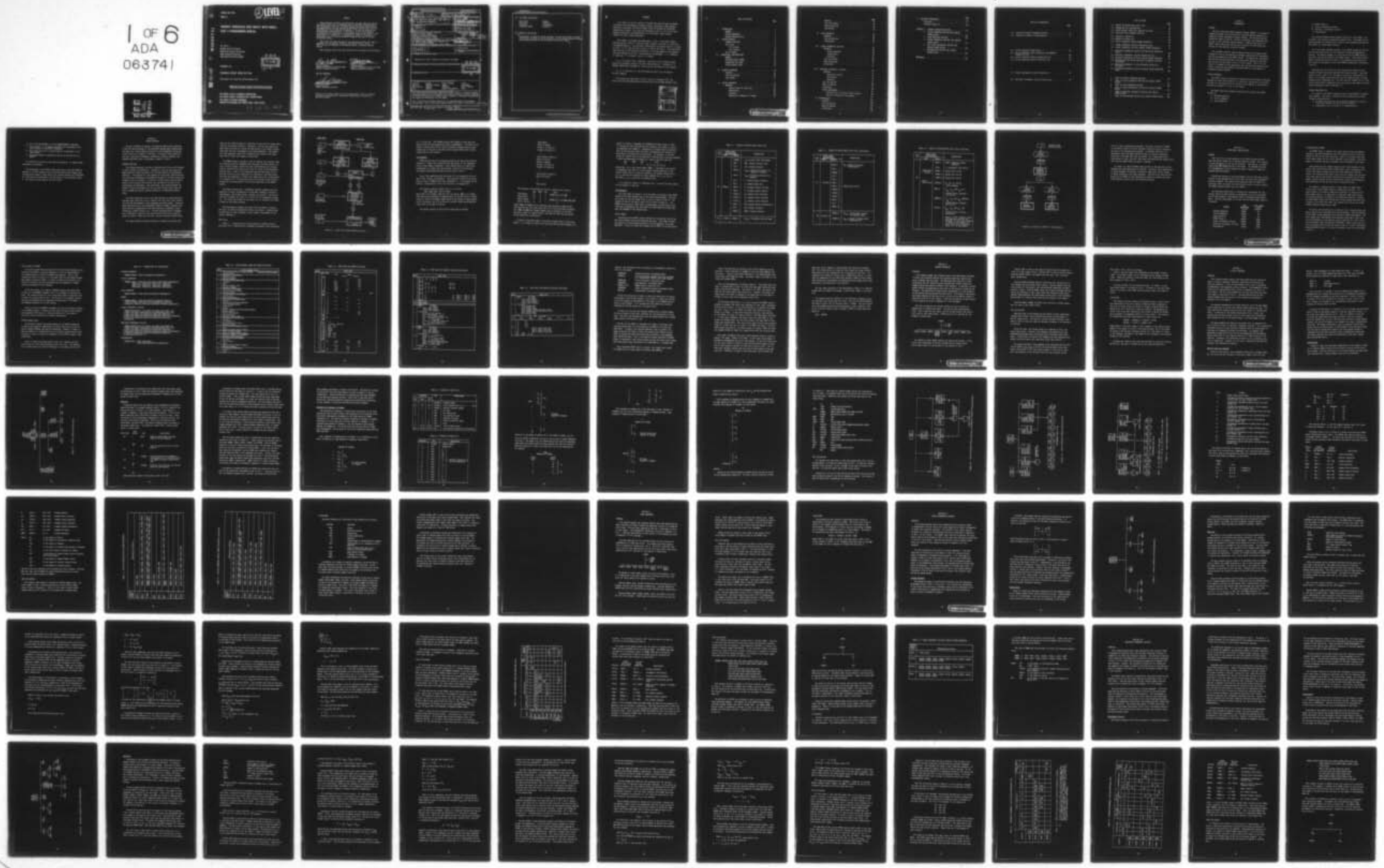
AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3

AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)

DEC 77 R C MORRIS F33615-75-C-3105

UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

1 OF 6
ADA
063741



AFFDL-TR-77-99

PART 3

AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL, PART 3 PROGRAMMING MANUAL

R.C. Morris
Douglas Aircraft Company
McDonnell Douglas Corporation
3855 Lakewood Boulevard
Long Beach, California 90846

DECEMBER 1977

TECHNICAL REPORT AFFDL-TR-77-99

Final Report For Period July 1975-December 1977

Approved for public release; distribution unlimited

AIR FORCE FLIGHT DYNAMICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Larry G Moosman
LT. LARRY G. MOOSMAN
Project Manager
Improved Windshield Protection ADPO
Vehicle Equipment Division

Robert E Witt
ROBERT E. WITTMAN
Program Manager
Improved Windshield Protection ADPO
Vehicle Equipment Division

FOR THE COMMANDER:

SOLOMON R. METRES
Acting Director
Vehicle Equipment Division

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

18 1. REPORT NUMBER
AFFDL-TR-77-99 PART 3

2. GOVT ACCESSION NO.
19 TR-77-99-PT-3

3. RECIPIENT'S CATALOG NUMBER

4. TITLE ~~(and Subtitle)~~
6 AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL.
PART 3. PROGRAMMING MANUAL.

~~5. TYPE OF REPORT & PERIOD COVERED~~
9 FINAL REPORT.
July 1975 - December 1977,

~~6. PERFORMING ORG. REPORT NUMBER~~
14 MDC-J7174-PT-3

7. AUTHOR(s)
R. C. Morris
10

~~8. CONTRACT OR GRANT NUMBER(s)~~
15 F33615-75-C-3105

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Douglas Aircraft Company
McDonnell Douglas Corporation
Long Beach, California 90846

416 400

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Project: 16 2202
Task: 02
Work Unit: 01
17 02

11. CONTROLLING OFFICE NAME AND ADDRESS
Air Force Flight Dynamics Laboratories (AFFDL/FEW)
Air Force Wright Aeronautical laboratories
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

12. REPORT DATE
11 December 1977

~~13. NUMBER OF PAGES~~
506

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
12 494 p.

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Aircraft
Analysis
Applications
Bird
Canopy
Computer Program
Damping
Dynamic
Finite Element
Impact
Laminate
Large Displacements
Linear
Math Model
Matrix Methods
Modal

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the Bird Impact Math Model (IMPACT), a computer program designed especially for the purpose of calculating transient dynamic responses of aircraft windshield and canopy systems, composed of laminated transparencies and supporting structures, to bird impact.

Part 3 describes the design, operation, and implementation of the computer program code. Each of the seven programs comprising IMPACT are described

→ next page

slt

416 400

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (Continued)

Multi-layer
Nonlinear
Plasticity
Structural model
Thermal
Transient
Transparency
Windshield

20. ABSTRACT (Continued)

individually in terms of coding strategy, internal and external storage utilization, limitations, error detection, and interfaces between programs. Detailed descriptions of each subroutine are also included.

FOREWORD

This report is one of a series of reports that describe work performed by Douglas Aircraft Company, McDonnell Douglas Corporation, 3855 Lakewood Boulevard, Long Beach, California 90846, under the Windshield Technology Demonstrator Program. This work was sponsored by the U.S. Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base, under Contract F33615-75-C-3105, Project 2202/0201.

This report is divided into three parts. Part 1 is entitled "Theory and Application", Part 2 is entitled "User's Manual", and Part 3 is entitled "Programming Manual." The principal investigators and authors were P. H. Denke for Part 1, G. R. Eide for Part 2 and R. C. Morris for Part 3. The significant contributions of the following individuals in the development of the computer code and preparation of Part 3 are gratefully acknowledged: J. E. Anderson, L. Chahinian, T. W. Gladhill, and P. R. Lindsey.

Mr. D. C. Chapin, Capt., USAF Ret., was the Air Force Project Manager during the conceptual phase of the work reported herein. Lieutenant L. G. Moosman (AFFDL/FEW) succeeded Mr. Chapin during the conduct of the program.

Mr. J. H. Lawrence, Jr., was the Program Director for the Douglas Aircraft Company.

This report was submitted to the Air Force on 7 December 1977, and covers the work performed during the period July 1975 through December 1977.

TABLE OF CONTENTS

PAGE

PRECEDING PAGE BLANK-NOT FILMED

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
INTRODUCTION

GENERAL

The Bird Impact Math Model Computer Program (IMPACT) is a system of computer programs for calculating the transient dynamic response of a windshield system subjected to impact loading. The programs were designed and coded to provide a reliable, efficient, and convenient tool for use in the windshield design process. Effective application of IMPACT should reduce the time and cost required for both design and testing of these complex structural systems.

This document provides detailed descriptions of the computer code intended for use by the engineer/programmer responsible for implementation and maintenance of the IMPACT system. The theoretical development and the resulting formulations of the analytical approach to the solution process are described in Part 1 of this report. Part 2 of this report is intended for the engineering user and describes modeling techniques, input data requirements, and application of the programs to windshield design problems.

SYSTEM COMPONENTS

The IMPACT system is comprised of seven stand alone computer programs designed for batch mode operation. In addition, the design of the system provides for the optional use of the CDC utility UPDATE, an editing and maintenance program for files of card images.

The seven stand alone computer programs which comprise the IMPACT system are

1) Laminate Generator
2) Initial Generator
3) Loads Generator

4) FORMAT Phase II
5) Linear Incremental Solution
6) Nonlinear Incremental Solution
7) Postprocessor

The interface between these programs consists of card image or binary matrix data stored on sequential external files. The design of the system provides for optional use of some of these programs when applying the system to a given design problem.

Six of the seven computer programs in the IMPACT system are new and were written specifically for the Windshield Technology Demonstrator Program. The seventh program is a modified version of FORMAT Phase II, the matrix abstraction phase of the FORMAT system. These modifications, made in conjunction with this effort, are documented separately from this report as supplements to existing FORMAT documentation (References 6 and 10).

Some code for the solution of eigen problems and theory for the sorting of tabular data were obtained from outside sources. The eigen equation solver, used in the incremental solution of the equations of motion, is the RGEIG module obtained from the EISPACK library of eigen problem solvers (Reference 16). The sort routines (QKSØRT, et al.) used in the initial matrix generation step, are original code based on sorting theory developed by Knuth (Reference 17).

CODING CHARACTERISTICS

All coding in the IMPACT system was done in the FORTRAN IV programming language. The code is oriented to the CDC computer due to the following characteristics.

1) No double precision data or arithmetic operations are used in lieu of the accuracy of the 60 bit word length.
2) Alphanumeric data is stored at 10 characters/word.

3) Use of the CDC FORTRAN utilities ENCØDE/DECØDE is employed.

4) The character "*" is used as delimiters for alphanumeric character strings in some FORMAT statements.

5) The system input file is repositioned to re-read data in the input stream.

6) The system design is enhanced by the use of the CDC utility, UPDATE.

All external files used by the code are sequential, no random access techniques are employed.

The development of the IMPACT system took place on the CDC CYBER-74 computer at the ASD Computer Center, Wright-Patterson AFB, Dayton, Ohio. The operating system in place during this time was NOS/BE L414J CYBR CMR3. All compilation was performed by the FTN compiler at optimization level 1. The version of the compiler was FTN 4.5+414.

SECTION II
SYSTEM OVERVIEW

The user initiates an analysis by preparing tabular data which describes the idealized model of the windshield system under investigation. The model may include support structure for the transparency which may itself consist of a number of laminates. The tabular data prepared by the user, consisting of geometry, constraint, material property, and element definition data, is described in detail in Part 2.

PROGRAM FUNCTIONS

The Laminate Generator provides a preprocessing function with respect to multi-layered transparencies. Optionally, the user may prepare the model definition data consistent with the requirements of the Laminate Generator. The program will then augment the input model definition data with the joint coordinates and element definitions of interior joints and elements within a laminate. In addition, surface normal vectors representing a unit pressure distribution over the exterior surface of the transparency are generated. When applicable, this program reduces the amount of input data prepared by the user and provides vector data for use in assembling pressure loading conditions in subsequent analysis steps.

The Initial Generator serves the prime function of transforming the complete model definition data in tabular form into matrix data consistent with the requirements of the subsequent solution process. Using the model definition data, the unassembled element stiffness, damping, and mass matrices are generated as are other matrices of geometric transformations, thermal effects, etc. The data generated here is independent of mechanical loading on the structure and satisfies the requirements of either a linear and nonlinear analysis whichever might follow.

The Loads Generator provides the means of computing discrete joint loads on the idealized model as a function of time and bird length, mass, velocity, and impact direction. Optionally, the user may prepare the model definition data to include the description of the impact loading consistent with the requirements of the Loads Generator. The program will then generate a matrix of incremental joint loads at the times specified in either card image or binary form.

The FORMAT Phase II program is used to compute the structural mass and stiffness matrices, decompose the structural stiffness matrix, extract vibration modes, perform the modal transformation, and produce properly ordered binary output data files for input to the incremental solution step. Input to this processing step are the binary file of matrices from the Initial Generator and, optionally, the loads matrix from the Loads Generator in either card image form or as a binary matrix in a separate file. The required tasks in this phase of the solution process may be accomplished in one or more executions of the FORMAT program.

The Linear and Nonlinear Incremental Solution programs solve the equations of motion incrementally for the modal response, structural displacements, and element forces, stresses, and strains. The results from each increment are output to a binary file for subsequent processing. The format and content of the output file is identical for either a linear or non-linear solution.

The Postprocessor accepts the output file from either a linear or nonlinear solution and selectively prints the results. Selection may be specified for time increments, joints, modes, and elements in any logical combination.

DATA FILES

Figure 2.1 shows the flow of data through a complete solution. Each data file is identified for subsequent reference in this discussion,

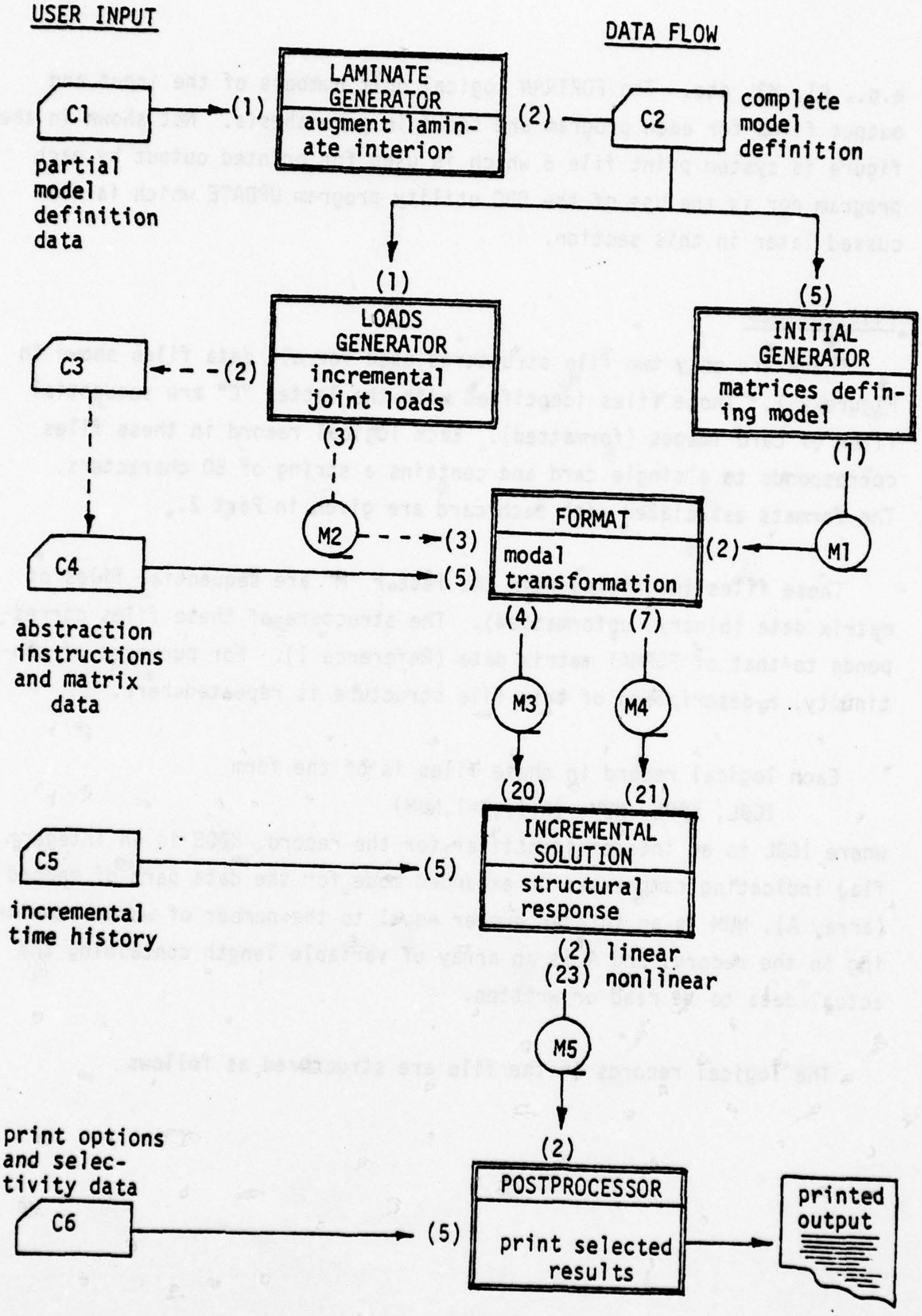


Figure 2.1. Data Flow Through Complete Solution

e.g., C1, M3, etc. The FORTRAN logical unit numbers of the input and output files for each program are shown in parenthesis. Not shown in the figure is system print file 6 which is used for printed output by each program nor is the use of the CDC utility program UPDATE which is discussed later in this section.

File Formats

There are only two file structures used for all data files shown in Figure 2.1. Those files identified with the letter "C" are sequential files of card images (formatted). Each logical record in these files corresponds to a single card and contains a string of 80 characters. The formats associated with each card are given in Part 2.

Those files identified with the letter "M" are sequential files of matrix data (binary, unformatted). The structure of these files corresponds to that of FORMAT matrix data (Reference 1). For purposes of continuity, a description of this file structure is repeated here.

Each logical record in these files is of the form

ICØL, KØDE, NUM, (A(I),I=1,NUM)

where ICØL is an integer identifier for the record, KØDE is an integer flag indicating compressed or expanded mode for the data part of record (array A), NUM is an integer number equal to the number of words remaining in the record, and A is an array of variable length containing the actual data to be read or written.

The logical records in the file are structured as follows

Tape header
Matrix A header
Column 1 of matrix A
Column 2 of matrix A
⋮
Last column of matrix A
Matrix A trailer
Matrix B header
Column 1 of matrix B
Column 2 of matrix B
⋮
Last column of matrix B
Matrix B trailer
⋮
Tape trailer

The contents of the header and trailer records are as follows

	ICØL	KØDE	NUM	Data
Tape header	-10	0	7	(TNAME(I),I=1,6),TMØD
Tape trailer	-20	0	1	0
Matrix header	- 1	0	9	(MNAME(I),I=1,6),MMØD,IMAX,JMAX
Matrix trailer	- 2	0	1	0

where TNAME and MNAME are six character tape and matrix names stored one character per word with blank word fill in trailing words as necessary, TMØD and MMØD are integer numbers used as modifiers of the tape and matrix names for identification, and IMAX and JMAX are the row and column dimensions of the matrix.

