCE/DISPLACEMENT VECTOR AND THE K MATRIX BLOCK
  DO 70 I=1Q,LK
    REAL(I)=0.
70 RETURN
END

```

ORK 0073
 ORK 0074
 ORK 0075
 ORK 0076
 ORK 0077
 ORK 0078
 ORK 0079
 ORK 0080
 ORK 0081
 ORK 0082
 ORK 0083
 ORK 0084
 ORK 0085
 ORK 0086
 ORK 0087
 ORK 0088
 ORK 0089
 ORK 0090
 ORK 0091
 ORK 0092
 ORK 0093
 ORK 0094
 ORK 0095
 ORK 0096
 ORK 0097
 ORK 0098
 ORK 0099
 ORK 0100
 ORK 0101
 ORK 0102
 ORK 0103
 ORK 0104
 ORK 0105
 ORK 0106

```

SUBROUTINE ROTATE(NODE, IROW, JROW, KRCW, ZANGLE, YANGLE, XANGLE)
*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
*****
      DIMENSION REAL(10000), INGR(10000)
      DIMENSION A(3,3), B(3,3), IR(3), INFC(10), KDR(3), PERM(3,3),
2 PROTUR(3,3), ROTOR(3,3), IEMPC(3), TEMPR(3), TEMPRR(3), TEMPC(3),
COMMON /I0/ KK, KW, KP, K11, K12, K13
COMMON /SIZE/ NET, NDT
COMMON /BEGIN/ ICON, IKOUNT, ILN2, IMASTR, IQ, IK
COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
C
C
      EQUIVALENCE (REAL(1), INGR(1))
C PRINT CONTROL
901 FORMAT(1H0,41X,26HROTATION REQUESTED AT NODE,110,/,24X,29HASSOCIATED
1ED MASTER NUMBERS FOR,8X,5HFIRST,6X,6HSECOND,7X,5HTHIRD,3X,3HDOF,/ROTA0020
2,54X,312X,110),/,27X,27HEULER ANGLES IN DEGREES ARE,8X,7HTHETA Z,5ROTA0021
3X,7HTHETA Y,5X,7HTHETA X,/,57X,312X,110.3))
902 FORMAT (1X,79HHALF TRANSFORM (K)*{TFETA TRANSPOSE) WAS APPLIED TO ROTA0023
1THE FOLLOWING DOF BELOW ROW,110,16H IN THE K MATRIX)
903 FORMAT (1J(2X,110))
904 FORMAT(103HABOVE REQUEST INCONSISTENT DUE TO INEQUALITY OF LEADINROTA0026
1G NON-ZERO ENTRY COLUMN NUMBERS OF ARGUMENT ROWS,/,9X,3HROW,1X,11HROTA0027
2LNZE CUL NJ,/,312X,110,2X,110,/)
905 FORMAT (1X,41HEXECUTION TERMINATED IN SUBROUTINE ROTATE)
906 FORMAT (54HABOVE REQUEST INCONSISTENT DUE TO REPEATED ROW NUMBER)ROTA0030
907 FORMAT(173HABOVE REQUEST INVALID DUE TO INCORRECT LOCATION OF UNDERROTA0031
1FINED ROW NUMBERS)
      IR(1)=IROW
      IR(2)=JKOW
      IR(3)=KKOW
C PRINT ENTRY MESSAGE

```

```

NNODE=IABS(NNODE)
WRITE (KW,901) NNODE, (IR(1), I = 1,3), ZANGLE, YANGLE, XANGLE
C ESTABLISH DIFF ROTATION LIMIT
LIMIT = 3
IF (KROW .LE. J) LIMIT = 2
IF (KROW .GT. 0 .AND. JROW .GT. 0) GO TO 100
WRITE(KW,907)
WRITE(KW,905)
STOP
C GET LNZE COL NOS AND TEST FOR EQUALITY
100 DO 1 I = 1,LIMIT
J = ILN2+IR(I)-1
1 INFO(I) = INTGR(J)
IF (LIMIT .EQ. 3 .AND. INFO(1) .EQ. INFO(2) .AND. INFO(2) .EQ.
2 INFO(3)) GO TO 2
IF (LIMIT .EQ. 2 .AND. INFO(1) .EQ. INFO(2)) GO TO 2
C SOMETHING WRONG - PRINT ERROR MESSAGE
WRITE (KW,904) (IR(I), INFO(I), I = 1,LIMIT)
WRITE (KW,902)
STOP
C CHECK FOR KEYPUNCH ERROR IN ARGUMENT LIST
2 IF (KROW .NE. JROW .AND. JROW .NE. KROW .AND. KROW .NE. IROW)
2 GO TO 3
WRITE (KW,906)
WRITE (KW,905)
STOP
C OK TO PROCEED - FORM ROTOR MATRIX AFTER CONVERTING
C ANGLES TO RADIAN MEASURE
3 Z = 3.14159*ZANGLE/180.
Y = 3.14159*YANGLE/180.
X = 3.14159*XANGLE/180.
ROTUR(1,1) = COS(Y)*COS(Z)
ROTUR(1,2) = COS(Y)*SIN(Z)
ROTUR(1,3) = SIN(Y)
ROTUR(2,2) = COS(X)*COS(Z) - SIN(X)*SIN(Y)*SIN(Z)
ROTUR(2,3) = SIN(X)*COS(Y)

```

ROTA0037
 ROTA0038
 ROTA0039
 ROTA0040
 ROTA0041
 ROTA0042
 ROTA0043
 ROTA0044
 ROTA0045
 ROTA0046
 ROTA0047
 ROTA0048
 ROTA0049
 ROTA0050
 ROTA0051
 ROTA0052
 ROTA0053
 ROTA0054
 ROTA0055
 ROTA0056
 ROTA0057
 ROTA0058
 ROTA0059
 ROTA0060
 ROTA0061
 ROTA0062
 ROTA0063
 ROTA0064
 ROTA0065
 ROTA0066
 ROTA0067
 ROTA0068
 ROTA0069
 ROTA0070
 ROTA0071
 ROTA0072

ROTA0073
 ROTA0074
 ROTA0075
 ROTA0076
 ROTA0077
 ROTA0078
 ROTA0079
 ROTA0080
 ROTA0081
 ROTA0082
 ROTA0083
 ROTA0084
 ROTA0085
 ROTA0086
 ROTA0087
 ROTA0088
 ROTA0089
 ROTA0090
 ROTA0091
 ROTA0092
 ROTA0093
 ROTA0094
 ROTA0095
 ROTA0096
 ROTA0097
 ROTA0098
 ROTA0099
 ROTA0100
 ROTA0101
 ROTA0102
 ROTA0103
 ROTA0104
 ROTA0105
 ROTA0106
 ROTA0107
 ROTA0108