In matrix column data, ICØL is a positive integer equal to the column number. If a column of a matrix has more than 50% non-zero elements, the column is written in expanded form (KØDE=0) and NUM is equal to IMAX. If a column has less than 50% non-zero elements, the column is written in compressed form (KØDE=1). Compression is accomplished by forming an array of the non-zero values and their corresponding row locations in the column as the data part of the record. For example, if the 12th column of a matrix with a row dimension of 100 had only 3 non-zero elements, it would be compressed and appear as a logical record of the form

ICØL	KØDE	NUM	Data
12	1	6	$V_1, L_1, V_2, L_2, V_3, L_3$

where ICØL = 12 is the column number, KØDE = 1 indicates a data record in compressed form, NUM = 6 is the number of words remaining in the record (three pairs of value and location), the element values are V_1, V_2, V_3, and the row locations of these elements are L_1, L_2, and L_3. Note that L_1, L_2, L_3 must be in ascending order.

If a column of a matrix is completely null, no record for that column will be present in the file.

File Contents

The files shown in Figure 2.1 are the master files used by the IMPACT system to pass data between each of the stand alone programs. The content of each of the card image files is given in detail in Part 2. The content of each of the matrix data files is given in Table 2.1. The FORMAT tape and matrix names shown in parentheses are specified by user input and, consequently, may vary while all other names are fixed since they are imbedded in the program code.

USE OF UPDATE

The design of the IMPACT system provides for the optional use of the CDC utility UPDATE during the preprocessing phase. This phase includes the execution of the Laminate Generator, Initial Generator, and Loads Generator. Figure 2.2 shows the intended use of UPDATE in an application

TABLE 2.1. CONTENT OF MASTER MATRIX DATA FILES

FILE ID	FORMAT NAMES AND MODIFIERS		DESCRIPTION
	TAPE	MATRIX	
M1	MTAPE,1	UZERØ,1	U_o, original joint coordinates
		MPT,1	MPT, material property data
		ECT,1	ECT, element constants
		PRUPT,1	P_{RUPT}, reordering transform for degrees of freedom
		PRUF,1	P_{RUF}, reordering transform for elements
		KEL,1	k, lumped element stiffness
		MEL,1	m, lumped element mass
		KBEL,1	$\bar{k}$, unlumped element stiffness
		CBEL,1	$\bar{c}$, unlumped element damping
		FFBAR,1	$F_{\bar{F}}$, element force transform
		SIGFB,1	$\sigma_{\bar{F}}$, element stress transform
		EPSIG,1	ϵ_{σ}, element strain transform
		DEBT,1	$\delta\bar{e}_T$, element thermal deformations
		EVT,1	EVT, element variables
		CØNST,1	CØNST, problem constants
M2	LTAPE,1	DPØT,1	$\delta P_{(\phi)T}$, incremental applied loads

TABLE 2.1. CONTENT OF MASTER MATRIX DATA FILES (Continued)

FILE ID	FORMAT NAMES AND MODIFIERS		DESCRIPTION
	TAPE	MATRIX	
M3	(FILE20,1)	(PBUF,1)	$\bar{P}_{UF}$, element force modal transformation
		MPT,1	Copied from file M1
		UZERØ,1	
		ECT,1	
		MEL,1	
		KBEL,1	
		CBEL,1	
		FFBAR,1	
		SIGFB,1	
		EPSIG,1	
		DEBT,1	
		EVT,1	
		CONST,1	
M4	(FILE21,1)	(PBPHI,1)	$\delta\bar{P}_{(\phi)U}$, transformed incremental applied loads
		(PBUPTJ,1)	$\bar{P}_{UPTJ}$, degree of freedom modal transformation

TABLE 2.1. CONTENT OF MASTER MATRIX DATA FILES (Continued)

FILE ID	FORMAT NAMES AND MODIFIERS		DESCRIPTION
	TAPE	MATRIX	
M5		CØNST,1	CØNST, augmented problem constants
		TIME,1	t, incremental time history
	WINDØW,52877 (linear)	UZERØ,1	Copied from file M3
		(PBPHI,1)	Copied from file M4
		(PBUPTJ,1)	Copied from file M4
	TAPE,1 (non-linear)	RESPNS,β	$\bar{\Delta}_\beta$, $\delta\bar{\Delta}_\beta$, $\bar{v}_\beta$, and $\ddot{\bar{v}}_\beta$, modal response
		BARS,β	$\bar{F}_{K_{\beta B}}$, σ_{β_B}, and ε_{β_B}, bar element forces, stresses, and strains
		MEMBRN,β	$\bar{F}_{K_{\beta_M}}$, σ_{β_M}, and ε_{β_M}, membrane element forces, stresses, and strains
		CELLS,β	$\bar{F}_{K_{\beta C}}$, σ_{β_C}, and ε_{β_C}, cell element forces, stresses, and strains
			where β is the increment number and these four matrices are repeated for each time increment. If there are no elements of a given type, the corresponding matrix will not be present.

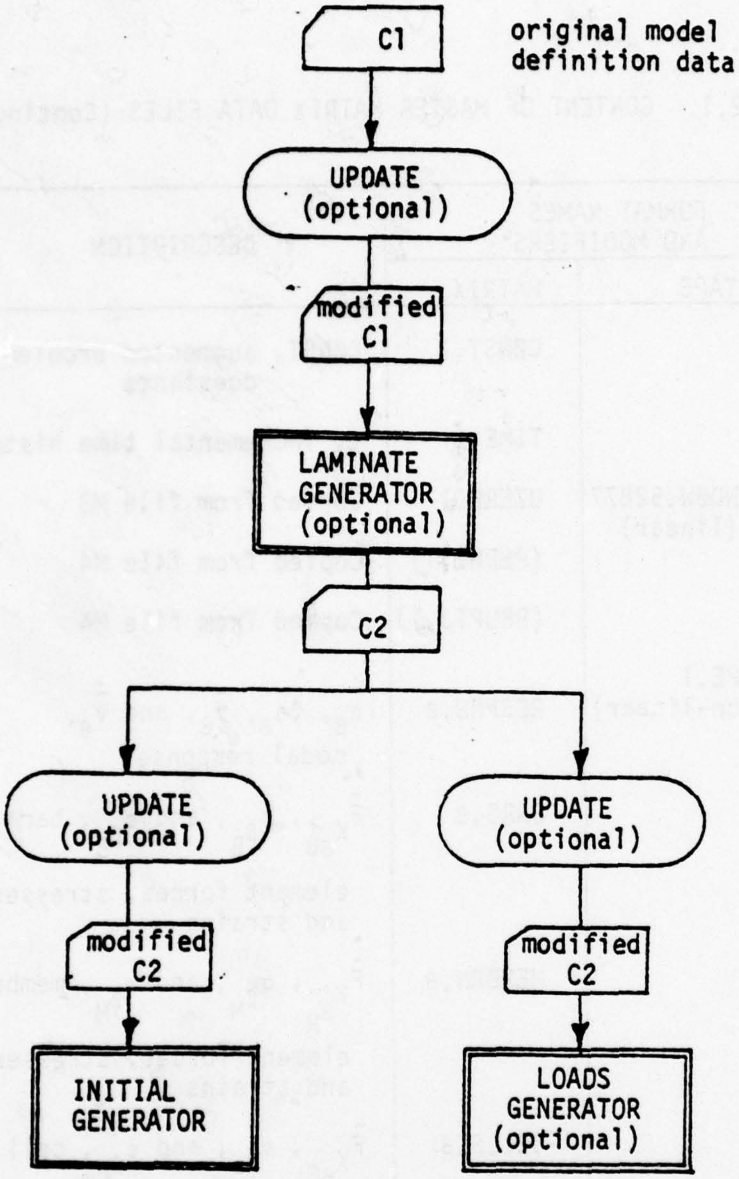


Figure 2.2. Function of UPDATE in Preprocessing

using all three preprocessing programs. The basic function of UPDATE in the IMPACT system is to edit the model definition data as necessary for each of the preprocessing programs. A secondary function is to permanently store the model definition data in an easily accessible form once it has been read into the system. On a large model with a large amount of data, this will avoid many card handling and reader errors each time the model definition data is required.

In Figure 2.2, each of UPDATE steps as well as the execution of the Laminate and Loads Generators are optional. Obviously, many combinations of these programs are possible in applying these tools to the variety of design problems a user might encounter. Suffice to say that the effective use of UPDATE in the preprocessing phase will enhance the users effectiveness by alleviating card handling problems.

SECTION III

OPERATIONAL CONSIDERATIONS

GENERAL

Only one of the six new programs in the IMPACT system currently uses the overlay capability of the CDC operating system. This feature is used extensively by the Initial Generator. The overlay is accomplished using the SEGLOAD option of the loader. The SEGLOAD directives for implementation of this program are given in Section V under the heading CORE UTILIZATION.

Significant savings in core requirements could be realized by using overlay in the implementation of the Linear and Non-Linear Incremental Solution programs. The remaining new programs coded for IMPACT are relatively small and would not yield enough savings to justify overlay. The FORMAT program already uses this feature.

The following tables gives 1) minimum core requirements for the current version of each program as implemented, and 2) the associated size of blank common in the compiled code as implemented where applicable. The core requirement is in octal and the blank common region is in decimal.

	CORE	BLANK COMMON
PROGRAM	REQUIREMENT	LENGTH
	(octal)	(decimal)
Laminate Generator	130000	NA
Initial Generator	130000	NA
Loads Generator	71000	NA
FORMAT Phase II	100000	10000
Linear Incremental Solution	104000	8000
Nonlinear Incremental Solution	134000	8000
Postprocessor	60000	10000

EXTENDING BLANK COMMON

The FORMAT Phase II program, the Linear and Non-Linear Incremental Solutions, and the Postprocessor use blank common exclusively for core storage of data during their execution. Space is allocated within the blank common region for all arrays required in each program. The array space is allocated according to problem size as defined by input data.

Even though the blank common regions are of fixed dimension in the source code, the effective blank common region can be increased at execution time without recompiling any code when operating on the CDC system. By requesting additional core on the job request card and supplying input data to the program indicating the increased size of blank common, larger problems can be processed than could be accommodated by the dimensioned size of the work area in the compiled code.

For example, a program requires a field length of 100000 words (octal) with a blank common region of 10000 words (decimal). For a given problem, an increase of 2000 words (decimal) is required in the blank common region. By requesting an additional 4000 words (octal) for program execution to account for the 2000 word (decimal) increase in blank common region and inputing data to the program specifying this increase in available work space, the program could process the larger problem without recompiling any code.

Of course, this procedure can be used only with a program whose structure and input are consistent with this philosophy. The four programs of the IMPACT system mentioned above conform to these requirements. The blank common size requirements and associated input data to over ride the size parameters in the code are given in Sections VII, VIII, and IX of this document under the heading LIMITATIONS. Again, input card formats associated with this data are given in Part 2. Similar information regarding FORMAT Phase II is given in References 6 and 10 and discussed in Part 2.

OVER RIDING FILE NAMES

On the CDC system when executing multiple stand alone programs as may be done using IMPACT, it may be necessary to over ride system default file names because of conflicts between two programs. That is, one program may output its data on a file named TAPE1 and the second program may expect its input on TAPE3. In order to resolve this conflict, the default file name in one of the programs must be temporarily renamed during the execution of that program.

On the CDC system, file names of FORTRAN programs are established by means of the PROGRAM statement, the first statement of the main program. The system provides for over riding these names at execution time by means of a system control card. The format of this command, however, requires foreknowledge of the sequence of appearance of file names in the PROGRAM statement for a given program.

Table 3.1 gives the PROGRAM statements for each of the seven programs in the IMPACT system. It shows all files that are used by each program, and,-together with Figure 2.1, implicity defines all files used as scratch as well as those used for master input/output.

SYSTEM CONTROL CARDS

As previously stated, the seven programs of the IMPACT system may be used in any number of combinations in actual application to the variety of design problems that may be encountered. However, in order to show an example of a complete deck set up including system control cards and data, a single run using all components of the IMPACT system is described here.

Table 3.2 gives the system control cards for a complete solution while Table 3.3 lists the corresponding user input data. The complete job is broken down into steps for reference in the tables and this dis-

TABLE 3.1. PROGRAM AND FILE DECLARATIONS

LAMINATE GENERATOR

```
  PRØGRAM LAMGEN ( TAPE1,TAPE2,ØUTPUT,TAPE6=ØUTPUT )
```

INITIAL GENERATOR

```
  PRØGRAM BIRDG1 ( TAPE1=512,INPUT,ØUTPUT,TAPE5=INPUT,TAPE6=ØUTPUT,
 .         TAPE7 =512, TAPE8 =512, TAPE9 =512, TAPE10=512,
 .         TAPE11=512, TAPE12=512, TAPE13=512, TAPE14=512,
 .         TAPE15=512, TAPE16=512, TAPE17=512, TAPE18=512 )
```

LOADS GENERATOR

```
  PRØGRAM LØDGEN ( TAPE1,TAPE2,TAPE3,ØUTPUT,TAPE6=ØUTPUT )
```

FORMAT

```
  PRØGRAM FØRMAT ( TAPE1=512,TAPE2=512,TAPE3=512,TAPE4=512,
 1 INPUT=512,TAPE5=INPUT,ØUTPUT=512,TAPE6=ØUTPUT,TAPE7=512,
 6 TAPE8=512,TAPE9=512,TAPE10=512,TAPE14=512,TAPE15=512,TAPE16=512 )
```

LINEAR INCREMENTAL SOLUTION

```
  PRØGRAM RESPNS(TAPE1 =512,TAPE2 =512,TAPE3 =512,TAPE4 = 512
 1,INPUT =512,TAPE5 =INPUT,ØUTPUT=512,TAPE6=ØUTPUT,TAPE7 =512
 2,TAPE8 =512,TAPE9 =512,TAPE10=512,TAPE11=512,TAPE12=512
 3,TAPE13=512,TAPE14=512,TAPE15=512,TAPE16=512,TAPE17=512,TAPE18=512
 4,TAPE19=512,TAPE20=512,TAPE21=512,TAPE22=512,TAPE23=512
 5,TAPE29=512                                                        )
```

NONLINEAR INCREMENTAL SOLUTION

```
  PRØGRAM RESPNS(TAPE1 =512,TAPE2 =512,TAPE3 =512,TAPE4 =512
 1,INPUT =512,TAPE5 =INPUT,ØUTPUT=512,TAPE6=ØUTPUT,TAPE7 =512
 2,TAPE8 =512,TAPE9 =512,TAPE10=512,TAPE11=512,TAPE12=512
 3,TAPE13=512,TAPE14=512,TAPE15=512,TAPE16=512,TAPE17=512,TAPE18=512
 4,TAPE19=512,TAPE20=512,TAPE21=512,TAPE22=512,TAPE23=512
 5,TAPE30=512,TAPE31=512  )
```

POSTPROCESSOR

```
  PRØGRAM PØST ( TAPE1,TAPE2,TAPE3,
 1               INPUT,TAPE5=INPUT,ØUTPUT,TAPE6=ØUTPUT )
```

TABLE 3.2. SYSTEM CONTROL CARDS FOR COMPLETE SOLUTION

STEP	SYSTEM CONTROL CARDS
	RCQ,T075,IO150.CM140000,STCS8. D750545,MORRIS,32096 LIMIT(7777)

1	UPDATE(N,F,C=TAPE1)
	RETURN(NEWPL)
	REWIND(TAPE1)

2	ATTACH(LGO,LAMGENLGO)
	MAP(ON)
	LGO.
	RETURN(TAPE1,LGO)
	REWIND(TAPE2)

3	UPDATE(N,F,C=CASE)
	RETURN(NEWPL.TAPE2)
	REWIND(CASE)

4	ATTACH(BSLCRD,BIRDG1SL)
	UPDATE(N,F)
	REWIND(COMPILE)

5	ATTACH(BLGO,BIRDG1LGO)
	MAP(ON)
	SEGLOAD(I=COMPILE,B=BABS)
	LDSET(PRESET=ZERO)
	LOAD(BLGO)
	NOGO.
	RETURN(BSLCRD,COMPILE,BLGO,NEWPL)
	REWIND(BABS)
	BABS(TAPE2,CASE)
	RETURN(BABS)
	REWIND(TAPE2,CASE)

6	ATTACH(LGO,LODGENLGO)
	MAP(ON)
	LGO(CASE,TX)
	RETURN(CASE,TX,LGO)
	REWIND(TAPE3)

7	ATTACH(PH2,PHASE2A,CY=1)
	PH2.
	REWIND(TAPE2,TAPE3,TAPE4)

8	PH2(,,TAPE4,TAPE3)
	REWIND(TAPE2,TAPE3,TAPE4)

9	PH2(,,,TAPE20,,,,,TAPE21)
	REWIND(TAPE20,TAPE21)

10	ATTACH(LGO,LINEARLGO)
	MAP(ON)
	RFL(140000)
	LGO.
	RETURN(LGO,TAPE20,TAPE21)
	REWIND(TAPE2)

11	ATTACH(LGO,POSTLGO)
	MAP(ON)
	LGO.

TABLE 3.3. INPUT DATA FOR COMPLETE SOLUTION

```
STEP | INPUT DATA
-----+---------------------------------------------------------------------------
     | *DECK CASE
     | 1                  75.0                9332.0
     | 1                             1.0E+6             1.0E-12
     | 19999
     | 2    1                                                  1.0
     | 2    3                                  4.0             1.0
     | 2    5             4.0                                  1.0
     | 2    7             4.0                  4.0             1.0
     | 2    9             8.0                                  1.0
     | 2   11             8.0                  4.0             1.0
     | 2   13            12.0                                  1.0
     | 2   14            12.0
     | 2   15            12.0                  4.0             1.0
     | 2   16            12.0                  4.0
     | 29999
     | 3    1   16                                                          75.0
     | 39999
     | 49999
     | 5    1    1        1.0
     | 5    2    1                             1.0
     | 5    3    1                                             1.0
     | 5    4    2        1.0
     | 5    5    2                             1.0
     | 5    6    2                                             1.0
     | 5    7    3        1.0
     | 5    8    3                             1.0
     | 5    9    3                                             1.0
  1  | 5   10    4        1.0
     | 5   11    4                             1.0
     | 5   12    4                                             1.0
     | 5   13   14                                             1.0
     | 5   14   16                                             1.0
     | 59999
     | 7 100
     | 7 101 100    1    1   1.0
     | 7 199
     | 79999
     | 8    1   11 1.0
     | 89999
     | 10 100      ALUMINUM    2024-T62
     | 10 101 1  10.6     E+6
     | 10 102 1  7.44     E+6
     | 10 103 1  50000.
     | 10 104 1  66800.
     | 10 105 1  .0488
     | 10 106 1  .33
     | 10 107 1  .126     E-4
     | 10 108 1  .259     E-3
     | 10 109 1  80.
     | 10 110 1  1.0    E-03
     | 10 199
     | 109999
     | 11   1   48           .004             5000.            15.
     | 11   2    3          -.8                                -.6
     | 11   3                                                  -1.0
     | 11   4               -1.0
     | 11   5                11.0              2.0              1.0
     | 11   6    1                             1.0
     | 11   6    2            8.0              1.0
     | 11   6    3            8.0              1.0
     | 119999
```

TABLE 3.3 INPUT DATA FOR COMPLETE SOLUTION (Continued)

STEP	INPUT DATA
1	12 1 9 11 13 15 12 2 5 7 9 11 12 3 1 3 5 7 129999 20 1 13 14 100 1.0 20 2 15 16 100 1.0 209999 30 1 9 11 15 13 100 .5 30 2 10 12 16 14 100 .5 309999 40 1 1 3 7 5 40 2 5 7 11 9 409999 50 1 9 .4 E-2 .4 E-2 .4 E-2 50 2 11 .4 E-2 .4 E-2 .4 E-2 50 3 13 .4 E-2 .4 E-2 .4 E-2 50 4 15 .4 E-2 .4 E-2 .4 E-2 509999
3	*DECK CASE1 *READ TAPE2
4	*READ BSLCRD

7	$FORMAT STANDARD
	$RUN GO, LOGIC
	      INPUT TAPE (MTAPE,1)
	      INPUT TAPE (LTAPE,1)
	      OUTPUT TAPE (DECOMP,1)
	$INSTRUCTION
	      PT,OPT = DPOT .DEJCOL. (1)
	      SAVE (DECOMP) DPOT
	      MR = PRUF,MEL .SEQWF.
	      PRUPJ,PRUPS = PRUPT .DEJCOL. (48)
	      PR = PRUPJ .MULT. PT
	      PRT = PR .TRANSP.
	      X,LTL,UR = PRUF,KEL .SEQWF. PRT
	      UT = PRUPT .TMULT. UR
	      SAVE (DECOMP) MR,PRUPJ,LTL,UR
	      PRINT (,,,) UT
	$END

8	$FORMAT STANDARD
	$RUN GO, LOGIC
	      INPUT TAPE (MTAPE,1)
	      INPUT TAPE (DECOMP,1)
	      OUTPUT TAPE (TRANSF,1)
	$INSTRUCTION
	      VAL,TR = MR .USER06. LTL
	      PRINT (,,,) VAL
	      PBUPJ = TR .TMULT. PRUPJ
	      PBUF = TR .TMULT. PRUF
	      OPBO = PBUPJ .MULT. OPOT
	      SAVE (TRANSF) TR,PBUPJ,PBUF,OPBO
	      MODES = PRUPJ .TMULT. TR
	      PRINT (,,F2,,05) MODES
	$SPECIAL
	   25 25 1.0E-6
	$END

TABLE 3.3. INPUT DATA FOR COMPLETE SOLUTION (Continued)

STEP	INPUT DATA
9	$FORMAT STANDARD $RUN GO, LOGIC INPUT TAPE (MTAPE,1) INPUT TAPE (TRANSF,1) OUTPUT TAPE (FILE20,1) OUTPUT TAPE (FILE21,1) $INSTRUCTION SAVE (FILE20) PRUF SAVE (FILE20) MPT SAVE (FILE20) UZERO SAVE (FILE20) ECT,MEL,KBEL,CBEL,FFBAR SAVE (FILE20) SIGFB,EPSIG,DEBT,EVT,CONST SAVE (FILE21) DPBO SAVE (FILE21) PRUPJ $END
10	1 6 10 25F 16 15000 10.0 20.0 30.0 40.0 50.0 60.0 70.0 80.0 90.0 100.0
11	1 11 2 1111 3 111 4 1111 5 1111 6 1111 7 CARD 1, FIRST TITLE LINE 8 CARD 2, FIRST TITLE LINE 9 CARD 1, SECOND TITLE LINE 10 CARD 2, SECOND TITLE LINE 20 9999 1= 10 30 9999 1= 16

cussion. The following files are assumed to be permanently stored disc files in the system.

LAMGENLGØ	Laminate Generator relocatable decks
BIRDG1SL	Initial Generator SEGLØAD directives preceded by a *DECK UPDATE control card (card images)
BIRDG1LGØ	Initial Generator relocatable decks
LØDGENLGØ	Load Generator relocatable decks
PHASE2A, CY=1	FØRMAT Phase II absolute file
LINEARLGØ	Linear Incremental Solution relocatable decks
PØSTLGØ	Postprocessor relocatable decks

During this discussion, Figure 2.1 should be referred to for master input/output files of each program. Also, Table 3.1 should be referred to for the sequence of appearance of file names in the PRØGRAM declarations of each program. It is assumed that the reader has a working knowledge of the CDC system and its control cards.

In Table 3.2, the job card requests 140000 words of central memory for the run which is more than sufficient to load and execute all programs. The LIMIT card requests more than the system default disc storage space normally allocated to a job.

The CDC utility UPDATE is executed first (Step 1) receiving input from the system input file and outputing the CØMPILE file under the name TAPE1. In Step 2, the Laminate Generator uses all default file names accepting file TAPE1 as input and creating file TAPE2 as output. TAPE1 is the partial model definition data and TAPE2 is the augmented model definition data. This is followed by another execution of UPDATE (Step 3) simulating a final editing and/or save of this data which, rather than being output under the default file name CØMPILE, is renamed CASE.

Step 4 executes UPDATE again to create a card image input stream for SEGLØD which is output under the default name CØMPILE.

Step 5 includes both the loading of the Initial Generator and its execution. Input directives to SEGLØAD is the card image CØMPILE file from Step 4. Input to the Initial Generator is the complete model definition data, card image file CASE. The normal output file TAPE1 is over ridden and renamed TAPE2.

Next the Load Generator is executed (Step 6). The normal input file TAPE1 is renamed CASE in order to accept the same model definition data used by the Initial Generator. The output is on TAPE3, the default file name for the binary matrix data output file. However, the normal card image output file TAPE2 is renamed to the dummy name TX since TAPE2 is already in use as the Initial Generator output file.

The following three steps, 7, 8, and 9, execute the FØRMAT Phase II program to assemble the input files necessary for the Linear Incremental Solution which follows. In the first execution, all default file names are used. Input is on file TAPE2 from Step 5 and TAPE3 from Step 6. Output is on TAPE4. In the second execution (Step 8), input consists of file TAPE2, the same file input to Step 7, and TAPE4 output from Step 7. In this step, therefore, the normal input file name TAPE3 is over ridden and renamed TAPE4, and the normal output file name TAPE4 is over ridden and renamed TAPE3. This sets up the stage for the final execution (Step 9) in which the input is, again, file TAPE2 from Step 5, and file TAPE3 output from Step 8. The normal output file names, TAPE4 and TAPE7, are renamed TAPE20 and TAPE21 for compatibility with the following step.

The Linear Incremental Solution is then executed (Step 10) using all default file names. Output is on TAPE2. In this step, blank common is extended to accommodate the size of the problem. On the first input card for this step (Table 3.3, Step 10), the available blank common is specified to the program in decimal as 15000 (last value on card). This is 7000 words (decimal) greater than the size of blank common in the compiled code. Therefore, in order to make additional core available for execution, an RFL command is placed before the load and go command, LGØ. This effectively over rides the field length the loader normally establishes from the relocatable file it accesses for the load. The requested field length on the RFL card, 140000 words (octal), is sufficient to compensate for the 7000 words (decimal) added to blank common. (See minimum core requirements and dimensioned blank common block sizes discussed at the beginning of this section).

The last step, execution of the Postprocessor (Step 11) is then performed using all default file names, in which case, TAPE2 is the input file.

This example would be identical if the Nonlinear Incremental Solution had been used rather than the linear with the following exception. The default output file for the non-linear is TAPE23. The last control card, the command executing the Postprocessor, would, therefore, have to over ride the default input file name of TAPE2 to TAPE23 and would appear as

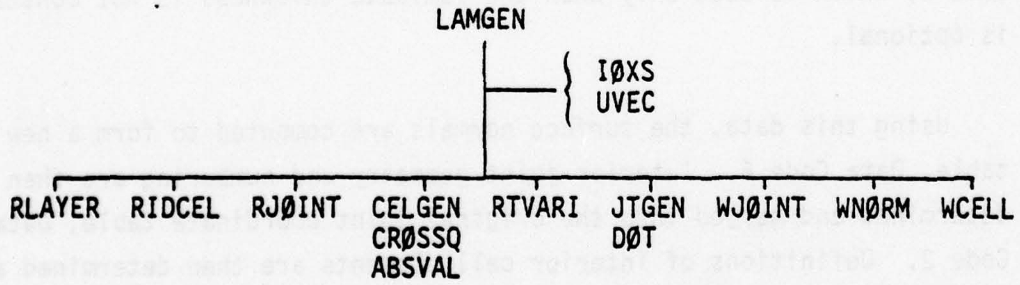
```
LGØ (, TAPE23)
```

SECTION IV
LAMINATE GENERATOR

OVERVIEW

This program accepts user prepared tabular data describing the model, including the laminated section and generates additional tabular data which is merged into the input. The generated data consists of joint coordinates and cell definition data representing interior joints and elements below the exterior surface of the laminate. In addition, surface normal vectors are generated at each joint on the exterior surface of the laminate. The magnitude of these vectors corresponds to a unit pressure on the surface area associated with each joint. The format of the tabular card input prepared by the user as well as that of the tabular output is described in Part 2 of this report. Detailed descriptions of each routine in this program are given in Appendix A of this document.

The core required by this program without the use of overlay or segmentation is 130000 octal words. The program uses three external files. One is used for card input, one for printed output, and, optionally, one for card image output. Organization of all routines for the program is shown below.



No labeled or blank common regions are used by the program. Arrays A and N are dimensioned in the main program which allocates space within these arrays for use by each of the routines it calls.

Routine IØXS is used to read data not required by the program and write it on the output file(s). Routine UVEC is used to unitize vectors. Routines RLAYER, RIDCEL, and RJØINT read the layer, cell, and joint data, respectively.

Routine CELGEN generates the cell definition data for the cells in the laminate and the surface normal vectors. Routine CRØSSQ finds the cross product of two vectors, and routine ABSVAL determines the magnitude of a vector. Routine RTVARI reads the variable thickness data and stores the thickness of each layer under each laminate surface joint. Routine JTGEN generates the coordinates of each joint in the laminates. Routine DØT is a function that forms the dot product of two vectors.

Routines WJØINT, WNØRM, and WCELL write the joint, surface normal, and cell data on the output file(s).

FILE UTILIZATION

Required input to the program are the tables of joint coordinates, laminate definitions, and cell definitions which are Data Codes 2, 7 and 40, respectively. The table of variable laminate thickness, Data Code 8, which is used only when the laminate thickness is not constant, is optional.

Using this data, the surface normals are computed to form a new table, Data Code 6. Interior joint geometry and numbering are then determined and merged into the original joint coordinate table, Data Code 2. Definitions of interior cell elements are then determined and merged into the original cell definition table, Data Code 40.

The output consists of the augmented joint coordinate and cell definition tables, Data Codes 2 and 40, the surface normal table, Data Code 6, and copies of all other tables present in the original input with the exception of the laminate definition and thickness tables, Data Codes 7 and 8, which are omitted.

There is one card input file referenced by the integer variable ITAPE. The primary output file is referenced by the integer variable JTAPE. These two variables, ITAPE and JTAPE, are initialized in the main program as files 1 and 2.

The code provides for the following option. If JTAPE = 6, only the printed output is written to file 6. If JTAPE ≠ 6, the card image output is written to file JTAPE and the printed output is written to file 6.

LIMITATIONS

The program uses two singularly subscripted arrays, A and N, which are dimensioned in the main program as 25000 and 2000, respectively. The sizes of these arrays are stored in integer variables MSIZEA and MSIZEN which are initialized in the main program. The dimensions of arrays A and N and their respective sizes stored in MSIZEA and MSIZEN can be changed to accommodate smaller or larger models. The formulae for determining the space requirements for these arrays are:

$$\mathrm{MSIZEA} \geq 10*\mathrm{NCELLS} + 7*\mathrm{NJOINT} + \mathrm{NJT}*\mathrm{NL}$$

$$\mathrm{MSIZEN} \geq 4*(\mathrm{NL}-1) + \mathrm{NQUAD} + \mathrm{NJT}$$

where NCELLS is the total number of cell elements in the complete model, NJØINT is the number of joints in the complete model, NJT is the number of joints on the exterior laminate surface, NL is the number of layers in the laminate, and NQUAD is the number of cell elements on the exterior surface of the laminate.

Although the format of the input data provides for multiple laminate definitions, the code is limited to only one laminate definition.

SECTION V

INITIAL GENERATOR

OVERVIEW

This program accepts tabular input data describing the structural model in terms of geometry, elements, constraints and material and physical properties. From this description, matrix data is generated as demanded by the subsequent solution process. The mathematical formulation of the output matrix data is given in Appendix H of Part 1 of this report. The card input prepared by the user describing the structurla model is described in Part 2. Detailed descriptions of each labeled common block and routine in this program are presented in Appendix B of this document.

The program implementation includes extensive use of the CDC segmentation capability to overlay both labeled common regions and code. The core required for program execution using this feature is 130,000 octal words. In addition to the system input/output files, 5 and 6, the program uses thirteen external files of which twelve are intermediate data storage and one for the master output. No blank common is used.

The data assembled and output on the master file is compatible for use in either a linear or nonlinear incremental solution. This compatibility inherently results in some additional overhead since the requirements of each type of incremental solution are different. For example, the generation of matrices ECT and EVT and the computation of the Ramberg-Osgood coefficient in runs preparing data for a linear solution is not required. Conversely, the generation of matrix σ_F for a nonlinear solution is unnecessary. However, for the sake of consistency and user convenience, this approach was adopted.

Special Input for Checkout

There are some special input parameters pertaining to program checkout runs which are not discussed in the User's Manual, Part 2 of this report. These parameters are input under Data Code 1. On card 1 in columns 3 to 6 an integer master tape dump flag, MTDF, may be entered which controls the printing of output matrix data according to:

MTDF = 0	no dump
MTDF = 1	dump header/trailer
MTDF = 2	full dump

Also, on card 1 in columns 7 to 10 an integer joint number can be entered which will trigger the printing of all intermediate calculations for all elements connected to the joint specified. Finally, on card 1 in columns 11 to 14 an integer print flag can be input, which, if non zero, triggers the printing of all intermediate calculations in the assembly of matrices PRUPT and PRUF.

On card 2 of Data Code 1, in columns 3 to 6, an integer can be entered which is used as the maximum number of lines per page for printing. If not input, the number defaults to 45. Also, in columns 19 to 33 and 34 to 48, two coefficients are input as noted in the User's Manual. The first coefficient is used in defining the stiffness, or more appropriately, the lack of stiffness in bar elements with gaps in connectivety specified. The second coefficient is used to suppress insignificant values from the stiffness matrix of each element in the structural model. This was done to reduce matrix density and thus promote storage in compressed form and reduce execution time in mathematical operations. The values specified for these two coefficients in the User's Manual were derived empirically.

Organization

Figure 5.1 shows the functional organization of the program in terms of all labeled common blocks and the principle routines called by the main program. Vertical positioning in the figure depicts the basic overlay structure used. Labeled common blocks are in parentheses.

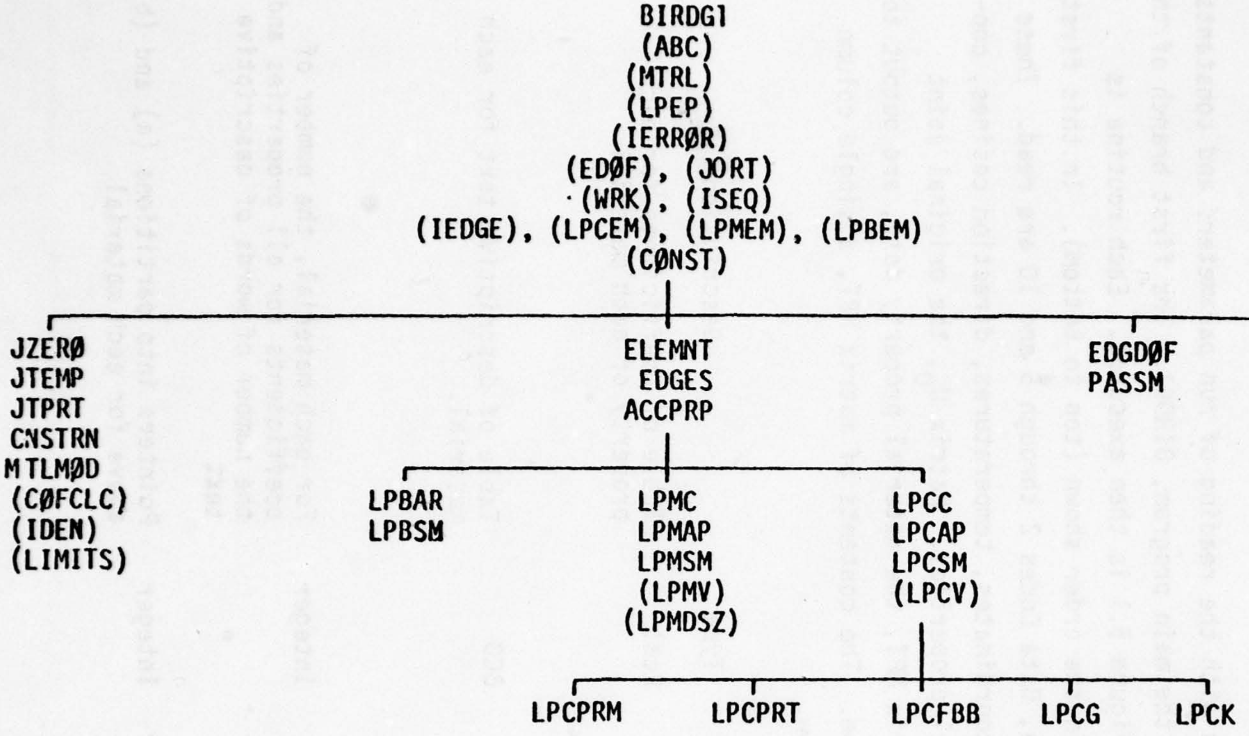


Figure 5.1. Initial Generator Organization

Functionally, the program can be broken down into three major steps; the processing of joint, vector and material property data, the processing of element data, and the reordering of degrees of freedom prior to final output of matrix data.

Operation

Processing begins with the reading of run parameters and constants under Data Code 1 by the main program, BIRDG1. The first branch of the tree on the left in Figure 5.1 is then executed. Each routine is invoked in sequence in the order shown (top to bottom). In this first major processing step, Data Codes 2 through 5 and 10 are read. These are tables of joint coordinates, temperatures, direction cosines, constraints, and material properties. Matrix U_0, the original joint coordinates, and matrix MPT, the material property data, are output to the master output tape. The contents of matrix MPT, a single column matrix, is shown below.

Partitions	MPT Record	Type	Description
(a)	1	Real	Table of coefficients for each property of each material
	.		
	.		
	NCF		
	1	BCD	Table of descriptive text for each material.
(b)	.		
	.		
	ND		
	1	Integer	For each material, the number of coefficients for all properties and the number of words of descriptive text
(c)	.		
	.		
	MT		
	1	Integer	Pointers into partitions (a) and (b) above for each material
(d)	.		
	.		
	MT		

where the total length of the column is NCF + ND + 2MT.

Processing of element data, the second major step, is accomplished by middle branch of tree shown in Figure 5.1. A loop is set up in the main program to process all element data. The loop begins with a call to routine ELEMNT. This routine reads element definition data, Data Codes 20, 30, 40 and 50, one element at a time, and returns flags to the main program which control subsequent processing. One flag signals the end of data for the Data Code currently being processed which dictates either going on to next table, or, if all tables have been processed, exiting the loop.

If a return from routine ELEMNT indicates the presence of data for a particular type element, routines EDGES and ACCPRP are called in turn to assemble an array of joint number pairs for each element edge force and to establish the material properties for the element, respectively. One of the three sub-branches is then executed according to the element type; lumped parameter bar (LPB), lumped parameter membrane (LPM), or lumped parameter cell (LPC). No output data is written to the master output file in this step. Output consists of intermediate data stored on scratch files.