```

    ROTUR(3,3) = COS(X)*COS(Y)
    ROTUR(2,1) = -KOTOR(1,2)
    ROTUR(3,1) = -KOTOR(1,3)
    ROTUR(3,2) = -KOTOR(2,3)
    C SKIP K MATRIX SECTION IF A RE-ROTATION
    IF(NUDE .LT. 0) GO TO 38
    C ESTABLISH LEAST AND LARGST DJF NUMBERS FROM ARGUMENT LIST
    LEAST = IROW
    LARGST = JROW
    DO 4 I = 1,LIMIT
      IF (IR(I) .LT. LEAST) LEAST = IR(I)
      IF (IR(I) .GT. LARGST) LARGST = IR(I)
    4 CONTINUE
    C APPLY (ROTUR)*(K) TO ROWS IROW, JROW, KROW - ONE COLUMN AT A TIME
    C FROM LNZE TO LEAST-1 -- SKIP SECTION IF LEAST ROW = 1ST ROW
    INIT = INFO(1)
    IF (LEAST .EQ. 1 .OR. INIT .EQ. LEAST) GO TO 8
    LAST = LEAST-1
    DO 7 MCUL = INIT,LAST
      C GET ENTRIES OF K MATRIX INTO TEMPR, ZERC TEMPC
      DO 5 I = 1,LIMIT
        J = IKUUNT+IR(I)-1
        KADR = INTGR(J)+MCOL
        TEMPR(I) = REAL(KADR)
      5 TEMPC(I) = 0.
      C APPLY TRANSFORM
      DO 6 I = 1,LIMIT
        DO 6 J = 1,LIMIT
          6 TEMPL(I) = TEMPC(I)+ROTUR(I,J)*TEMPR(J)
      C RESTORE TRANSFORMED COLUMN TO K MATRIX
      DO 7 I = 1,LIMIT
        J = IKUUNT+IR(I)-1
        KADR = INTGR(J)+MCOL
        7 REAL(KADR) = TEMPC(I)
      C TEST FOR ROWS INTERVENING BETWEEN IROW, JROW AND KROW
      8 IF (LARGST .EQ. LEAST+LIMIT-1) GC TC 26
  
```