The last major processing step is accomplished by the two remaining branches on the right in Figure 5.1. These routines are executed once in the order EDGDØF, PASSM, WTAPE1, DUMPMT, and PRTCØN. Routine EDGDØF assembles an array of sorted unique joint pairs for each edge in the model from data previously assembled by routine EDGES. Using this and other data regarding constraints, routine PASSM then assembles and outputs matrices PRUPT and PRUF in the reordered row format. Finally, the three routines of the last branch are executed to write all remaining master output data, dump the master output tape (optional), and print the data from matrix CØNST, respectively. CØNST contains all model constants, sizing data, and other run parameters (Appendix B, Labeled Common CØNST).

Two special or psuedo matrices are formed and output which are not part of the theoretical development given in Part 1. These matrices are ECT, the element constant table, and EVT, the element variable table.

Each element contributes a column to ECT and EVT. The data for a column of ECT and EVT for each type element is given in Tables 5.1 and 5.2, respectively. These two matrices are used during each time increment of the nonlinear response solution. Matrix ECT is read only while matrix EVT, containing material property and stress/strain state data of each element, is read and rewritten during each increment.

Reordering of Degrees of Freedom

The reordering of degrees of freedom which takes place in the final processing step is done to reduce the wavefront of the structural stiffness matrix subsequently decomposed by the SEQWF module of FORMAT (Reference 10). The three-row formats (sequence) of degrees of freedom referred to in this report are the total degrees of freedom, T, the unconstrained degrees of freedom, U, and the reordered unconstrained degrees of freedom, RU. The latter being the desired order for SEQWF and, consequently, the row order of the Initial Generator output matrices PRUPT and PRUF.

The T degrees of freedom consist of global X,Y,Z translation at each joint followed by all edge degrees of freedom as shown below.

T

Degrees of Freedom

$$\begin{array}{cll}
1 & J_{1X} & \\
2 & J_{1y} & \left.\right\} J_{1_T} \\
3 & J_{1Z} & \\
. & J_{2X} & \qquad \text{All joint degrees of freedom} \\
. & J_{2y} & \left.\right\} J_{2_T} \\
. & J_{2Z} & . \\
. & & . \\
. & & . \quad .
\end{array}$$

TABLE 5.1 CONTENTS OF MATRIX ECT

LOCATION			TYPE	DESCRIPTION
Bar	Mem	Cell		
1	1	1	integer	element number
2-3	2-5	2-9	integer	joint connectivity (p, q, r . . . etc.)
4	6	10	integer	material property number
5	-	-	real	bar area
6	-	--	real	bar tension gap
7	-	-	real	bar compression gap
-	7	-	real	membrane thickness
-	8	11	real	stress orientation angle
8	9	12	real	mass
9	10-13	13-24	integer	T dof of edge forces
10-11	14-22	25-54	real	unit thermal deformation coefficients

TABLE 5.2 CONTENTS OF MATRIX EVT

LOCATION	TYPE	DESCRIPTION	
1	integer	element number	
2	real	temperature	
3	real	E	material properties at current temperature
4	real	E_A	
5	real	$\bar{\sigma}_A$	
6	real	$\bar{\sigma}_r$	
7	real	$\bar{\epsilon}_r$	
8	real	μ	
9	real	α_T	
10	real	ρ	
11	real	s	
12	real	h	
13	real	n	
14	real	$\bar{E}$	
15	real	$\bar{\sigma}_L$	

$$\begin{array}{l}
\vdots \\
\left.\begin{array}{rl} \cdot & J_{n_x} \\ \cdot & J_{n_y} \\ 3*NJ & J_{n_z} \end{array}\right\} J_{n_T} \\
\left.\begin{array}{r} 1 \\ \cdot \end{array}\right\} E_1 \\
\left.\begin{array}{r} \cdot \\ \cdot \end{array}\right\} E_2 \qquad \text{All edge degrees of freedom} \\
\vdots \\
\left.\begin{array}{r} \cdot \\ NE \end{array}\right\} E_n
\end{array}$$

where NJ is the number of joints, NE is the number of edges, J_{n_T} are the X,Y,Z global translation at joint n, and E_n are all edges connected to joint n. The edges in E_n are identified by joint pairs in ascending order where the second joint is always greater than the first and the positive direction is taken to be from the first to the second joint. For example,

Edge
Degrees of Freedom

First Joint	Second Joint	
1	6	E_1
1	18	
1	56	
2	3	E_2
2	17	

3	54	E_3
4	6	E_4
4	11	
4	15	
4	105	

etc.

The U degrees of freedom are in the same order as the T degrees of freedom, but with joint constrained degrees of freedom omitted. Edge degrees of freedom cannot be constrained.

U

Degrees of Freedom

```
      1    } J1U
      .    }
      .    }
      .    } J2U      Unconstrained joint
      .        .      degrees of freedom
      .        .
      .        .
      .        .
      .        .
      .        .
      .    } JnU
3*NJ-NC    }
      1    } E1
      .               All edge
      .    } E2       degrees of freedom
      .        .
      .        .
      .        .
     NE    } En
```

where NC is the number of constraints, and J_{n_U} are the unconstrained global translation at joint n.

The RU degrees of freedom consist of the U degrees of freedom with the edge degrees of freedom for a joint immediately following the unconstrained joint degrees of freedom, as shown below.

RU
Degrees of Freedom

$$\begin{array}{rl}
\left.\begin{array}{r}1\\ \cdot\end{array}\right\} & J_{1_U}\\
\left.\begin{array}{r}\cdot\\ \cdot\\ \cdot\end{array}\right\} & E_1\\
\left.\begin{array}{r}\cdot\\ \cdot\end{array}\right\} & J_{2_U}\\
\left.\begin{array}{r}\cdot\\ \cdot\\ \cdot\end{array}\right\} & E_2\\
\left.\begin{array}{r}\cdot\\ \cdot\end{array}\right\} & J_{3_U}\\
\cdot & \cdot\\
\cdot & \cdot\\
\cdot & \cdot\\
\cdot & \cdot\\
\left.\begin{array}{r}\cdot\\ \cdot\\ \cdot\end{array}\right\} & J_{n_U}\\
\left.\begin{array}{r}\cdot\\ \text{3*NJ+NE-NC}\end{array}\right\} & E_n
\end{array}$$

OVERLAY

Loading of the Initial Generator program should include the use of the CDC segmentation capability. The basic overlay structure is shown in Figure 5.1. Note that the labeled common regions are overlayed as well as code. The SEGLOAD directives to accomplish this overlay structure are given below. Beginning card columns are noted above each of the three fields.

```
1      8        17
       TREE     BIRDG1-(IN,ELEM,ASSM,ØUT)
ELEM   TREE     EA-(BAR,MEM,CELL)
       GLØBAL   CØNST,IEDGE,IERRØR,ISEQ,EDØF,LPEP,MTRL
       GLØBAL   CØN.RM,Q8.IØ.,FCL.C.,IØ.BUF.
EDØF   EQUAL    JORT
ISEQ   EQUAL    WRK
IEDGE  EQUAL    LPBEM,LPMEM,LPCEM
IN     INCLUDE  CNSTRN,JTEMP,JTPRT,JZERØ,MTLMØD,SKDATA,TAPEHD
IN     GLØBAL   CØFCLC,LIMITS
EA     INCLUDE  ACCPRP,EDGES,ELEMNT
BAR    INCLUDE  LPBAR,LPBPRT,LPBSM
MEM    INCLUDE  LPMAP,LPMC,LPMPRM,PMPRT,LPMSM
MEM    GLØBAL   LPMDSZ,LPMV
CELL   TREE     LPCAP-LPCC-LPCSM-(LPCPRM,LPCPRT,LPCFBB,LPCG,LPCK)
LPCAP  GLØBAL   LPCV
ASSM   INCLUDE  PASSM,EDGDØF
ØUT    INCLUDE  DUMPMT,PRTCØN,TAPETR,WTAPE1
       END      BIRDG1
```

FILE UTILIZATION

All tabular card input data is read from system input file 5 and all printed output is written to system output file 6. In addition, thirteen external files are used. Units 7 through 18 are used as scratch files and unit 1 is used for master matrix data binary output.

Figures 5.2, 5.3 and 5.4 show the use of these files during the three major processing steps of the Initial Generator program. The content of each of these files is summarized in the following.

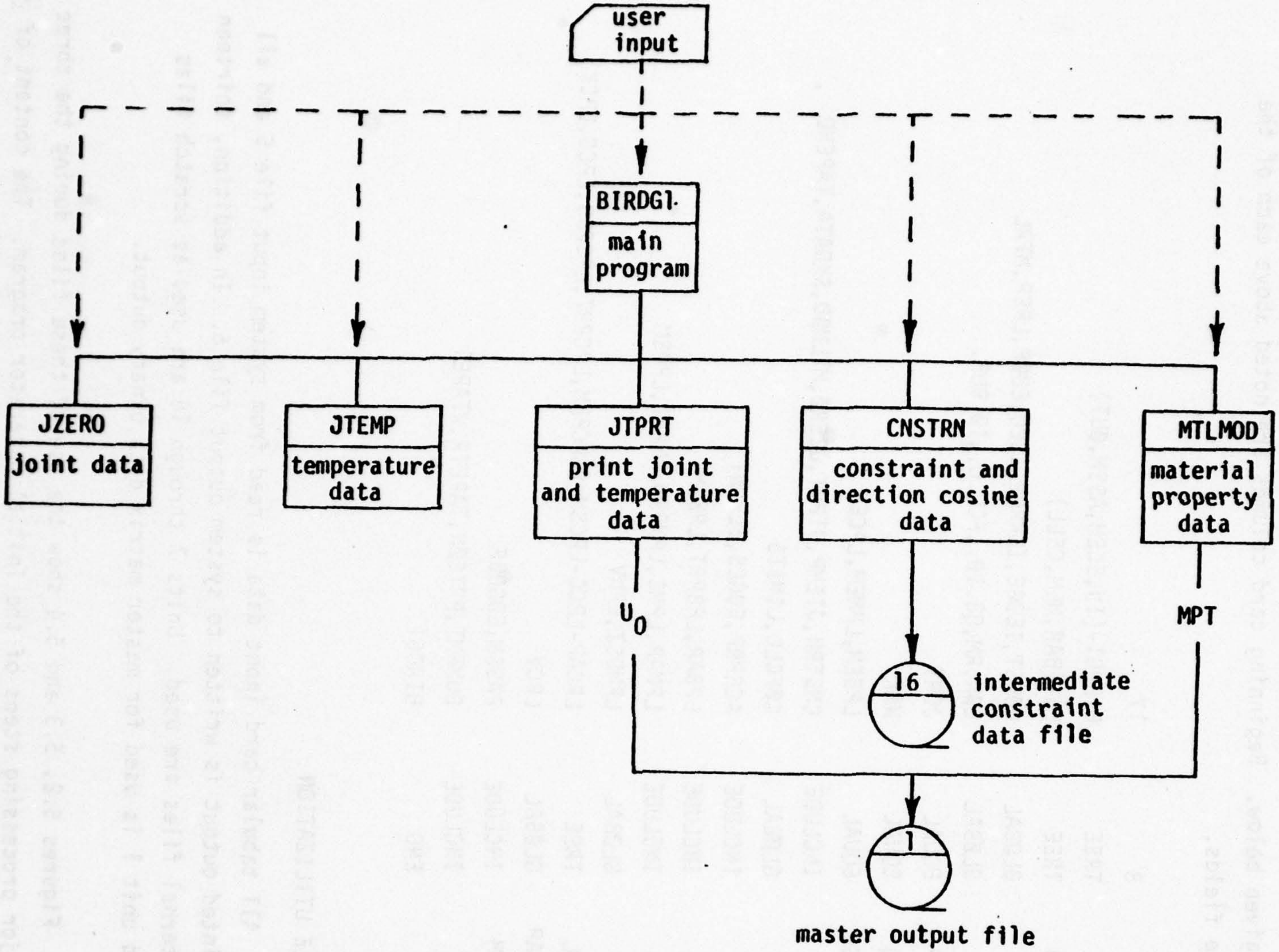


Figure 5.2. Initial Generator Joint, Constraint, and Material Property Processing Files

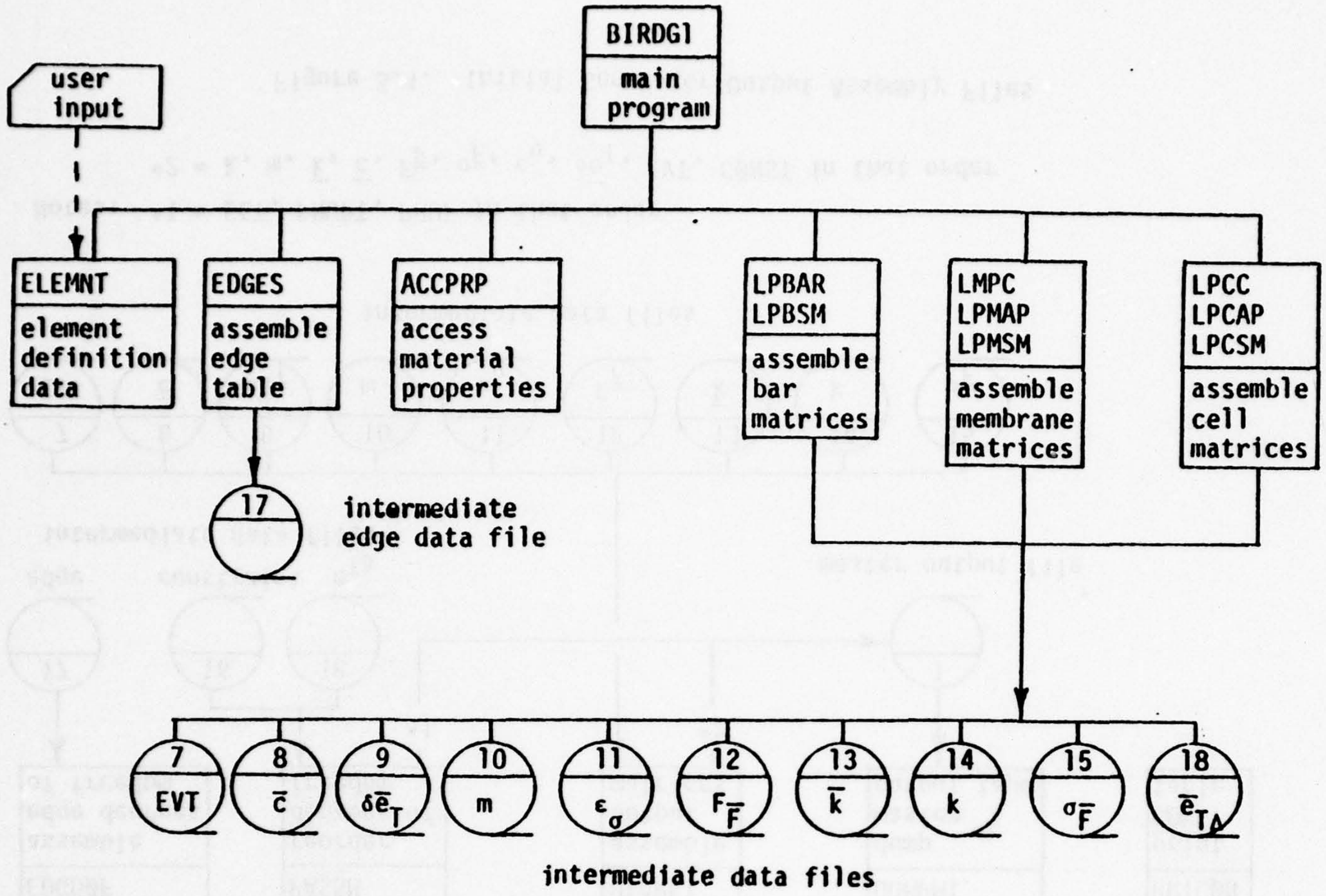
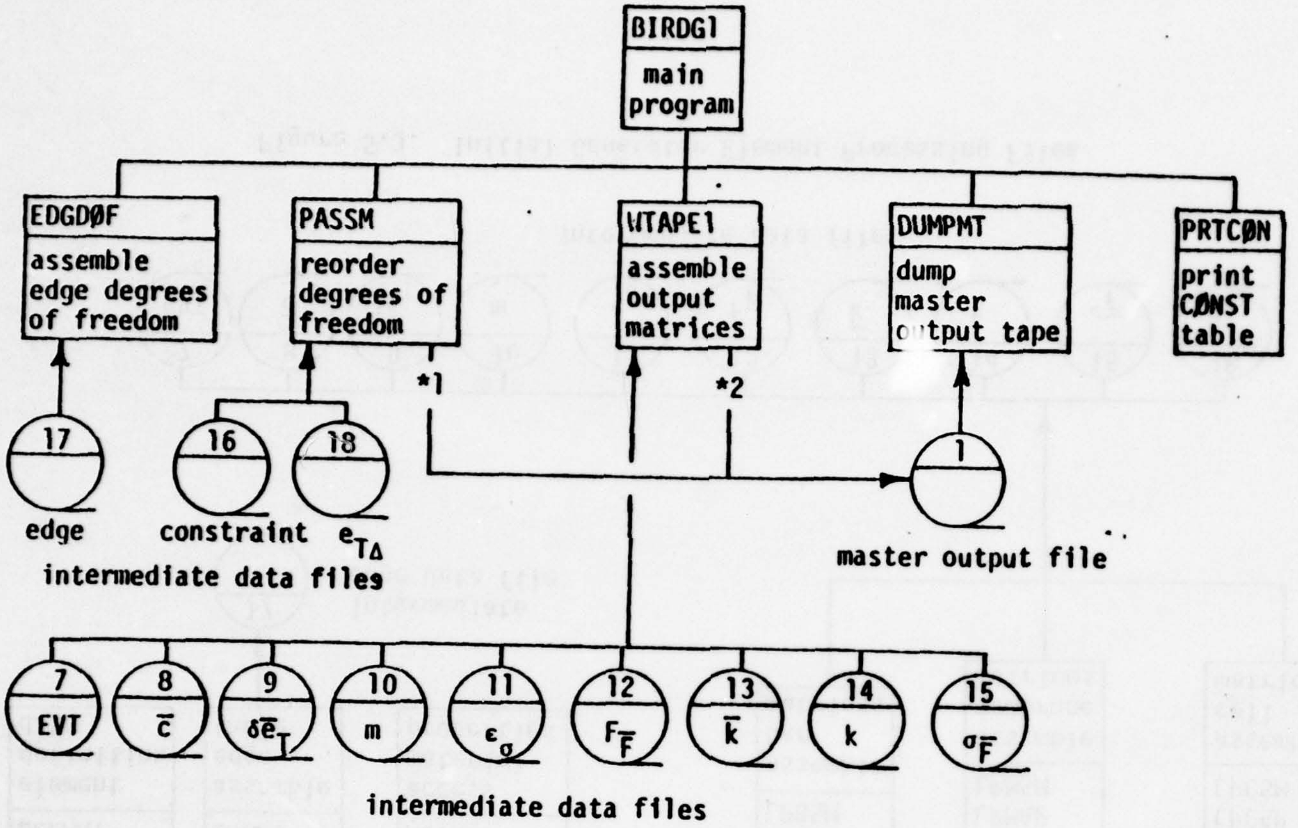


Figure 5.3. Initial Generator Element Processing Files



Notes: *1 = ECT, PRUPT, PRUF in that order

*2 = k, m, $\overline{k}$, $\overline{c}$, $F_{\overline{F}}$, $\sigma_{\overline{F}}$, ε_σ, $\delta\overline{e}_T$, EVT, CØNST in that order

Figure 5.4. Initial Generator Output Assembly Files

File	Content
1	Master output matrix data
7	Intermediate material property and other potentially variable data for each element (EVT)
8	Intermediate unassembled damping matrix for each element ($\bar{c}$)
9	Intermediate deformations due to initial thermal gradients for each element ($\delta\bar{e}_T$)
10	Intermediate unassembled lumped mass matrix for each element (m)
11	Intermediate unassembled strain transformation for each element (ε_σ)
12	Intermediate unassembled force transformation for each element ($F_{\bar{F}}$)
13	Intermediate unassembled stiffness matrix for each element ($\bar{k}$)
14	Intermediate unassembled lumped stiffness matrix for each element (k)
15	Intermediate unassembled stress transformation for each element ($\sigma_{\bar{F}}$)
16	Intermediate constraint data for each constraint
17	Intermediate edge data for each element
18	Intermediate thermal deformation transform for each element ($\bar{e}_{T\Delta}$)

The output of matrices of element data as well as the processing of such data is in the order bars, membranes, cells, and point mass elements. The size of some of the matrices and arrays for individual contributions from the three types of elements is given below.