```

*****ROTA0109
C SPECIAL ALGORITHMS FOR INTERVENING ROWS ROTA0110
*****ROTA0111
C PRINT HEADING ROTA0112
      WRITE (KM,902) LEAST ROTA0113
C FIND VALUE OF MIDDLE ARGUMENT ROW IF 3 DOF ARE BEING ROTATED ROTA0114
      IF (LIMIT .EQ. 2) GO TO 10 ROTA0115
      DO 9 I = 1,3 ROTA0116
        IF (IR(I) .NE. LEAST .AND. IR(I) .NE. LARGST) MIDDLE = IR(I) ROTA0117
      9 CONTINUE ROTA0118
C TEST TO SEE IF ARGUMENT ROWS WERE GIVEN IN ASCENDING ORDER. IF ROTA0119
C NOT, A PERMUTATION SIMILARITY TRANSFORM MUST BE APPLIED TO THE ROTA0120
C ROTOR MATRIX TO TRANSFORM THE INTERVENING ROWS PROPERLY ROTA0121
      IF (IROW .LT. JROW .AND. JROW .LT. KROW) GO TO 15 ROTA0122
      10 IF (KROW .EQ. 0 .AND. IROW .LT. JROW) GO TO 15 ROTA0123
C PREPARE STORAGE ROTA0124
      DO 11 I = 1,LIMIT ROTA0125
      DO 11 J = 1,LIMIT ROTA0126
        A(I,J) = 0. ROTA0127
      11 PERM(I,J) = 0. ROTA0128
      DO 12 I = 1,LIMIT ROTA0129
        IF (IR(I) .EQ. LEAST) PERM(1,I) = 1. ROTA0130
        IF (IR(I) .EQ. LARGST) PERM(LIMIT,I) = 1. ROTA0131
        IF (LIMIT .EQ. 2) GO TO 12 ROTA0132
        IF (IR(I) .EQ. MIDDLE) PERM(2,I) = 1. ROTA0133
      12 CONTINUE ROTA0134
C APPLY SIMILARITY TRANSFORM ROTA0135
      DO 13 I = 1,LIMIT ROTA0136
      DO 13 J = 1,LIMIT ROTA0137
      DO 13 K = 1,LIMIT ROTA0138
        13 A(I,J) = A(I,J)+PERM(I,K)*ROTOR(K,J) ROTA0139
      DO 14 I = 1,LIMIT ROTA0140
      DO 14 J = 1,LIMIT ROTA0141
      DO 14 K = 1,LIMIT ROTA0142
        14 PROTOK(I,J) = ROTOK(I,J)+A(I,K)*PERM(K,J) ROTA0143
      ROTA0144

```

```

      GO TO 17
      C PERMUTATION NOT REQUIRED
      15 DO 16 I = 1,3
      DO 16 J = 1,3
      16 PROTOK(I,J) = KOTOR(I,J)
      C INITIALIZE INFO VECTOR FILL INDEX AND ESTABLISH LOOP LIMITS FOR
      C INTERVENING ROWS
      17 INFOX = 0
      INIT = LEAST+1
      C LAST ROW TO BE DONE DEPENDS ON WHETHER 2 OR 3 DOF ARE BEING ROTATED
      IF (LIMIT.EQ. 2) LAST = LARGST-1
      IF (LIMIT.EQ. 3) LAST = MIDDLE-1
      C LOOP OVER REMAINING COLUMNS, (PROTOR)*K(K), AND FIRST ROWS,
      C K(K)*(PROTOR-I), FROM INIT TO LAST -- SKIP IF MIDDLE DOF = LEAST+1
      C WHEN 3 DOF ARE BEING ROTATED
      IF (LIMIT.EQ. 3 .AND. LAST.EQ. LEAST) GO TO 25
      C THIS IS THE RE-ENTRY POINT IF 3 DOF ARE BEING ROTATED AND ROWS
      C INTERVENE BETWEEN MIDDLE AND LARGST
      18 DO 24 KROW = INIT, LAST
      C CHECK LNZE COL NO OF CURRENT ROW -- IF .GT. LOWER ROTATED DOF,
      C WHICH = INIT-1, K MATRIX CONTAINS STORED OR NON-STORED ZEROS AND
      C BOTH ROW AND COLUMN TRANSFORM CAN BE SKIPPED
      J = ILNZE+KROW-1
      IF (INTGR(J).GT. INIT-1) GO TO 24
      C INCREMENT INFO FILL INDEX AND STORE CURRENT ROW NUMBER
      INFOX = INFOX+1
      INFO(INFJX) = KROW
      C IF TEN ROW NUMBERS HAVE BEEN COLLECTED, PRINT THEM AND RESET INDEX
      IF (INFOX.LT. 10) GO TO 19
      WRITE (KW,903) (INFO(J), J = 1,10)
      INFOX = 0
      C GET CORRECT ABSOLUTE ADDRESSES FOR K MATRIX ENTRIES, REPLACING COLUMN
      C SECTORS ABOVE LOWER TRIANGLE BY ROW SECTORS BELOW, AND VICE-VERSA
      19 J = IKOUNT+KROW-1
      KDR(I) = INTGR(J)
      J = IKOUNT+LAST

```

ROTA0145
 ROTA0146
 ROTA0147
 ROTA0148
 ROTA0149
 ROTA0150
 ROTA0151
 ROTA0152
 ROTA0153
 ROTA0154
 ROTA0155
 ROTA0156
 ROTA0157
 ROTA0158
 ROTA0159
 ROTA0160
 ROTA0161
 ROTA0162
 ROTA0163
 ROTA0164
 ROTA0165
 ROTA0166
 ROTA0167
 ROTA0168
 ROTA0169
 ROTA0170
 ROTA0171
 ROTA0172
 ROTA0173
 ROTA0174
 ROTA0175
 ROTA0176
 ROTA0177
 ROTA0178
 ROTA0179
 ROTA0180

ROTA0181
 ROTA0182
 ROTA0183
 ROTA0184
 ROTA0185
 ROTA0186
 ROTA0187
 ROTA0188
 ROTA0189
 ROTA0190
 ROTA0191
 ROTA0192
 ROTA0193
 ROTA0194
 ROTA0195
 ROTA0196
 ROTA0197
 ROTA0198
 ROTA0199
 ROTA0200
 ROTA0201
 ROTA0202
 ROTA0203
 ROTA0204
 ROTA0205
 ROTA0206
 ROTA0207
 ROTA0208
 ROTA0209
 ROTA0210
 ROTA0211
 ROTA0212
 ROTA0213
 ROTA0214
 ROTA0215
 ROTA0216

```

    KOR(2) = INTGR(J)+MROW
    IF (INIT .EQ. LEAST+1) GO TO 20
    KDR(2) = KDR(1)+INIT-1
    20 KDR(1) = KDR(1)+LEAST
    IF (LIMIT .EQ. 2) GO TO 21
    J = IKUUNT+LARGST-1
    KDR(3) = INTGR(J)+MROW
    C PICK UP K MATRIX ENTRIES IN TEMPC, TEMPR AND ZERO TEMPCC, TEMPRR
    21 DO 22 I = 1,LIMIT
      KADR = KDR(1)
      TEMPCC(1) = REAL(KADR)
      TEMPR(1) = REAL(KADR)
      TEMPCC(1) = 0.
    22 TEMPRR(1) = 0.
    C APPLY TRANSFORMS (PROTOR)*(TEMPC) AND (TEMPR)*(PROTOR-TRANSPOSE)
    DO 23 I = 1,LIMIT
      DO 23 J = 1,LIMIT
        TEMPCC(1) = TEMPCC(1)+PROTOR(I,J)*TEMPC(J)
        TEMPRR(1) = TEMPRR(1)+TEMPR(J)*PROTCR(I,J)
    23 KESTORE TRANSFORMED VALUES TO K MATRIX
    KADR = KDR(1)
    REAL(KADR) = TEMPCC(1)
    KADR = KDR(2)
    IF (INIT .EQ. LEAST+1) REAL(KADR) = TEMPRR(2)
    IF (INIT .NE. LEAST+1) REAL(KADR) = TEMPCC(2)
    IF (LIMIT .EQ. 2) GO TO 24
    KADR = KDR(3)
    REAL(KADR) = TEMPRR(3)
    24 CONTINUE
    C PRINT PARTIALLY FILLED INFO VECTOR IF THERE IS AT LEAST ONE ROW,
    C RESET INDEX
    IF (INFOX .EQ. 0) GO TO 25
    WRITE (KM,903) (INFO(I), I = 1,INFOX)
    INFOX = 0
    C COMPLETION TEST
    25 IF (LIMIT .EQ. 2 .OR. INIT .NE. LEAST+1) GO TO 26
  
```

```

C CHECK FOR INTERVENING ROWS BETWEEN MIDDLE AND LARGEST
  IF (MIDDLE+1 .EQ. LARGST) GO TO 26
C RESET LCCP LIMITS, PRINT NEW HEADING
  INIT = MIDDLE+1
  LAST = LARGST-1
  WRITE (KH,902) MIDDLE
  GC TC 16
C*****
C END OF SPECIAL ALGORITHM SECTION
C*****
C FOR 2X2 CR 3X3 K MATRIX ENTRIES DIRECTLY ASSOCIATED WITH CCF BEING
C ROTATED, APPLY FULL SIMILARITY TRANSFORM
  26 DC 28 I = 1,LIMIT
  DO 28 J = 1,I
    K = IKOUNT+IR(I)-1
    KACR = INTGR(K)+IR(J)
    IF (IR(I) .GT. IR(J)) GO TO 27
    K = IKOUNT+IR(J)-1
    KACR = INTGR(K)+IR(I)
  C STORE K MATRIX ENTRY IN A, ZERO B AND SYMMETRIZE
  27 A(I,J) = REAL(KACR)
    B(I,J) = 0.
    A(J,I) = A(I,J)
  28 B(J,I) = 0.
C APPLY HALF CF TRANSFORM, (KUTUR)*(A)
  DC 29 I = 1,LIMIT
  DO 29 J = 1,LIMIT
    DC 29 K = 1,LIMIT
    29 E(I,J) = B(I,J)+ROTUR(I,K)*A(K,J)
  C PREPARE STORAGE
  DC 30 I = 1,LIMIT
  DO 30 J = 1,LIMIT
    30 A(I,J) = 0.
  C APPLY 2ND HALF CF TRANSFORM, (B)*(ROTOR-TRANSPOSE)
  DC 31 I = 1,LIMIT
  DO 31 J = 1,LIMIT

```