Matrix or Array	Size	
$\bar{k}$, $\bar{c}$	NK x NK	(symmetric)
k, c, m	NF x NF	(symmetric)
$F_{\bar{F}}$	NF x NK	
$\sigma_{\bar{F}}$	NS x NK	

ε_σ	NS x NS	(symmetric)
$\delta\bar{e}_T$, $\bar{e}_{T\Delta}$	NK x 1	
ECT	NB x 1	
EVT	NV x 1	

where		bar	membrane	cell
	NF =	7	16	36
	NK =	2	9	30
	NS =	1	3	12
	NB =	11	22	54
	NV =	14	15	24

Only the mass matrix, m, and the element constant table, ECT, apply to point mass elements. In this case, NF = 3 and NB = 5.

The master output file is written in a format consistent with FORMAT master input/output matrix tapes. The FORMAT tape name and modifier in the header record is MTAPE, 1. The order and discription of the matrices output for the structural model defined by the tabular input data is given below.

Matrix or Array	FORMAT Matrix Name and Modifier	Row and Column Dimensions	Description
U_o	UZERØ, 1	NJ x 3	joint coordinates
MPT	MPT, 1	NM x 1	material properties
ECT	ECT, 1	100 x NEM	element constants
PRUPT	PRUPT, 1	NU x NT	T dof reordering
PRUF	PRUF, 1	NU x NPF	element force reordering
k	KEL, 1	NPF x NPF	lumped element stiffness
m	MEL, 1	NPF x NPF	lumped element mass
$\bar{k}$	KBEL, 1	NFB x NFB	element stiffness

$\bar{C}$	CBEL, 1	NFB x NFB	element damping
$F_{\bar{F}}$	FFBAR, 1	NFF x NFB	element force transform
$\sigma_{\bar{F}}$	SIGFB, 1	NSS x NFB	element stress transform
ϵ_σ	EPSIG, 1	NSS x NSS	element strain transform
$\delta\bar{e}_T$	DEBT, 1	30 x NEM	element thermal deformation
EVT	EVT, 1	24 x NEM	element variables
CØNST	CØNST, 1	30 x 1	problem constants

where NJ is the number of joints

NM is the length of the material property array

NEM is the number of elements

NU is the number of unconstrained degrees of freedom

NT is the total number of degrees of freedom

NPF is the number of lumped element forces including point mass element

NFB is the number of lumped element forces

NFF is the number of internal element forces

NSS is the number of stresses/strains

and NPF is the only dimension which includes point mass elements. Matrices MPT, ECT, $\delta\bar{e}_T$, EVT and CØNST are mixed integer/floating point arrays which are not usable directly by FORMAT.

CORE UTILIZATION

All internal core storage is by means of labeled common blocks. No blank common is used. Detailed descriptions of each labeled common block is given in Appendix B. Table 5.3 lists each of these labeled common blocks and all routines in the program which reference them.

TABLE 5.3 INITIAL GENERATOR LABELED COMMON REFERENCES

Labeled Common Block	References By Routine
CØFCLC	CØCALC, PRØP
CØNST	BIRDG1, JTEMP, JZERØ, MTLMØD, PASSM, PASSM1, PASSM2, PASSM4, PASSM5, PRTCØN, WTAPE1
IERRØR	BIRDG1, ACCPRP, CNSTRN, CØCALC, DRCNUM, EDGDØF, ELEMNT, JTEMP, JZERØ, LPCKC, MATDES, MTLMØD, ØVER3, PASSM, PASSM1, PASSM2, PASSM4, PASSM5, PRØP, PRTCØN, RAMØSG, WTAPE1
EDØF	EDGDØF, PASSM
IDEN	CØCALC, MATDES, MTLMØD, PRØP
JØRT	BIRDG1, ACCJZE, JTEMP, JTPRT, JZERO, LPBAR, LPBPRT, LPCAP, LPCG, LPCPRT, LPMAP, LPMG, LPMPRT
LIMITS	CØCALC, MATDES, MTLMØD, PRØP
LPBEM	LPBAR, LPBPRT, LPBSM
LPCEM	LPCAP, LPCEBT, LPCK, LPCKC, LPCRM, LPCSM, LPCZM

TABLE 5.3. INITIAL GENERATOR LABELED COMMON REFERENCES (Continued)

Labeled Common Block	References By Routine
LPCV	LPCAP, LPCEBT, LPCFFB, LPCG, LPCKC, LPCPRT, LPCSM
LPEP	BIRDG1, ELEMNT, LPBAR, LPBPRT, LPBSM, LPCAP, LPCC, LPCEBT, LPCFFB, LPCG, LPCKC, LPCPRT, LPCSM, LPMAP, LPMC, LPMFFB, LPMEBT, LPMG, LPMPRT, LPMSM
LPMDSZ	LPMAP, LPMKC, LPMPRT, LPMSZD, LPMS1, LPMS2
LPMEM	LPMAP, LPMFFB, LPMEBT, LPMPRM, LPMSM, LPMSZD, LPMZM
LPMV	LPMAP, LPMFFB, LPMEBT, LPMG, LPMPRT, LPMSM, LPMSZD
MTRL	BIRDG1, ACCPRP, LPBSM, LPCSM, LPMSM, MATDES, MTLMØD, PRØP, RAMØSG
WRK	CNSTRN, DUMPMT, LPCK, LPCKC
ABC	PARTN, QKSØRT, STACK, SWAP

LIMITATIONS

Currently problem size limitations of the program are as follows:

Maximum	Type Data
1200	joints
50	direction cosines
600	constraints
300	oblique constraints
20	materials
10	coefficients or values/material property
660	coefficients for all properties of all materials
240	words of descriptive text for all materials (10 characters/word)
7200	edge element forces
7200	T degrees of freedom
3600	edge degrees of freedom

Limitations with respect to material properties can be modified by changing dimensions of arrays in labeled common block MTRL. At the same time, the data statement in routine MTLMØD initializing the values in labeled common block LIMITS must be changed to be consistent with the array sizes of labeled common block MTRL.

All other limitations are basically functions of the size of labeled common blocks EDØF, IEDGE, and ISEQ. These common regions are used as a group in the reordering of degrees by routines EDGDØF and PASSM and must be of equal size. This size is the limit for number of T degrees of freedom and edge element forces. Half this dimension is the limit for edge degrees of freedom. The current dimensioned size of each of these three common regions is 7200 which is reflected in the list of limitations above.

Labeled common JORT is used to store joint coordinate and temperature tables and is overlayed with labeled common EDØF. JORT requires four equal size partitions whose length is the limit for number of joints. The current implementation with common region EDØF of size 7200 is, therefore, sufficient for 1800 joints. Present dimensions in common region JORT, however, are sized for a limit of 1200 joints.

Limitations on direction cosines and constraints are a function of array sizes in labeled common block WRK as defined in routine CØNSTN. This common region is overlayed with labeled common block ISEQ. The declaration for common region WRK in routine CØNSTN is the largest in the program and uses only 3453 locations of the 7200 abailable. By increasing the array sizes of labeled common WRK as defined in routine CØNSTN but not exceeding the size of labeled common ISEQ, these limitations could be eased without any overall penalty.

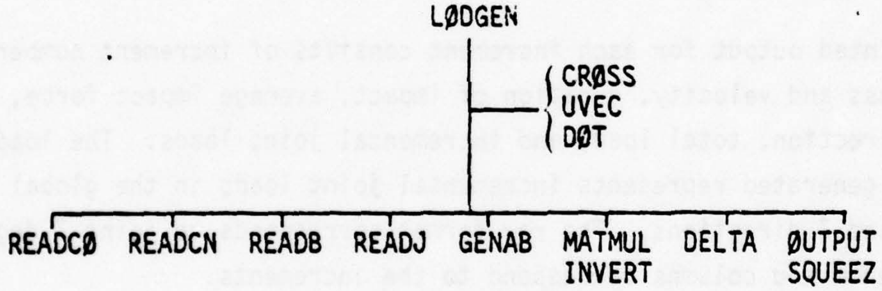
Even though some of the current limitations could, and probably should, be eased without increasing overall core requirements for the program, the array sizes as presently defined, have been sufficient for processing the largest possible problems under the T degrees of freedom limitation.

SECTION VI
LOADS GENERATOR

OVERVIEW

This program accepts user prepared tabular input data describing the model including the impact loading and generates a matrix of incremental point loads. The card input prepared by the user is described in Part 2 of this report. The mathematical formulation is given in Appendix H of Part 1. Detailed descriptions of each routine in this program are given in Appendix C of this document.

The core required for this program without the use of overlay or segmentation is 71000 octal words. The program uses three external files in addition to system print file 6. One file is used for card input, one for card image output, and one for binary matrix data output. Organization of all routines for the program is shown below.



No labeled or blank common regions are used by the program. Array A is dimensioned in the main program which allocates space within the array for use by each of the routines it calls.

Routines CRØSS, UVEC, and DØT perform the vector operations of cross product, unitize, and dot product, respectively. Routines READCØ and READCN read the joint coordinate and impact constant data, respectively.

Routines READB, READJ, GENAB, MATMUL, DELTA, and ØUTPUT are called for each load increment. READB reads the footprint travel and the load factor. READJ reads the numbers of joints that receive load. GENAB generates the A and B matrices used by MATMUL in performing the matrix multiplication $A^T(AA^T)^{-1}BF$ which results in an array of joint loads. INVERT inverts the 3x3 matrix AA^T. DELTA finds the change in joint loads from the previous to the current load increment.

ØUTPUT writes the change in joint loads on the output file(s) using SQUEEZ to compress the data written on the FORMAT tape.

FILE UTILIZATION

Required input to the program are the table of joint coordinates, Data Code 2, and the tables of impact definition data and impact footprint joint data, Data Codes 11 and 12. This data provides: 1) over all model geometry, 2) bird length, mass and velocity, and 3) the incremental load in terms of magnitude, direction and loaded joints.

Printed output for each increment consists of increment number, bird mass and velocity, duration of impact, average impact force, load direction, total load, and incremental joint loads. The load matrix generated represents incremental joint loads in the global X, Y, and Z directions. The row format corresponds to joint T degrees of freedom and columns correspond to the increments.

The output card image file is formatted for use as a FØRMAT Phase II card input matrix with the name DPØT. The binary output file is formatted for use as a FØRMAT Phase II master input matrix tape with the tape name LTAPE,1 and a matrix name of DPØT.

There is one card input file referenced by the integer variable ITAPE. The card image matrix output file is referenced by the integer variable JTAPE. The binary matrix data output file is referenced by the integer variable KTAPE. These three variables, ITAPE, JTAPE and KTAPE are initialized in the main program as files 1, 2 and 3, respectively. All printed output is written to file 6.

LIMITATIONS

The program uses the singularly subscripted array A which is dimensioned in the main program as 10000. The size of this array is stored in the integer variable NSIZEA which is initialized in the main program. The dimension of array A and its size stored in NSIZEA can be changed to accommodate smaller or larger models. The formula for determining the space requirement for array A is:

$$NSIZEA \geq 3*NJT\O T + 9*NJTN + ND\O F$$

where NJTØT is the number of joints under Data Code 2, NJTN is the largest number of joints in any increment under Data Code 12, and NDØF is the number of joint T degrees of freedom from Data Code 11.

SECTION VII
LINEAR INCREMENTAL SOLUTION

OVERVIEW

This program accepts matrix data representing the structural model in mathematical form and solves the linear equations of motion incrementally to obtain the structural response. The solution of the linear equation of motion is a subset of the nonlinear solution as described in Appendix H of Part 1 of this report. The FORMAT steps required to generate the input matrix data for this program as well as the card input required are described in Part 2 of this report. Detailed descriptions of each labeled common block and routine in this program are given in Appendix D of this document.

The core required by the program is problem dependent. Sufficient space must be available in blank common in which all working storage space is dynamically allocated during execution. This requirement is covered in detail in the latter part of this section (see LIMITATIONS). However, for a nominal size problem of 30 or less transformation modes, which is the predominant sizing factor, the program requires 140,000 octal words for execution. The minimum requirement is 104,000 octal words. In addition to the system input/output files 5 and 6, the program requires 22 external files of which 19 are used as scratch, two as master input, and one as master output.

Current Approach

The program as coded is significantly different from the theoretical development given in Part 1 in one area. This involves the introduction of the cosine load variation or impedance matrix and the formation of the A matrix from which all eigenvalues and eigenvectors are extracted to obtain the modal incremental response.

Currently, the program does not include the computation and application of the impedance matrix in the modal response equations. In addition, the equations solved for the modal response are based on an A matrix of the form

$$\left[\begin{array}{c|c} -\bar{K}^{-1}\bar{C} & -\bar{K}^{-1}\bar{M} \\ \hline I & 0 \end{array}\right]$$

where the desired form as given in Part 1, which permits a singular modal stiffness matrix K, is

$$\left[\begin{array}{c|c} 0 & I \\ \hline -\bar{M}^{-1}\bar{K} & -\bar{M}^{-1}\bar{C} \end{array}\right]$$

This inconsistency came about due to changes in theoretical approach which took place during method development. Originally, the form of the A matrix was the latter using $\bar{M}^{-1}$. Subsequently, it was decided to use the form with $\bar{K}^{-1}$ because of singularities in the modal mass matrix in some classical test problems. However, the final approach as documented in Part 1, reverts back to the original form of the A matrix using $\bar{M}^{-1}$. This was made possible by introducing the impedance matrix and automatically accounting for null rows/columns in the modal mass matrix. Unfortunately, coding of the linear solution was nearly complete using the $\bar{K}^{-1}$ form of the A matrix and time did not permit changing the code to be consistent with the current theoretical approach.

Organization

Figure 7.1 shows the functional organization of the program in terms of all labeled common blocks and the principle routines. Labeled common blocks are in parentheses. Vertical positioning in the figure depicts a basic overlay structure which could be implemented in the future.

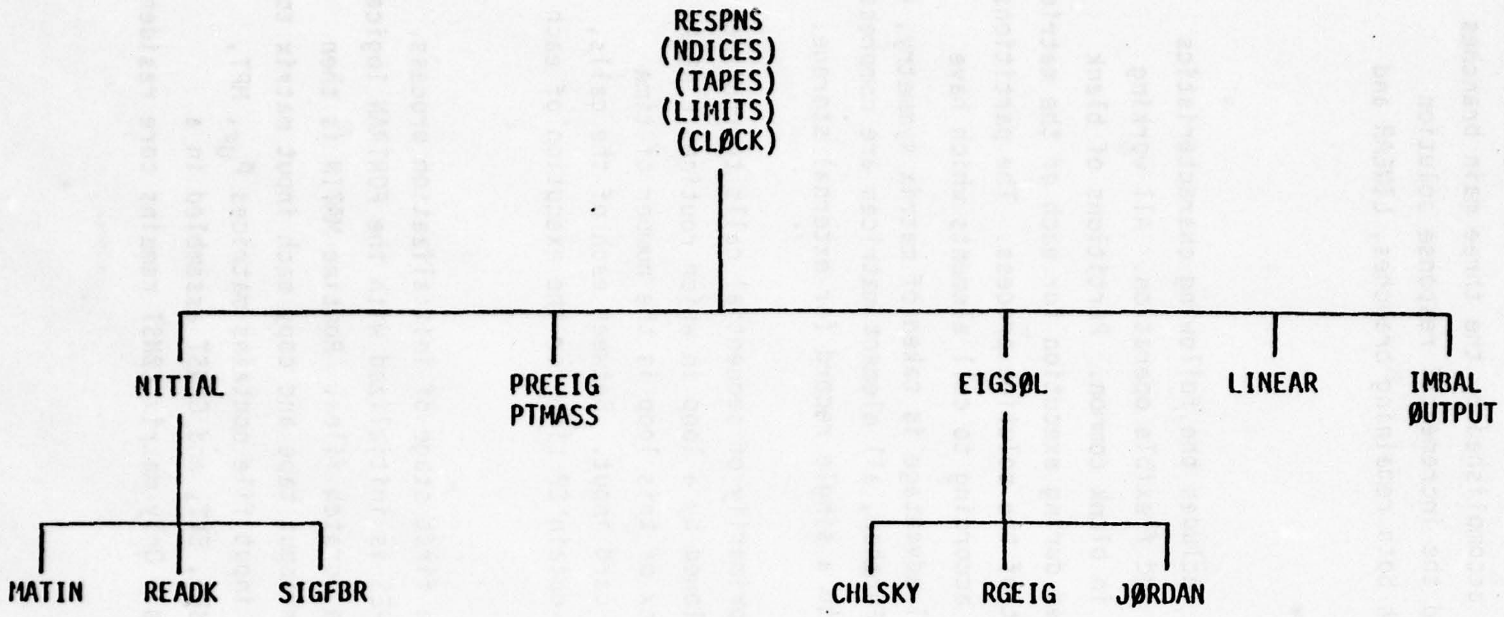


Figure 7.1 Linear Incremental Solution Organization

Functionally, the program can be broken down into two major processing steps; the initialization step accomplished by the three main branches NITIAL, PREEIG, and EIGSØL, and the incremental response solution accomplished by looping through both remaining branches, LINEAR and IMBAL.

Operation

The design of this program includes the following characteristics deemed essential for efficient and flexible operation. All working storage used by the program is in blank common. Partitions of blank common are dynamically allocated during execution for each of the matrices encountered during an increment of the solution process. The partitions for element matrices are sized according to cell elements which have the largest requirements. Full advantage is taken of matrix symmetry, both in storage and computations. Further, all element matrices are compressed if possible and reformatted into a single record for external storage.

The main program consists primarily of sequential calls to routines NITIAL, PREEIG, and EIGSØL followed by a loop in which routines LINEAR and IMBAL are called. The index of this loop is the number of time increments as specified by the card input. Between each of the calls, timing routines are invoked to obtain CP time for the execution of each module.

Routine NITIAL performs the first stage of initialization process. First, labeled common block TAPES is initialized with the FORTRAN logical unit numbers to be used for the scratch files. Routine MATIN is then called to read the first master input tape and copy each input matrix to individual scratch files. The input file contains matrices $\bar{P}_{UF}$, MPT, U_o, ECT, m, $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, $\sigma_{\bar{F}}$, ε_σ, $\delta\bar{e}_T$, EVT, and CØNST assembled in a previously executed FORMAT step. Only matrix CØNST remains core resident.