```

DO 31 K = 1,LIMIT
  31 A(I,J) = A(I,J)+B(I,K)*RUTOR(J,K)
  C RESTORE TRANSFORMED VALUES TO K MATRIX
DO 32 I = 1,LIMIT
  DO 32 J = 1,I
    K = IKUUNT+IR(I)-1
    KADR = INTGR(K)+IR(J)
    IF (IR(I) .GE. IR(J)) GO TO 32
    K = IKUUNT+IR(J)-1
    KADR = INTGR(K)+IR(I)
  32 REAL(KADR) = A(I,J)
  C APPLY (K)*(KUTOR-TRANPOSE) TO ROWS BELCW LARGST IF LNZE COL NOS
  C ARE .LE. LEAST. (IF LNZE COL NO IS .GT. LEAST, THE ROW CONTAINS
  C STORED OR NJN-STURED ZEROS IN CCLUMAS IROW, JROW, KROW AND CAN
  C BE SKIPPED.) --- SKIP THIS SECTION IF LARGST DOF = NDT
  IF (LARGST .EQ. NDT) GO TO 38
  C PRINT NEW HEADING, RESET LOOP LOWER LIMIT AND INFO INDEX
  WRITE (KW,902) LARGST
  INIT = LARGST+1
  INFOX = 0
DO 37 MROW = INIT,NDT
  C CHECK LNZE COL NO
  J = ILNZ+MROW-1
  IF (INTGR(J) .GT. LEAST) GO TO 37
  C INCREMENT INFO INDEX, STORE ROW NUMBER
  INFOX = INFOX+1
  INFO(INFOX) = MROW
  C IF INFO VECTOR IS FILLED, PRINT AND RESET INDEX
  IF (INFOX .LT. 10) GO TO 33
  WRITE (KW,903) (INFO(I), I = 1,10)
  INFOX = 0
  C GET ENTRIES OF K MATRIX INTO TEMPC, ZERC TEMPR
  33 K = IKUUNT+MROW-1
  K = INTGR(K)
  DO 34 I = 1,LIMIT
    KADR = K+IR(I)

```

ROTA0253
 ROTA0254
 ROTA0255
 ROTA0256
 ROTA0257
 ROTA0258
 ROTA0259
 ROTA0260
 ROTA0261
 ROTA0262
 ROTA0263
 ROTA0264
 ROTA0265
 ROTA0266
 ROTA0267
 ROTA0268
 ROTA0269
 ROTA0270
 ROTA0271
 ROTA0272
 ROTA0273
 ROTA0274
 ROTA0275
 ROTA0276
 ROTA0277
 ROTA0278
 ROTA0279
 ROTA0280
 ROTA0281
 ROTA0282
 ROTA0283
 ROTA0284
 ROTA0285
 ROTA0286
 ROTA0287
 ROTA0288

ROTA0289
 ROTA0290
 ROTA0291
 ROTA0292
 ROTA0293
 ROTA0294
 ROTA0295
 ROTA0296
 ROTA0297
 ROTA0298
 ROTA0299
 ROTA0300
 ROTA0301
 ROTA0302
 ROTA0303
 ROTA0304
 ROTA0305
 ROTA0306
 ROTA0307
 ROTA0308
 ROTA0309
 ROTA0310
 ROTA0311
 ROTA0312
 ROTA0313
 ROTA0314
 ROTA0315
 ROTA0316

```

    TEMPC(I) = REAL(KADR)
  34 TEMPK(I) = 0.
  C APPLY TRANSFORM
    DO 35 I = 1,LIMIT
      DO 35 J = 1,LIMIT
        35 TEMPR(I) = TEMPR(I)+TEMPC(J)*ROTOR(I,J)
  C RESTORE TO K
    DO 36 I = 1,LIMIT
      KADR = K+IR(I)
      36 REAL(KADR) = TEMPR(I)
  37 CONTINUE
  C CHECK FOR PARTIALLY FILLED INFO VECTOR
    IF (INFOX .EQ. 0) GO TO 38
    WRITE (KW,903) (INFO(I), I = 1,INFOX)
  C APPLY TRANSFORM (ROTOR)*(Q) TO ASSEMBLED ELEMENT EQUIVALENT
  C NOVAL FORCES
    DO 39 I = 1,LIMIT
      J = IQ+IR(I)-1
      TEMPR(I) = REAL(J)
    39 TEMPC(I) = 0.
    DO 40 I = 1,LIMIT
      DO 40 J = 1,LIMIT
        40 TEMPC(I) = TEMPC(I)+ROTOR(I,J)*TEMPR(J)
      J = IQ+IR(I)-1
    41 REAL(J) = TEMPC(I)
    RETURN
  END
  
```

```

SUBROUTINE SETUP(LENGTH,NCON,MASTRL)
*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AERUELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
*****
DIMENSION REAL(10000),INTGR(10000)
DIMENSION I1(6), LI(6), KD(6)
COMMON /IO/ KR, KW, KP, K1, K12, K13
COMMON /SIZE/ NET, NDT
COMMON /BEGIN/ ICON,IKOUNT,ILNZ,IMASTR,IQ,IK
COMMON /END/ LCON,LKOUNT,LLNZ,LMASTR,LQ,LK
COMMON /DATA/ REAL
C VERSION 1 RELEASE 1 AUGUST 1972
EQUIVALENCE (REAL(1), INTGR(1))
EQUIVALENCE (ICON,I1(1)), (LCON,LI(1)), (KR,KD(1))
C PRINT CONTROL
901 FORMAT (1H0,53X,11HSETUP ENTRY,/,42X,35HUSER SPECIFICATIONS TO FEASETU0018
1BL SYSTEM,/,1H0,51X,16HI/O DEVICE CODES,/,27X,6HREADER,5X,7HPRINTESEIU0019
2R,2X,10HCARD PUNCH,7X,5HTAPE1,7X,5HTAPE2,7X,5HTAPE3,/,21X,6(2X,110SETU0020
3),/,1H0,53X,12HPROBLEM SIZE,/,42X,25HTOTAL NUMBER OF ELEMENTS=,110SETU0021
4,/,42X,25HTOTAL DEGREES OF FREEDOM=,110,/,1H0,51X,16HENTRY PARAMETSETU0022
5ERS,/,39X,32HASSUMED LENGTH OF /DATA/ VECTOR=,110,/,32X,45HNUMBER SETU0023
6UF DISPLACEMENT CONSTRAINTS REQUESTED=,110,/,33X,44HNUMBER OF WORDSETU0024
7S REQUESTED FOR ASSEMBLY LIST=,110)
902 FORMAT (1HG,43X,31H/DATA/ VECTOR ADDRESS INDEX MAP,/,22X,11HCONSTRSETU0026
1AINTS,2X,10HDG ABS ADR,1X,11HLNZE COL NO,1X,11HASMBLY LIST,2X,10HQSETU0027
2/U VECTOR,4X,8HK MATRIX,/,16X,5HBEGIN,6(2X,110),/,18X,3HEND,5(2X,1SETU0028
310),11X,1H,/,1H0,36H+ LK IS CALCULATED BY SUBROUTINE ORK)
903 FORMAT (40HSTORAGE EXCEEDS LENGTH OF /DATA/ VECTOR,/,1X,34HLENGTHSETU0030
1 SUGGESTED FOR THIS PROBLEM=,112,6H WORDS,/,1X,40HEXECUTION TERMINSETU0031
2ATED IN SUBROUTINE SETUP)
C *****
C REMOVE THIS FORMAT AND WRITE STATEMENT INDICATED BELOW IF FEABL
C HEADING IS NOT DESIRED
1001 FORMAT(1H1/3(1H ,92(1HX)/),4(5H XXXX,84X,4HXXXXX/),5H XXXX,7X,10(1HSETU0036

```

```

1F),5X,10(1HE),9X,2HAA,9X,8(1HB),7X,3HLLL,14X,4HXXX/5H XXXX,7X,10(SETU00037
21HF),5X,10(1HE),8X,4HAAA,8X,5(1HB),6X,3HLLL,14X,4HXXX/5H XXXX,7XSETU00038
3,3HFFF,12X,3HEEE,14X,6(1HA),7X,3HB8B,3X,4HB8B8,5X,3HLLL,14X,4HXXXSETU00039
4/5H XXXX,7X,3HFFF,12X,3HEEE,13X,3HAAA,2X,3HB8B,4X,3HB8B,4X,3HB8B,5SETU00040
5X,3HLLL,14X,4HXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HAAA,4X,3HAAA,5SETU00041
6X,3HB8B,4X,3HB8B,5X,3HLLL,14X,4HXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12SETU00042
7X,3HAAA,4X,3HAAA,5X,3HB8B,3X,4HB8B8,5X,3HLLL,14X,4HXXX/5H XXXX,7XSETU00043
8,8(1HF),7X,8(1HE),7X,3HAAA,4X,3HAAA,5X,9(1HB),6X,3HLLL,14X,4HXXX/5H XXXX,7XSETU00044
95H XXXX,7X,8(1HF),7X,8(1HE),7X,3HAAA,4X,3HAAA,5X,9(1HB),6X,3HLLL,14X,4HXXX/5H XXXX,7XSETU00045
AXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,10(1HA),5X,3HB8B,3X,4HB8B8,5X,3HSETU00046
BLLL,14X,4HXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HAAA,4X,3HAAA,5X,3HSETU00047
CB8B,4X,3HB8B,5X,3HLLL,14X,4HXXX/5H XXXX,7X,3HFFF,12X,3HEEE,12X,3HSETU00048
DAAA,4X,3HAAA,5X,3HB8B,3X,4HB8B8,5X,3HLLL,14X,4HXXX/5H XXXX,7X,3HSETU00049
EFF,12X,10(1HE),5X,3HAAA,4X,3HAAA,5X,9(1HB),6X,10(1HL),7X,4HXXX/5HSETU00050
F XXXX,7X,3HFFF,12X,10(1HE),5X,3HAAA,4X,3HAAA,5X,8(1HB),7X,10(1HL),SETU00051
G7X,4HXXX/4(5H XXXX,84X,4HXXX/),5H XXXX,24X,37HFINITE ELEMENT ANASETU00052
HLYSIS BASIC LIBRARY,23X,4HXXX/4(5H XXXX,84X,4HXXX/),3(1H,92(1HSETU00053
I/),1H1)
C *****
C PRINT ENTRY MESSAGE
C *****
C REMOVE THIS WRITE STATEMENT IF FEABL. HEADING IS NOT DESIRED
      WRITE (KW,1001)
C *****
C *****
      WRITE (KW,901) (KD(I), I = 1,6), NET, NDT, LENGTH, NCON, MASTRL
C CALCULATE ADDRESS INDEX VALUES
      ICON = 1
      LCON = NCON
      IKOUNT = LCON+1
      LKOUNT = LCON+NDT
      ILNZ = LKOUNT+1
      LLNZ = LKOUNT+NDT
      IMASIR = LLNZ+1
      LMASIR = LLNZ+MASTRL
      IQ = LMASIR+1
      LQ = LMASIR+NDT

```