All card input is then read by routine NITIAL consisting of run parameters and incremental time history. The CØNST array is augmented with the number of modes and time increments from the card input. The first data card is read with a format (4I6, 1L1, 2I6) and contains the following run parameters:

BETA	beginning time interval
NELEMS	total number of physical elements (excludes point mass elements)
NTRVLS	number of time intervals
NG	number of transformation modes
HEAT	"F", dummy logical control flag
NJTS	number of joints
NWØRK	optional extent of blank common

The array NTRVLS, ascending values of elapsed time, is then read with format (6E12.0).

Using the problem sizing information obtained from the matrix and card input, routine NITIAL then begins the process of allocating partitions of the blank common region for each array required in subsequent processing steps. The first location of each of these partitions is stored in labeled common block NDICES. The details of the dynamic allocation scheme used are given later in this section (see CORE UTILIZATION).

Next, the cell stress transform, $\sigma_{s\sigma}$, is initialized by routine NITIAL since it is constant for all cell elements.

Routine READK is then called by NITIAL to rewrite matrices $\overline{k}$, $\overline{c}$, $F_{\overline{F}}$, ϵ_σ, and $\delta\overline{e}_T$ in optimum format. Element partitions of matrices $\overline{k}$, $\overline{c}$, and ϵ_σ are reformatted in upper triangular row-wise form to take advantage of symmetry. The partitions of matrices $\overline{k}$, $\overline{c}$, and $\delta\overline{e}_T$ for each element are then written on a single file as three records. The partition of ϵ_σ is written to a seperate file as one record. Element partitions of matrix $F_{\bar{F}}$ are compressed and written to a seperate file as a single record.

After setting various blank common partitions to zero, NITIAL calls routine SIGFBR to rewrite matrix $\sigma_{\bar{F}}$ in optimum format. Element partitions of $\sigma_{\bar{F}}$ are compressed and written to a seperate file as a single record.

Interspersed in the above initialization steps is the setting of parameters in labeled common block LIMITS and the transcribing of certain input data to the master output file. The data written to the master output file by routine NITIAL are the augmented matrix of problem parameters, CØNST, the matrix of card input incremental time history, TIME, the matrix of original joint coordinates, UZERØ, and the contents of the second master input file, matrices DPBPHI and PBUPTJ, the incremental applied loads and the modal to joint T degree of freedom transform, respectively.

The second stage of initialization is performed by routine PREEIG. This consists of computing the modal stiffness, damping, mass, and loads matrices, $\bar{K}$, $\bar{C}$, $\bar{M}$, and $\delta\bar{P}_{KO}$, respectively. Some of the byproducts of this task are also stored seperately on external files. After first initializing the arrays $\bar{K}$, $\bar{C}$, $\bar{M}$, $\bar{v}$, $\bar{\Delta}$, $\delta\bar{P}_{KO}$, $\bar{P}_{(M)U}$ to zero, the following operations are performed for each physical element in the model.

Read $\bar{k}$, $\bar{c}$, $\delta\bar{e}_T$, m, $F_{\bar{F}}$, and $\bar{P}_{UF}$ from scratch files

$\delta\,\bar{F}_{Ko} = -\,\bar{k}\,\delta\bar{e}_T$

$D = \bar{P}_{UF}\,F_{\bar{F}}$

$DK = D\,\bar{k}$

Write $\delta\bar{F}_{Ko}$ and DK to seperate scratch files

$$\delta\bar{P}_{KO} = \delta\bar{P}_{KO} + D\delta\bar{F}_{Ko}$$

$$\bar{K} = \bar{K} + DK\,D^T$$

$$\bar{C} = \bar{C} + D\,\bar{c}\,D^T$$

$$\bar{M} = \bar{M} + \bar{P}_{UF}\,m\,\bar{P}_{UF}{}^T$$

After all bars, membranes, and cells have been processed in this manner, routine PTMASS is called to add point mass element contributions to matrix $\bar{M}$ if any of these elements are present in the model.

The third and final stage of initialization is performed by routine EIGSØL, the third main branch from the left in Figure 7.1. This consists of the A matrix assembly and subsequent solution for eigenvalues and eigenvectors. Routine CHLSKY is called to obtain the decomposition of K and the A matrix is assembled according to

$$A = \left[\begin{array}{c|c} -\bar{K}^{-1}\bar{C} & -\bar{K}^{-1}\bar{M} \\ \hline I & 0 \end{array}\right]$$

and the eigen equation of the form

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \left[\begin{array}{c|c} H_\Delta & H_v \end{array}\right] = \left[\begin{array}{c|c} H_\Delta & H_v \end{array}\right] \begin{bmatrix} \ddots & & \\ & \frac{1}{\lambda} & \\ & & \ddots \end{bmatrix}$$

is solved for the eigenvalues $H = \left[\begin{array}{c} H_\Delta \\ \hline H_v \end{array}\right]$ and the diagonal matrix of eigenvalues λ_D. The inverse of the eigenvectors is then obtained using routine JØRDAN and both the eigenvectors and their inverse are written to the same scratch file.

The eigenvalue problem is solved by routine RGEIG which is part of the EISPACK library (Reference 16). The version of routine RGEIG in this code has been modified slightly from that in the EISPACK library. In an effort to reduce core space, the origin of the real input matrix was moved n^2 locations down from the origin of the matrix of eigenvectors where n is the order of the problem. This allows for the eigenvectors exclusively in the complex mode.

This concludes the initialization steps. Other than core resident data, input to the incremental solution step consists of matrices $\sigma_{\bar{F}}$, ϵ_{σ}', DK, $\delta\bar{F}_{KO}$, H and H^{-1} residing on scratch files, and matrix $\delta\bar{P}_{(\phi)U}$ on the second master input file.

Executing the incremental solution is accomplished by routines LINEAR and IMBAL. These routines are called in this sequence by the main program for each time increment. Routine LINEAR computes the modal response and routine IMBAL computes the element forces, stresses and strains from the modal response.

Core resident data during the incremental solution which remains constant includes $\bar{K}$, $\bar{K}^{-1}$, $\bar{C}$, $\bar{M}$, λ_D, $\delta\bar{P}_{KO}$ and the cell stress transform $\sigma_{S\sigma}$ previously set in routine NITIAL. Core resident data from the previous increment includes $\bar{P}_{(M)U}$, $\bar{V}$ and $\bar{\Delta}$. Using this data and previously assembled data on external files, routine LINEAR performs the following operations for the increment.

Read $\delta\bar{P}_{(\phi)U}$ from the second master inrut file

Read H and H^{-1} from scratch file

$\bar{P} = \delta\bar{P}_{KO} + \bar{P}_{(M)U} + \delta\bar{P}_{(\phi)U}$

$Z = -K^{-1}\,\bar{P}$

$C_{a_D} = H^{-1}\left[\frac{Z}{\bar{V}}\right]$ diagonalized

$F(\tau) = e^{\lambda\tau}$ where τ is the incremental time

$Q = C_{a_D}\,F(\tau)$

$$\left[\frac{\delta\bar{\Delta}}{\bar{v}}\right] = H\ Q$$

$$\delta\bar{\Delta} = \delta\bar{\Delta} - Z$$

$$\dot{\bar{v}} = H_v\ \lambda_D\ C_{a_D}$$

Routine IMBAL then completes the computation of the modal response by performing the following operations.

$$\bar{P}_{(M)U} = \bar{M}\ \dot{\bar{v}} + \bar{C}\ \bar{v}$$

$$\bar{\Delta} = \bar{\Delta} + \delta\bar{\Delta}$$

This is followed by a call to routine ØUTPUT which writes the modal response to the master output file. The modal response is output as a column matrix with the name RESPNS and a subscript equal to the increment number. The data output consists of $\bar{\Delta}$, $\delta\bar{\Delta}$, $\bar{v}$, $\dot{\bar{v}}$, P_1 and P_2 arranged in that order in a column of length 6*NG where NG is the number of transformation modes. The P_1 and P_2 partitions of the column have no real.significance.

Routine IMBAL then enters a loop for processing each element. This loop processes bar, membrane, and cell elements in that order. A matrix is written to the master output file for each element type that exists in the model. The following operations are performed for each element.

Read $\sigma_{\bar{\bar{F}}}$, ε_σ, DK, and $\delta\bar{\bar{F}}_{Ko}$ from scratch files

$\bar{F}_K = \delta\bar{\bar{F}}_{Ko} - DK^T\bar{\Delta}$

$\sigma = \sigma_{\bar{\bar{F}}}\ \bar{\bar{F}}_K$ for bars and membranes

or $\sigma = \sigma_{s\sigma}\ \sigma_{\bar{\bar{F}}}\ \bar{\bar{F}}_K$ for cells

$\varepsilon = \varepsilon_\sigma\ \sigma$

Write $\bar{\bar{F}}_K$, σ, and ε to master output file

The element forces, stresses, and strains are arranged in that order into a single column of the output matrix for each type of element. The matrix names used for the three element types are BARS, MEMBRN, and CELLS. Each is subscripted with the increment number.

This ends the processing for an increment. Execution of routines LINEAR and IMBAL is repeated for each of the increments specified on the first input data card.

FILE UTILIZATION

All card input is read from the system input file 5 and all printed output is written to the system output file 6. Although FORTRAN logical units 7 and 29 are declared in the PRØGRAM statement in the main program, they are never referenced. FORTRAN logical units 1 through 4 and 8 through 23 are referenced as external files using the variables N1 through N20 stored in labeled common block TAPES (see Appendix D, LABELED COMMON TAPES). File N2 is used as the master output file while N17 and N18 are used for the two master input files. All three master input/output files are formatted consistent with FORMAT master input/output matrix tapes.

The code does not key on the FORMAT tape or matrix names in the input files. Matrix data is identified by its sequential position on the input files. The sequence of matrix data on the first input file, N17, is $\bar{P}_{UF}$, MPT, U_o, ECT, m, $\bar{k}$, $\bar{c}$, $F_{\bar{\bar{F}}}$, $\sigma_{\bar{\bar{F}}}$ ε_{σ}, $\delta\bar{e}_T$, EVT, and CØNST. The sequence of matrix data on the second input file, N18, is $\delta\bar{P}_{(\phi)U}$ and $\bar{P}_{UPTJ}$. All of these matrices are created in the Initial Generator except for $\bar{P}_{UF}$, $\delta\bar{P}_{(\phi)U}$, and $\bar{P}_{UPTJ}$ which are computed in preceding FORMAT steps.

Table 7.1 shows the use of external files by each of the principle routines during the solution process. The routines are listed in their execution sequence. All relevant data stored on external files are listed at the top. In the table, the variable external file references alongside "IN" implies the data is read from that file in the corresponding

TABLE 7.1. LINEAR INCREMENTAL SOLUTION PROCESSING FILES

	Routine		Master Input File 1													Master Input File 2		Internally Generated		
			$\bar{P}_{UF}$	MPT	U_o	ECT	m	$\bar{k}$	$\bar{c}$	$F_{\bar{F}}$	$\sigma_{\bar{F}}$	ϵ_σ	$\delta\bar{e}_T$	EVT	CØNST	$\delta\bar{P}_{(\phi)U}$	$\bar{P}_{UPJ}$	DK	δF_{Ko}	H and H^{-1}
Initialization Steps	NITIAL	IN													N17	N18	N18			
		OUT													core		*			
	MATIN	IN	N17	N17	N17	N17	N17	N17	N17	N17	N17	N17	N17	N17						
		OUT	N11	*	*	*	N10	N8	N14	N4	N12	N19	N7	*						
	READK	IN						N8	N14	N4		N19	N7							
		OUT						N1	N1	N13		N16	N1							
	SIGFBR	IN									N12									
		OUT									N14									
	PREEIG	IN	N11				N10	N1	N1	N13			N1							
		OUT	*				*	*	*	*			*					N7	N15	
	EIGSØL	IN																		
		OUT																		N8
Incremental Loop	LINEAR	IN														N18				N8
		OUT																		
	IMBAL	IN									N14	N16						N7	N15	
		OUT																		

Note * = this data no longer required for remainder of solution.

routine. File references alongside "OUT" imply the data is written to that file in the corresponding routine.

All output to the master output file, N2, occurs in routines NITIAL, IMBAL and routine ØUTPUT which is called by IMBAL. The FORMAT tape name and modifier of the master output is WINDØW, 52877 and is written by NITIAL. A description of each of the output matrices including the routine from which they originate is given below.

	FORMAT	Row and	
	Matrix Name	Column	
Routine	and Modifier	Dimensions	Description
NITIAL	CONST, 1	30 x 1	problem constants
NITIAL	TIME, 1	NTRVLS x 1	incremental time history
NITIAL	UZERØ, 1	NJTS x 3	original joint coordinates
NITIAL	DPBPHI, 1	NG x NTRVLS	incremental transformed applied loads
NITIAL	PBUPTJ, 1	NG x 3*NJTS	modal to joint T degree of freedom transform
ØUTPUT	RESPNS, β	6*NG x 1	modal response
IMBAL	BARS, β	4 x NBAR	bar element response
IMBAL	MEMBRN, β	15 * NMEM	membrane element response
IMBAL	CELLS, β	54 * NCEL	cell element response

where β is the increment number and NBAR, NMEM, and NCEL are the number of bar, membrane, and cell elements, respectively. Note that matrices BARS, MEMBRN, and CELLS will be present only if the corresponding element types are present in the structural model. Also, the names DPBPHI and PBUPTJ may vary since they are user defined in a preceeding FORMAT step. All other matrix names shown above are imbedded in the program code.

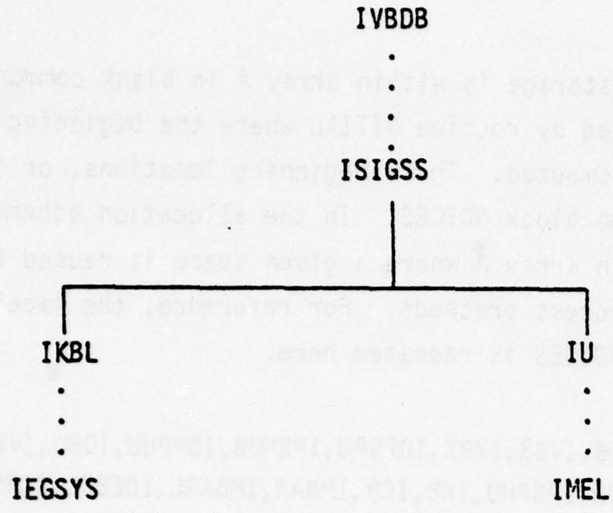
CORE UTILIZATION

All internal core storage is within array A in blank common. The array is implicitly partitioned by routine NITIAL where the beginning locations of each partition are computed. These beginning locations, or indices, are stored in labeled common block NDICES. In the allocation scheme, an overlay structure is used within array A where a given space is reused for different data as the solution process proceeds. For reference, the labeled common block declaration for NDICES is repeated here.

```
COMMØN /NDICES/IVBDB,IVBB,IVBX,IDPBPU,IPBMUB,IDPPUB,IDBL,IVBL
,              IPBAR,IPBPHU,IKB,ICB,IMBAR,IMBARL,IDEBCL,ICØNST
,              ITIME,ISIGSS
,              IKBL,IZ,ICA,IQ,IVAL,IFTAU,IVEC,IEGSYS
,              IU,LU,IECT,IEVT,IMPT,IDEBO,IDFBKO,IDSEB
,              IFK,IFBK,IDEL,IDEDL,IFSFB,IFSFBB,IPBCU
,              IPBKU,ISIGFB,ISIGBH,IEPSIG,IPSLØN,ID,IDK
,              ISKB,ISKBB,ISCB,ISCBB,IPBUF,ISK,ITK,ICIB,IMEL
```

Each integer variable in NDICES is the first location of a partition in array A. For example, A (IVBDB) and A (IVBB-1) point to the first and last locations of the partition used to store matrix $\dot{V}_B$. By definition, the length of the partition would be equal to the number of transformation modes NG.

The variables in NDICES are allocated in such a way as to represent three contiguous members. The members are from IVBDB through ISIGSS, from IKBL through IEGSYS, and from IU through IMEL. The member IVBDB through ISIGSS is permanently resident in array A, while the remaining to members are effectively overlaid. Graphically, the overlay structure can be shown as



where the vertical positioning implies relative location in array A and IVBDB starts at A(1). The member IKBL through IEGSYS is used exclusively by routine EIGSOL for the eigen problem solution. Member IU through IMEL is used by all other parts of the program.

The actual allocation in the present code deviates from this scheme in that the partition associated with IU has been moved to the end of the permanently resident member IVBDB through ISIGSS. This was done so that the joint coordinate data would be permanently core resident. This is no longer a requirement in the linear incremental solution and could be changed to conform to the allocation scheme as shown above.

There are three other labeled common blocks used by the program, LIMITS, TAPES, and CLØCK. Detailed descriptions of each common block are given in Appendix D. Table 7.2 lists each of the common blocks and all routines which reference them.

LIMITATIONS

Sufficient space must be available in blank common array A to accommodate the problem. The size of array A required for a given problem is a function of many parameters. The dimensioned size of array A is 8000 which is stored

TABLE 7.2 LINEAR INCREMENTAL SOLUTION LABELED COMMON REFERENCES

Labeled Common Block	Referenced by Routine
CLØCK	TIMEQ, TSETQ
LIMITS	RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR
NDICES	RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR
TAPES	RESPNS, EIGSØL, IMBAL, LINEAR, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR

in variable NWØRK and initialized by routine NITIAL. Larger sizes can be defined by inputting the appropriate value of NWØRK on the first input data card which is read by NITIAL.

The size of NWØRK must be sufficient to satisfy the following relationships

$$\text{NWØRK} \geq 1200 + 5\text{NM} + 27\text{NG} + \text{NELEMS} + \text{NTRVLS} + 3\text{NJTS} + 8\text{NG}^2$$
$$\text{NWØRK} \geq 6700 + 5\text{NM} + 113\text{NG} + \text{NELEMS} + \text{NTRVLS} + 4\text{NJTS} + \text{NUM}$$

where NG is the number of transformation modes

NM $= (\text{NG}^2 + \text{NG})/2$

NELEMS is the number of physical elements (excludes point mass elements)

NTRVLS is the number of time increments

NJTS is the number of joints

and NUM is the length of the MPT record (row dimension of matrix MPT)

SECTION VIII
NONLINEAR INCREMENTAL SOLUTION

OVERVIEW

This program accepts matrix data representing the structural model in mathematical form and solves the nonlinear equations of motion incrementally to obtain the structural response. The solution of the nonlinear equations is mathematically consistant with the theory developed in Part 1 of this report and the equations summarized in Appendix H of that document. However, the procedures followed by this program are not complete with respect to material nonlinearity nor is the code optimized for geometric nonlinearity. This will be covered in detail later in this section.

The FORMAT steps required to generate the input matrix data for this program as well as the card input required are described in Part 2 of this report. Detailed descriptions of each labeled common block and routine in this program are given in Appendix E of this document.

The core required by the program is problem dependent. Sufficient space must be available in blank common in which all working storage space is dynamically allocated during execution. This requirement is covered in detail in the latter part of this section (see LIMITATIONS). However, for a nominal size problem of 30 or less transformation modes, which is the predominant sizing factor, the program requires 170000 octal words for execution. The minimum requirement is 134000 octal words. In addition to the system input/output files 5 and 6, the program requires 23 external files of which 20 are used as scratch, two as master input, and one as master output.

Development History

The original design of this code was based on a theoretical approach significantly different from that documented in Part 1. The design included strategies for the computations of both geometric and material nonlinearities according to the original theoretical approach.