```

      IK = LQ+1
      C PRINT INDEX MAP
      WRITE (KM,902) (II(I), I = 1,6), (LI(I), I = 1,5)
      C STORAGE BOUNDS TEST
      IF (LMASTR.LE. LENGTH) GO TO 1
      C STORAGE EXCEEDED - ESTIMATE REQUIRED LENGTH BASED ON LOWER TRIANGLE
      C ESTIMATED POPULATION FACTOR
      TR = (NDT*(NDT+1))/2
      DENS = 0.5
      IF (NDT.GT. 200) DENS=0.3
      IF (NDT.GT. 500) DENS=0.2
      IF (NDT.GT. 1500) DENS = 0.15
      IF (NDT.GT. 2000) DENS = 0.10
      TR = TR*DENS
      LENGTH = LQ+TR
      WRITE (KM,903) LENGTH
      STOP
      C ENOUGH STORAGE EXISTS TO GO THRU CRK
      1 DO 2 I = ICUN,LCON
      2 INTR(I) = J
      RETURN
      END

```

```

SETU0073
SETU0074
SETU0075
SETU0076
SETU0077
SETU0078
SETU0079
SETU0080
SETU0081
SETU0082
SETU0083
SETU0084
SETU0085
SETU0086
SETU0087
SETU0088
SETU0089
SETU0090
SETU0091
SETU0092
SETU0093
SETU0094

```

```

SIMU00001
SIMU00002
SIMU00003
SIMU00004
SIMU00005
SIMU00006
SIMU00007
SIMU00008
SIMU00009
SIMU00010
SIMU00011
SIMU00012
SIMU00013
SIMU00014
SIMU00015
SIMU00016
SIMU00017
SIMU00018
SIMU00019
SIMU00020
SIMU00021
SIMU00022
SIMU00023
SIMU00024
SIMU00025
SIMU00026
SIMU00027
SIMU00028
SIMU00029
SIMU00030
SIMU00031
SIMU00032
SIMU00033
SIMU00034
SIMU00035
SIMU00036

SUBROUTINE SIMULQ(ENERGY)
C*****
C FINITE ELEMENT ANALYSIS BASIC LIBRARY SUBROUTINE
C AEROELASTIC AND STRUCTURES RESEARCH LABORATORY
C MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C*****
DIMENSION REAL(10000), INTGR(10000)
COMMON /IO/ KR, KW, KP, KT1, KT2, KT3
COMMON /SIZE/ NET, NDT
COMMON /BEGIN/ ICCN, IKOUNT, ILNZ, IMASTR, IQ, IK
COMMON /END/ LCON, LKOUNT, LILNZ, LMASTR, LQ, LK
COMMON /DATA/ REAL

C VERSION 1 RELEASE 1 AUGUST 1972
C
EQUIVALENCE (REAL(1), INTGR(1))
C PRINT CONTROL
901 FORMAT (1H0,53X,18HSIMULQ ENTRY POINT,/,38H PRESCRIBED FORCE/DISPL
2ACEMENT VECTOR: )
902 FORMAT (30H0DISPLACEMENT SOLUTION VECTOR: )
903 FORMAT (1H0,2X,3HROW,10I12)
904 FORMAT (6H VALUE,10(2X,E10.3))
905 FORMAT (62H0STRAIN ENERGY IN THE STRUCTURE=,E10.3)
IKOUNT1 = IKOUNT-1
ILNZM1 = ILNZ-1
IQM1 = IQ-1
C PRINT ENTRY MESSAGE, SET PRINT SECTION CONTROL FLAG
WRITE (KW,901)
IFLAG = 0
C PRINT SECTION: DIVIDE Q VECTOR INTO GROUPS OF TEN
C PLUS A POSSIBLE REMAINDER
MUST = NDT/10
LEFT = NDT-10*MUST
JBEG = 1
NBEG = IQ
C RE-ENTRY POINT FOR SOLUTION OUTPUT
1 IF (NDT .LI. 10) GO TO 3

```

SIMU00037
SIMU00038
SIMU00039
SIMU00040
SIMU00041
SIMU00042
SIMU00043
SIMU00044
SIMU00045
SIMU00046
SIMU00047
SIMU00048
SIMU00049
SIMU00050
SIMU00051
SIMU00052
SIMU00053
SIMU00054
SIMU00055
SIMU00056
SIMU00057
SIMU00058
SIMU00059
SIMU00060
SIMU00061
SIMU00062
SIMU00063
SIMU00064
SIMU00065
SIMU00066
SIMU00067
SIMU00068
SIMU00069
SIMU00070
SIMU00071
SIMU00072