The theoretical approach in this early stage of development attempted to account for geometric nonlinearity by means of a scalar correction factor to be applied to the linear incremental modal response. The correction factor, $\hat{a}$, was computed as the single unknown of a third order polynomial whose coefficients included the effects of fictitious forces and deformations.

Subsequent application of this code to geometrically nonlinear problems proved unsatisfactory. Several alternate theoretical approaches to geometric nonlinearity were proposed, implemented in the code, tested, and either accepted or rejected. During this time, some code relative to material nonlinearity was deleted from the test versions of the program since 1) the test cases did not include these effects, 2) the material nonlinearity code was incomplete, and 3) because of the change in approach to geometric nonlinearity, the original program design and associated code for processing material nonlinearity was not compatible with the modified code either in an operational or theoretical sense. At this point in time, all executive routines and associated data relative to processing material nonlinearity according to the original theoretical approach were implemented throughout the code. However, the routines necessary for the actual updating of material properties and regeneration of element stiffness, damping, etc. were not yet ready for implementation.

The modifications made to the original code during this development process were intended as temporary changes for testing purposes only. It was assumed that subsequently, the final approach adopted would be implemented as efficiently as possible. However, time constraints did not permit reimplementation of the final approach to geometric nonlinearity nor the completion of the material nonlinearity code. The final version of the code, therefore, reflects this development history and the consequences of time constraints. Under the heading FUTURE DEVELOPMENT at the end of this section, more specific information is provided regarding elimination of some of these shortcomings.

The final version of the code reflects the final theoretical approach adopted for geometric nonlinearity which is that documented in Part 1 of this report. The principle differences between this approach and the original is the iterative solution process for the modal response rather than the $\hat{a}$ approach, and the introduction of the cosine load variation or impedance matrix.

The following discussion of the Nonlinear Incremental Solution program covers those areas of processing consistant with the final theoretical approach to geometric nonlinearity. The actual code, however, still contains many operations based on the original approach to both material and geometric nonlinearity. These areas of the code are described under the heading FUTURE DEVELOPMENT at the end of this section. In addition, the documentation of each routine in Appendix E identifies these areas of code in more detail.

Organization

Figure 8.1 shows the functional organization of the program in terms of all labeled common blocks and the principle routines. Labeled common blocks are in parentheses. Vertical positioning in the figure depicts a basic overlay structure which could be implemented in the future.

Functionally, the program can be broken down into two major processing steps; the initialization step accomplished by the two main branches NITIAL and ØUTPUT, and the incremental solution accomplished by looping through the remaining branches, PREEIG, EIGSØL, FIFDEF, PREBAL, and IMBAL. The iterative solution for the modal response is an inner loop consisting of the EIGSØL and FIFDEF modules.

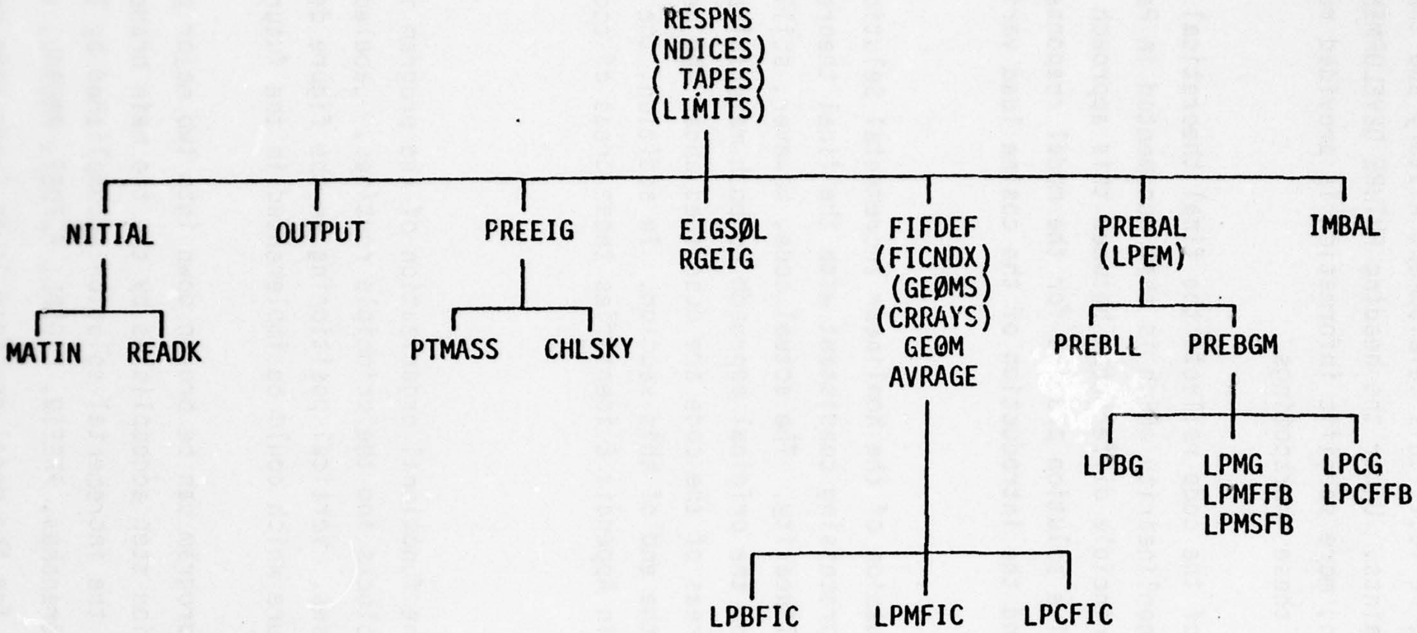


Figure 8.1. Nonlinear Incremental Solution Organization

Operation

The design of this program includes the following characteristics deemed essential for efficient and flexible operation. All working storage used by the program is in blank common. Partitions of blank common are dynamically allocated during execution for each of the matrices encountered during an increment of the solution process. The partitions for element matrices are sized according to cell elements which have the largest requirements. Full advantage is taken of matrix symmetry, both in storage and computations. Further, all element matrices are compressed if possible and reformatted into a single record for external storage.

The main program RESPNS consists of sequential calls to routines NITIAL and ØUTPUT followed by a loop whose index is the number of time increments as specified by the card input. Within this loop, routines PREEIG, EIGSØL, FIFDEF, PREBAL, and IMBAL are called in sequence where EIGSØL and FIFDEF are called repeatedly under control of an inner loop. The index of this inner loop, which is the iterative processing to account for geometric nonlinearity in the solution of the modal response, is a constant initialized as 5.

Routine NITIAL performs the first stage of initialization process. First, labeled common block TAPES is initialized with the FORTRAN logical unit numbers to be used for the scratch files. Routine MATIN is then called to read the first master input tape and copy each input matrix to individual scratch files. The input file contains matrices $\bar{P}_{UF}$, MPT, U_o, ECT, m, $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, $\sigma_{\bar{F}}$, ϵ_{σ}, $\delta\bar{e}_T$, EVT, and CØNST assembled in a previously executed FORMAT step. Only matrices CØNST and U_o remain core resident.

All card input is then read by routine NITIAL consisting of run parameters and incremental time history. The first data card is read with a format (4I6, 1L1, 2I6) and contains the following run parameters:

BETA	beginning time interval
NELEMS	total number of physical elements (excludes point mass elements)
NTRVLS	number of time intervals
NG	number of transformation modes
HEAT	"F", dummy logical control flag
NJTS	number of joints
NWØRK	optional extent of blank common

The array NTRVLS, ascending values of elapsed time, is then read with format (6E12.0).

Using the problem sizing information obtained from the matrix and card input, routine NITIAL then begins the process of allocating partitions of the blank common region for each array required in subsequent processing steps. The first location of each of these partitions is stored in labeled common block NDICES. The details of the dynamic allocation scheme used are given later in this section (see CORE UTILIZATION).

Next, the cell stress transform, $\sigma_{s\sigma}$, is initialized by routine NITIAL since it is constant for all cell elements.

Routine READK is then called by NITIAL to rewrite matrices $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, ε_σ, and $\delta\bar{e}_T$ in optimum format. Element partitions of matrices $\bar{k}$, $\bar{c}$, and ε_σ are reformatted in upper triangular row-wise form to take advantage of symmetry. The partitions of matrices $\bar{k}$, $\bar{c}$, and $\delta\bar{e}_T$ for each element are then written on a single file as three records. The partition of ε_σ is written to a separate file as one record. Element partitions of matrix $F_{\bar{F}}$ are compressed and written to a separate file as a single record.

NITIAL then reads the original joint coordinate data into core from the scratch file where it had been written by routine MATIN. Before exiting NITIAL, various blank common partitions are initialized to zero including those for $\bar{v}$, $\dot{\bar{v}}$, $\delta\bar{P}_U$, $\bar{P}_{(M)U}$, $\delta\bar{P}_{(\phi)U}$, and $\delta\bar{E}_{CL}$.

Interspersed in the above initialization steps is the setting of problem parameter variables in labeled common block LIMITS.

Routine ØUTPUT transcribes certain input matrix data to the master output file. After augmenting array CØNST with the number of modes and time increments from the card input, the data written to the master output file are the augmented matrix of problem parameters, CØNST, the matrix of card input incremental time history, TIME, the matrix of original joint coordinates, UZERØ, and the contents of the second master input file, matrices DPBPHI and PBUPTJ, the incremental applied loads and the modal to joint T degree of freedom transform, respectively.

This concludes the initialization process. All data passed to the incremental solution process is on external files except for the arrays of incremental times, original joint coordinates, and problem constants. During the incremental solution, it is assumed that the element stiffness and damping remain constant and that no initial element deformations from thermal or other causes are present.

Incremental solution processing begins with the execution of routine PREEIG. First, the transformed incremental applied loads for the increment, if any, are read from the second master input file. The matrix of transformed loads $\bar{P}$ is then partially assembled according to

$$\bar{P} = \delta\bar{P}_U - \bar{P}_{(M)U} + \delta\bar{P}_{(\phi)U}$$

where $\delta\bar{P}_U$ are the unbalanced forces from the previous increment, $\bar{P}_{(M)U}$ are the inertia forces from the previous increment, and $\delta\bar{P}_{(\phi)U}$ are the applied loads.

A loop is then entered whose index is the number of elements in the structural model. The following operations are performed for each element.

```
Read $\bar{k}$, $\bar{c}$, and $\delta\bar{e}_T$ from scratch file
$\delta\bar{F}_{Ko} = - \bar{k}\ \delta\bar{e}_T$
Read D from scratch file ($D = \bar{P}_{UF}\ F_{\bar{F}}$)
$\bar{P} = \bar{P} + D\ \delta\bar{F}_{Ko}$
$DK = D\ \bar{k}$
$\bar{K} = \bar{K} + DK\ D^T$
$\bar{C} = \bar{C} + D\ \bar{c}\ D^T$
Read $\bar{F}_K$ from scratch file
$FK = \bar{F}_K + \delta\bar{F}_{Ko}$
Write DK, $\bar{k}$ and FK to scratch file
```

After all bars, membranes, and cell elements have been processed, the assembly of the transformed loads is completed by subtracting the modal damping forces of the previous increment $\bar{P}_{(C)U}$ from the partially assembled loads $\bar{P}$.

The sequence of operations above is slightly different on the initial pass for the first increment. Matrix D is computed from matrices $\bar{P}_{UF}$ and $F_{\bar{F}}$ read from scratch files and then matrix D is written to a separate scratch file. Matrix $\bar{F}_K$, the element forces from the previous increment, is initialized to zero rather than read from a scratch file. Matrix $\bar{M}$ is computed according to

$$\bar{M} = \bar{M} + \bar{P}_{UF}\ m\ \bar{P}_{UF}{}^T$$

where $\bar{M}$ is implicitly initialized to zero as are $\bar{K}$ and $\bar{C}$ in the sequence above. After all bar, membrane, and cell elements have been processed, routine PTMASS is called by PREEIG to add contributions from point mass elements, if any, to matrix $\bar{M}$. Finally, any null rows/columns of $\bar{M}$ are augmented on the diagonal with a value equal to 1×10^{-4} of the root mean square of all non zero diagonal elements of the matrix. Routine CHLSKY is then called to decompose $\bar{M}$. The decomposition of this matrix then remains core resident for the remainder of the run.

The inner loop consisting of routines EIGSØL and FIFDEF is then executed for 5 iterations. Routine EIGSØL assembles the A matrix for which the eigenvalues, λ, and the eigenvectors, H, are obtained. Routine RGEIG, which is part of the EISPACK library (Reference 16), is called for solution of the eigen problem. The version of routine RGEIG in this code has been modified slightly from that in the EISPACK library. In an effort to reduce core space, the origin of the real input matrix was moved n^2 locations down from the origin of the matrix of eigenvectors where n is the order of the problem. This allows for the eigenvectors exclusively in the complex mode.

Input to EIGSØL are the core resident transformed modal stiffness, damping, decomposed mass, and loads matrices $\bar{K}$, $\bar{C}$, $\bar{M}^{-1}$, and $\bar{P}$, respectively. Also input for all iterations other than the first are the equivalent loads derived from fictitious force and deformation effects computed by routine FIFDEF which are also core resident. Output from EIGSØL are the modal displacements, velocities, and accelerations, $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$, respectively. The equations for solution of the modal response are given in Appendix H of Part 1 (H.161 through H.175).

Routine FIFDEF is then executed to obtain the effects of fictitious forces and deformations. Using the modal response computed by routine EIGSØL, and element forces due to fictitious deformations computed by FIFDEF in the previous incrment, FIFDEF processes each element individually to determine the element deformations, temporarily updated joint coordinate data, and compute element forces. This data is then used to solve for the element fictitious forces and deformations from which the equivalent transformed loads and element forces due to fictitious deformations are derived for use in the next iteration. The element forces due to fictitious deformations are output to a scratch file for use by FIFDEF in the next iteration.

Routines GEØM and AVRAGE are called by FIFDEF to temporarily update geometry and average geometry, respectively. Routines LPBFIC, LPMFIC, and LPCFIC are called by FIFDEF to compute fictitious force and deformation effects from bar, membrane, and cell elements, respectively.

Routines EIGSØL AND FIFDEF are then executed again for the next iteration until 5 iterative steps have been completed. At the end of the iterative process, the final element forces and deformations from FIFDEF are passed on a scratch files to routine PREBAL, the next step in the incremental solution. The final modal response from EIGSØL consisting of the incremental displacements $\delta\Delta$, the cumulative velocities $\bar{v}$, and the cumulative accelerations $\ddot{\bar{v}}$ are core resident.

Routine PREBAL permanently updates joint coordinates, computes the incremental element forces, and regenerates matrices $F_{\bar{\bar{F}}}$ and $\sigma_{\bar{\bar{F}}}$ based on the new geometry. Before processing each element to accomplish these tasks, the transformed modal inertia forces are updated based on the accelerations at the end of the increment according to

$$\bar{P}_{(M)U} = - \bar{M} \ddot{\bar{v}}$$

Following this, the implicit initialization of the arrays for $\bar{P}_{(C)U}$ and $\bar{P}_{(K)U}$ to zero, and a call to routine PREBLL to perform some intermediate allocation of work space, the following operations are performed for each element.

Read $\bar{P}_{UF}$, $F_{\bar{\bar{F}}_{\beta-1}}$, ECT, δe and $\delta\dot{e}$ from scratch files

Call routine PREBGM to update coordinates and regenerate $F_{\bar{\bar{F}}}$ and $\sigma_{\bar{\bar{F}}}$

$D = \bar{P}_{UF} \; F_{\bar{\bar{F}}}$

Read $\bar{F}_K$ and $\bar{c}$ from scratch files

$\bar{P}_{(C)U} = \bar{P}_{(C)U} + D\,\bar{c}\,F_{\bar{\bar{F}}_{\beta-1}}\,\delta\dot{e}$

Read $\bar{\bar{F}}_{K_{\beta-1}}$ from scratch file

$\delta\bar{\bar{F}}_K = \bar{\bar{F}}_K - \bar{\bar{F}}_{K_{\beta-1}}$

$\bar{P}_{(K)U} = \bar{P}_{(K)U} + D\,\bar{F}_K$

Write D, $F_{\bar{\bar{F}}}$, $\delta\bar{F}_K$, and $\sigma_{\bar{\bar{F}}}$ to scratch files

The last step in processing for an increment is the execution of routine IMBAL. First, the core resident matrices for cumulative transformed modal applied loads and displacements are augmented with their incremental components according to

$$\bar{P}_{(\phi)U} = \bar{P}_{(\phi)U} + \delta\bar{P}_{(\phi)U}$$

$$\bar{\Delta} = \bar{\Delta} + \delta\bar{\Delta}$$

Next, routine IMBAL writes the model response to the master output file. The modal response is output as a column matrix with the name RESPNS and a subscript equal to the increment number. The data output consists of $\bar{\Delta}$, $\delta\bar{\Delta}$, $\bar{v}$, $\dot{\bar{v}}$, P_1 and P_2 arranged in that order in a column of length 6*NG where NG is the number of transformation modes. The P_1 and P_2 partitions of the column have no real significance.

Routine IMBAL then enters a loop for processing each element. This loop processes bar, membrane, and cell elements in that order. A matrix is written to the master output file for each element type that exists in the model. The following operations are performed for each element.

Read $\sigma_{\bar{\bar{F}}}$, ϵ_σ, $\bar{\bar{F}}_K$, $\epsilon_{\beta-1}$ and $\delta\bar{\bar{F}}_K$ from scratch files

$\sigma = \sigma_{\bar{\bar{F}}}\,\bar{\bar{F}}_K$ for bars and membranes

or $\sigma = \sigma_{s\sigma}\,\sigma_{\bar{\bar{F}}}\,\bar{\bar{F}}_K$ for cells

$\varepsilon = \varepsilon_{\beta-1} + \varepsilon_{\sigma} \sigma_{\bar{F}} \delta\bar{F}_K$

Write $\bar{F}_K$, σ, and ε to master output file

The element forces, stresses, and strains are arranged in that order into a single column of the output matrix for each type of element. The matrix names used for the three element types are BARS, MEMBRN, and CELLS. Each is subscripted with the increment number.

This ends the processing for an increment. Execution of routines PREEIG, EIGSØL, FIFDEF, PREBAL, and IMBAL is repeated for each of the increments specified on the first input data card.

FILE UTILIZATION

All card input is read from the system input file 5 and all printed output is written to the system output file 6. Although FORTRAN logical unit 7 is declered in the PRØGRAM statement in the main program, it is never referenced. FORTRAN logical units 1 through 4 and 8 through 23 are referenced as external files using the variables N1 through N20 stored in labeled common block TAPES (see Appendix E, LABELED COMMON TAPES). Files N30 and N31, FORTRAN logical units 30 and 31, respectively are used by routine FIFDEF exclusively for passing data during the iterative solution of the modal response. File N23 is used as the master output file while N17 and N18 are used for the two master input files. All three master input/output files formatted consistent with FORMAT master input/output matrix tapes.

The code does not key on the FORMAT tape or matrix names in the input files. Matrix data is identified by its sequential position on the input files. The sequence of matrix data on the first input file, N17, is $\bar{P}_{UF}$, MPT, U_o, ECT, m, $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, $\sigma_{\bar{F}}$ ε_{σ}, $\delta\bar{e}_T$, EVT, and CØNST. The sequence of matrix data on the second input file, N18, is $\delta\bar{P}_{(\phi)U}$ and $\bar{P}_{UPTJ}$. All of these matrices are created in the Initial Generator except for $\bar{P}_{UF}$, $\delta\bar{P}_{(\phi)U}$, and $\bar{P}_{UPTJ}$ which are computed in preceding FORMAT steps.