```

DU 2 I = 1, MDS1
JBEG = 1+10*(I-1)
JEND = JBEG+9
NBEG = IQ+JBEG-1
NEND = NBEG+9
WRITE (KW,903) (J, J = JBEG, JEND)
2 WRITE (KW,904) (REAL(N), N = NBEG, NEND)
C CHECK FOR EXISTENCE OF REMAINDER
IF (LEFT.EQ. J) GO TO 4
JBEG = JEND+1
NBEG = NEND+1
3 WRITE (KW,903) (J, J = JBEG, NDT)
WRITE (KW,904) (REAL(N), N = NBEG, LC)
C CHECK CONTROL FLAG
4 IF (IFLAG.EQ. 0) GO TO 5
WRITE (KW,905) ENERGY
RETURN
C FORWARD SOLUTION - NO DIVISIONS, SO SKIP FIRST ROW ENTIRELY
C INITIALIZE CONSTRAINT VECTOR AT 1ST NONZERO ENTRY
5 DO 6 I = ICUN, LCCN
IF (INTGR(I).EQ. 0) GO TO 6
GO TO 7
6 CONTINUE
7 NEXT = I
IF (INTGR(NEXT).EQ. 1) NEXT = NEXT+1
C SOLVE (A)(R) = (Q)
DO 10 MRJW = 2, NDT
C CHECK FOR CONSTRAINT TO SKIP ROW
IF (MKRW.NE. INTGR(NEXT)) GO TO 8
C UPDATE NEXT
NEXT = NEXT+1
GO TO 10
C INITIALIZE SUM AND LOOP LIMITS FOR SUM
8 SUM = 0.
LAST = MKRW-1
M = ILNZ+LAST

```

```

M = INTR(M)
MM = IKOUNT+LAST
MM = INTR(MM)
DO 9 J = M, LAST
KADR = MM+J
N = IQMI+J
9 SUM = SUM+REAL(KADR)*REAL(N)
C SUBTRACT SUM FROM Q
N = IQMI+MROW
REAL(N) = REAL(N)-SUM
10 CONTINUE
C SOLVE (U)(P) = (K) AND CALCULATE ENERGY
N = IQ-1
ENERGY = J.
DO 11 MROW = 1, NDT
KADR = IKOUNT+MROW
KADR = INTR(KADR)+MROW
N = N+1
REAL(N) = REAL(N)/REAL(KADR)
11 ENERGY = ENERGY+REAL(KADR)*(REAL(N)**2)
ENERGY = U.5*ENERGY
C BACK SOLUTION - NJ DIVISIONS, SO SKIP LAST ROW ENTIRELY
NEXT = LCON
IF (INTR(NEXT) .EQ. NDT) NEXT = NEXT-1
C LOOP BACKWARDS OVER REMAINING ROWS
DO 14 I = 2, NDT
MROW = NDT+1-I
C CHECK FOR CONSTRAINT TO SKIP ROW
IF (MROW .NE. INTR(NEXT)) GO TO 12
C UPDATE NEXT
NEXT = NEXT-1
IF (NEXT .LT. ICON) NEXT = LCON
GO TO 14
C INITIALIZE SUM AND LOWER LOOP LIMIT
12 SUM = J.
INIT = MKOUNT+1

```

```

SIMU00073
SIMU00074
SIMU00075
SIMU00076
SIMU00077
SIMU00078
SIMU00079
SIMU00080
SIMU00081
SIMU00082
SIMU00083
SIMU00084
SIMU00085
SIMU00086
SIMU00087
SIMU00088
SIMU00089
SIMU00090
SIMU00091
SIMU00092
SIMU00093
SIMU00094
SIMU00095
SIMU00096
SIMU00097
SIMU00098
SIMU00099
SIMU0100
SIMU0101
SIMU0102
SIMU0103
SIMU0104
SIMU0105
SIMU0106
SIMU0107
SIMU0108

```

SIMU0109
SIMU0110
SIMU0111
SIMU0112
SIMU0113
SIMU0114
SIMU0115
SIMU0116
SIMU0117
SIMU0118
SIMU0119
SIMU0120
SIMU0121
SIMU0122
SIMU0123
SIMU0124
SIMU0125
SIMU0126
SIMU0127

```

DO 13 J = 1,N1,NDT
C CHECK IF LW2 CUL NU OF ROW J EXCEEDS MRCW - IF SO, SKIP
N = ILNZM1+J
IF (INTGR(N) .GT. MROW) GO TO 13
KADR = IKUJMI+J
KAUK = INTGR(KADR)+MROW
N = IJMI+J
SUM = SUM+REAL(KADR)*REAL(N)
13 CONTINUE
C SUBTRACT SUM FROM W
N = IJMI+MROW
REAL(N) = REAL(N)-SUM
14 CONTINUE
C PRINT SOLUTION HEADING, RESET CONTROL FLAG AND
C BRANCH TO OUTPUT SECTION
WRITE (KW,9J2)
IFLAG = 1
GO TO 1
END

```