Tables 8.1 and 8.2 show the use of external files by each of the principle routines during the initialization and solution processes, respectively. The routines are listed in their execution sequence. All relevant data stored on external files are listed at the top. In the tables, the variable external file references alongside "IN" implies the data is read from that file in the corresponding routine. File references alongside "OUT" imply the data is written to that file in the corresponding routine.

The file utilization shown in Table 8.1 is for a typical increment and does not apply to the first increment which includes some initialization processing not shown.

At the end of an increment, the absolute designations of the external files listed in Table 8.2 under Last Increment and Current Increment are interchanged so that the variable file references remain unchanged for the next increment. The files "flip-flopped" in this manner are

N13 and N14
N3 and N19
N4 and N5
N9 and N10

During the iterative solution for modal response, i.e. looping through routines EIGSØL and FIFDEF, routine FIFDEF passes the element forces due to fictitious deformations on files N30 and N31. File N30 contains element forces from the last iteration while file N31 is output with element forces from the current iteration. These files are "flip-flopped" at each iteration.

All output to the master file, N23, occurs in routines ØUTPUT and IMBAL. The FORMAT tape name and modifier of the master output is TAPE, 1 and is written by ØUTPUT. A description of each of the output matrices including the routine from which they originate follows on the next page.

TABLE 8.1. NONLINEAR INCREMENTAL INITIALIZATION PROCESSING FILES

Routine		Master Input File 1													Master Input File 2	
		P_{UF}	MPT	U_o	EĊT	m	$\bar{k}$	$\bar{c}$	$F_{\bar{F}}$	$\sigma_{\bar{F}}$	ϵ_σ	$\delta\bar{e}_T$	EVT	CØNST	$\delta\bar{P}_{(\phi)U}$	$\bar{P}_{UPJ}$
NITIAL	IN ØUT													N18 Core		
MATIN	IN ØUT	N17 N11	N17 *	N17 Core	N17 N12	N17 N10	N17 N8	N17 N14	N17 N4	N17 *	N17 N19	N17 N2	N17 *			
READK	IN ØUT						N8 N1	N14 N1	N4 N13		N19 N16	N2 N1				
ØUTPUT	IN ØUT														N18	N18 *

Notes: 1) * means this data no longer required for remainder of solution

2) In the original version of the code, the first master input file, N17, is subsequently used as a scratch file by routine PREBALT in processing material nonlinearity. Consequently, the first master input file should always be a temporary file if the original code for material nonlinearity is reactivated.

TABLE 8.2. NONLINEAR INCREMENTAL SOLUTION PROCESSING FILES

Routine		Constant Data				Data from Last Increment (β-1)				Data for Current Increment (β)				Intermediate Data					Master Input Data
		$\bar{P}_{UF}$	ECT	$\bar{k}$ $\bar{c}$ $\delta\bar{e}_T$	ε_σ	$F_{\bar{F}}$	D	$\bar{F}_K$	ε	$F_{\bar{F}}$	D	$\bar{F}_K$	ε	DK $\bar{k}$ FK	$\sigma\bar{F}$ $\delta\bar{F}_K$	H	$\delta\bar{F}_{K_f}$	δe $\delta\dot{e}$	$\delta\bar{P}_{(\phi)U}$
PREEIG	IN			N1			N3	N4											N18
	ØUT													N6					
EIGSØL	IN																		
	ØUT															N8			
FIFDEF	IN	N11	N12			N13								N6			N30		
	ØUT											N5					N31	N7	
PREBAL	IN	N11	N12	N1		N13		N4				N5						N7	
	ØUT									N14	N19				N6				
INBAL	IN				N16				N9			N5			N6				
	ØUT												N10						

Routine	FORMAT Matrix Name and Modifier	Row and Column Dimensions	Description
ØUTPUT	CØNST, 1	30 x 1	problem constants
ØUTPUT	TIME, 1	NTRVLS x 1	incremental time history
ØUTPUT	UZERØ, 1	NJTS x 3	original joint coordinates
ØUTPUT	DPBPHI, 1	NG x NTRVLS	incremental transformed applied loads
ØUTPUT	PBUPTJ, 1	NG x 3*NJTS	modal to joint T degree of freedom transform
IMBAL	RESPNS, β	6*NG x 1	modal response
IMBAL	BARS, β	4 x NBAR	bar element response
IMBAL	MEMBRN, β	15 * NMEM	membrane element response
IMBAL	CELLS, β	54 * NCEL	cell element response

where β is the increment number and NBAR, NMEM, and NCEL are the number of bar, membrane, and cell elements, respectively. Note that matrices BARS, MEMBRN, and CELLS will be present only if the corresponding element types are present in the structural model. Also, the names DPBPHI and PBUPTJ may vary since they are user defined in a preceeding FORMAT step. All other matrix names shown above are imbedded in the program code.

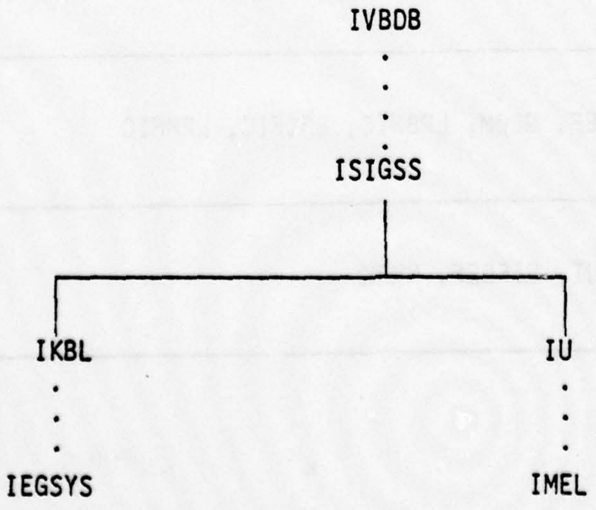
CORE UTILIZATION

All internal core storage is within array A in blank common. The array is implicitly partitioned by routine NITIAL where the beginning locations of each partition are computed. These beginning locations, or indices, are stored in labeled common block NDICES. In the allocation scheme, an overlay structure is used within array A where a given space is reused for different data as the solution process proceeds. For reference, the labeled common block declaration for NDICES is repeated here.

```
COMMON /NDICES/IVBDB,IVBB,IVBX,IDPBPU,IPBMUB,IDPPUB,IDBL,IVBL
,              IPBAR,IPBPHU,IKB,ICB,IMBAR,IMBARL,IDEBCL,ICØNST
,              ITIME,ISIGSS
,              IKBL,IZ,ICA,IQ,IVAL,IFTAU,IVEC,IEGSYS
,              IU,LU,IECT,IEVT,IMPT,IDEBO,IDFBKO,IDSEB
,              IFK,IFBK,IDEL,IDEDL,IFSFB,IFSFBB,IPBCU
,              IPBKU,ISIGFB,ISIGBH,IEPSIG,IPSLØN,ID,IDK
,              ISKB,ISKBB,ISCB,ISCBB,IPBUF,ISK,ITK,ICIB,IMEL
```

Each integer variable in NDICES is the first location of a partition in array A. For example, A (IVBDB) and A (IVBB-1) point to the first and last locations of the partition used to store matrix $\dot{v}_B$. By definition, the length of the partition would be equal to the number of transformation modes NG.

The variables in NDICES are allocated in such a way as to represent three contiguous members. The members are from IVBDB through ISIGSS, from IKBL through IEGSYS, and from IU through IMEL. The member IVBDB through ISIGSS is permanently resident in array A, while the remaining two members are effectively overlaid. Graphically, the overlay structure can be shown as



AD-A063 741   DOUGLAS AIRCRAFT CO LONG BEACH CALIF   F/G 1/3

AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)

DEC 77   R C MORRIS   F33615-75-C-3105

UNCLASSIFIED   MDC-J-7174-PT-3   AFFDL-TR-77-99-PT-3   NL

2 OF 6
ADA
063741

TABLE 8.3. NONLINEAR INCREMENTAL SOLUTION LABELED COMMON REFERENCES

Labeled Common Block	References by Routine
NDICES	RESPNS, AVRAGE, CHKØUT, CHKPNT, EIGSØL, FIFDEF, FØRMAT, GEØM, IMBAL, NDXSET, NITIAL, ØUTPUT, PØLSØL, PBARUF, PREBAL, PREBGM, PREBGU, PREBLL, PREBST, PREEIG, PTMASS, READK
LIMITS	RESPNS, AVRAGE, CHKØUT, CHKPNT, EIGSØL, FIFDEF, FØRMAT, IMBAL, NDXSET, NITIAL, NKRK, ØUTPUT, PBARUF, PØLSØL, PREBAL, PREBGM, PREBGU, PREBLL, PREBST, PREEIG, PTMASS, READK
TAPES	RESPNS, CHKPNT, EIGSØL, FIFDEF, IMBAL, NITIAL, ØUTPUT, PBARUF, PREBAL, PREBGM, PREBGU, PREBLL, PREBST, PREEIG, PTMASS, READK
LPEM	CHKPNT, LPBG, LPCFFB, LPCG, LPCSFB, LPMFFB, LPMG, LPMSZD, LPMS1 LPMS2, ØUTPUT, PREBAL, PREBGM, PREBGU, PREBLL, PREBST
FICNDX	CHKØUT, EDGUNV, FIFDEF, GEØM, NKRK
GEØMS	EDGUNV, FIFDEF, GEØM, LPBFIC, LPCFIC, LPMFIC
CRRAYS	AVRAGE, CHKØUT, FIFDEF, NKRK

where the vertical positioning implies relative location in array A and IVBDB starts at A(1). The member IKBL through IEGSYS is used exclusively by routine EIGSØL for the eigen problem solution. Member IU through IMEL is used by all other parts of the program.

The actual allocation in the present code deviates from this scheme in that the partition associated with IU has been moved to the end of the permanently resident member IVBDB through ISIGSS. This was done so that the joint coordinate data would be permanently core resident.

There are six other labeled common blocks used by the program, LIMITS, TAPES, LPEM, FICNDX, GEØMS, and GRRAYS. Detailed descriptions of each common block are given in Appendix E. Table 8.3 lists each of the common blocks and all routines which reference them.

LIMITATIONS

Sufficient space must be available in blank common array A to accommodate the problem. The size of array A required for a given problem is a function of many parameters. The dimensioned size of array A is 8000 which is stored in variable NWØRK and initialized by routine NITIAL. Larger sizes can be defined by inputting the appropriate value of NWØRK on the first input data card which is read by NITIAL.

The size of NWØRK must be sufficient to satisfy the following relationships

$$\text{NWØRK} \geq 1200 + 5\text{NM} + 27\text{NG} + \text{NELEMS} + \text{NTRVLS} + 3\text{NJTS} + 8\text{NG}^2$$

$$\text{NWØRK} \geq 6700 + 5\text{NM} + 113\text{NG} + \text{NELEMS} + \text{NTRVLS} + 4\text{NJTS} + \text{NUM}$$

where NG is the number of transformation modes

NM $=(\text{NG}^2 + \text{NG})/2$

NELEMS is the number of physical elements (excludes point mass elements)

NTRVLS is the number of time increments
NJTS is the number of joints
and NUM is the length of the MPT record (row dimension of matrix MPT)

FUTURE DEVELOPMENT

In the preceding description of the final version of the Nonlinear Incremental Solution, a number of processing operations designed for the original theoretical approach were not discussed. These will be covered here as a guide in future development, provide more insight to the code as it exists, and to promote salvaging of code where possible.

The modifications necessary to progress from the original version of the code to the final version were grouped into three UPDATE steps. Table 8.4 lists each of the decks modified in each of the three steps as well as the UPDATE identification name associated with subsets of these modifications. A description of the changes made is also provided. These modifications are a mixture of corrections to errors, supplements to capability, changes in order to implement the final approach to geometric nonlinearity, and deletions of some material nonlinearity code.

In order to preserve the original design concepts and provide a record of program development, the original source code and the three sets of deck modifications as well as the final source code have been included as part of the program delivery to the contractor.

Applicability of Original Design Concepts

In order to understand the Nonlinear Incremental Solution code in its final form, some background information regarding the original theoretical approach is necessary.

The original theoretical approach to geometric nonlinearity was to scale the linear incremental model response by a factor $\hat{a}$ computed as a function of fictitious force and deformation effects during the increment

TABLE 8.4. ROUTINES MODIFIED DURING NONLINEAR INCREMENTAL SOLUTION DEVELOPMENT

STEP	*IDENT	DECK	DESCRIPTION
1	CHK	FIFDEF GEØM AVRAGE NKRK	Suppress intermediate printing from fictitious force module designed into code for check out purposes only.
	FIXUP	CHKØUT FIFDEF AVRAGE NDICES ARRAYS LPBFIC VECT	Error corrections for fictitious force module.
	ERRPRT	FIFDEF GEØM EDGUNV LPBFIC AVRAGE	Delete intermediate printing from fictitious force module introduced into code for debugging during initial testing.
	TEMP1	RESPNS IMBAL PØLSØL PREEIG READK NITIAL HALT	Add routine HALT, modify intermediate printing, error corrections, and modify imbalance test.
	TEMP2	PREMBAL IMBAL EIGSØL	Modify intermediate printing, introduce output data for intermediate postprocessor, and delete portion of code in PREBAR relative to material nonlinearity.

TABLE 8.4. ROUTINES MODIFIED DURING NONLINEAR INCREMENTAL SOLUTION DEVELOPMENT (Continued)

STEP	*IDENT	DECK	DESCRIPTION
2	TEMP3	RESPNS PREBLL CHKPNT FIFDEF PREEIG NITIAL LPBFIC IMBAL EIGSØL PREBAL JØRDAN	Modify intermediate print, error corrections, remove some problem dependent code, replace routine JØRDAN, and introduce iterative approach to geometric nonlinearity, cosine load variation, and augmentation of $\overline{M}$ null rows and columns with diagonal terms having small values.
3	TEMP4	RESPNS IMBAL NITIAL PREEIG PREBAL READK ØUTPUT PTMASS	Add routine PTMASS, accommodate point mass elements, delete preliminary postprocessor output, process U_o as an NJ x 3 on external files, introduce output data generation consistant with final postprocessor.

(see History of Development at the beginning of this section). No processing of an iterative nature was involved here. However, in accounting for material nonlinearity, a second iteration through the entire incremental solution process was required. For this solution, average modal stiffness and damping matrices, $\bar{K}$ and $\bar{C}$, were used. The averaging was done using the initial $\bar{K}$ and $\bar{C}$ matrices at the beginning of the increment and the $\bar{K}$ and $\bar{C}$ computed at the end of the first iteration which included the effects of changes in the material properties of any elements due to plasticity and the failure of any elements that occurred in the first solution. The criteria for determining the necessity of this second iteration was the equilibrium imbalance test at end of the first solution.

For the second iterative solution, some data computed in the first iteration was passed on external files since it did not require regeneration. This included the element stiffness and damping matrices and some geometric data based on initial geometry for the increment generated for the computation of fictitious forces and deformations.

In the code, the variable denoting that the second iterative pass is being processed is called BETBAR for $\bar{\beta}$. This flag is used to control which operations of the complete solution process are to be executed during the second iteration. Array A (IDEBCL), which is core resident, is used to flag those elements that fail during the solution process. This array is primarily used to store the dissipated damping energy due to temperature change in each element, $\delta\bar{E}_{CL}$ (always positive). Elements which had failed during previous increments were flagged by a -10 and elements failing during the current increment were flagged as -1. The values in array A (IDEBCL) are tested at various points in the code to detect failed elements and omit processing for such elements.

The input matrix EVT was intended as a data file for those element parameters which may vary during the solution process. These parameters included element temperature, current material properties, stress/strain state in terms of equivalent stress and stress, increment at which last update occurred, increment at which element failed, etc.

The allocation of all data to external files for this original program design is given in Table 8.5. The following data quantities listed in the table are redefined here for clarification.

$$\begin{aligned} D &= \bar{P}_{UF}\, F_{\bar{F}} \\ DK &= D\, \bar{k} \\ FK &= \bar{F}_{K_{\beta-1}} + \delta\bar{F}_{K_{o}} \\ \bar{C}\bar{I} &= D\, \bar{c}\, D^{T} \\ D_{o}, G_{f} &= \text{geometric data for the generation of fictitious forces and deformations} \end{aligned}$$

where each of the above is computed individually for each element.

Table 8.6 shows the intended use of these files during a typical increment of the solution process. Certain initializing operations are performed in the first increment which are not shown here. The 20 external files used are listed across the top while the major modules appear in their execution sequence at the left.

This information should provide more insight to the code, the reasoning behind some operations still present there, and reduce the effort required for future modifications. The basic design features of this program enhance its adaptability to changes in methodology. These features include the library of utility routines to perform a wide variety of incore matrix operations and the partitioning of blank common for required array space.

Some other features of the code which should be pointed out are the intermediate print capabilities in routines CHKØUT and CHKPNT. Routine CHKØUT is used exclusively in the FIFDEF module to output intermediate calculations at various predetermined points in the computations. In

TABLE 8.5. NONLINEAR INCREMENTAL SOLUTION ORIGINAL DESIGN FILE ALLOCATION

Increment			Resident Data
β-1 Files	β Files	$\bar{\beta}$ Files	
N1	N2		$(\bar{k}_i, \bar{c}_i, \delta\bar{e}_{Ti}, i=1,NELM)$
N3	N19	N19	$(D_i, i=1,NELM), (D_{O_i}, G_{f_i}, i=1,NELM)$
N4	N5	N5	$(\bar{\bar{F}}_{k_i}, i=1,NELM)$
	N6		$(DK_i, \overline{CI}_i, FK_i, i=1,NELM)$
	N6		$(\sigma_{\bar{\bar{F}}_i}, \delta\bar{\bar{F}}_{K_i}, i=1,NELM)$
		N6	$(DK_i, i=1,NELM)$
		N6	$(\sigma_{\bar{\bar{F}}_i}, \delta\bar{\bar{F}}_{K_i}, i=1,NELM)$
			$(\delta e_i, \delta\dot{e}_i, i=1,NELM)$
		N8	$(\delta e_i, \delta\dot{e}_i, i=1,NELM)$
N9	N10		$(EVT_i, i=1,NELM), (\varepsilon_i, i=1,NELM)$
	N11		$(\bar{\bar{P}}_{UF_i}, i=1,NELM)$
	N12		$(ECT_i, i=1,NELM)$, MPT
N13	N14	N14	$(F_{\bar{\bar{F}}_i}. i=1,NELM)$
	N15		$(\delta\bar{\bar{F}}_{K_{O_i}}, i=1,NELM)$
N16	N17		$(\varepsilon_{\sigma_i}, i=1,NELM)$, U
	N18		$\delta\bar{\bar{P}}_{(\phi)U}$
		N20	$(\bar{k}_{AVE_i}, \bar{c}_{AVE_i}, i=1,NELM)$

Note: NELM = number of physical elements (excludes point mass elements)

TABLE 8.6. NONLINEAR INCREMENTAL SOLUTION ORIGINAL DESIGN PROCESSING FILES

	Data from Last Increment (β-1)								Data for Current Increment (β)								Intermediate Data					Stiffness Averaging Pass (β)		Constant Data			Input Data
	U	EVT	$\bar{k}$ $\bar{c}$ $\delta\bar{e}_T$	ϵ_σ	$F_{\bar{F}}$	D	$\bar{F}_K$	ϵ	U	EVT	$\bar{k}$ $\bar{c}$ $\delta\bar{e}_T$	ϵ_σ	$F_{\tilde{F}}$	D	$\bar{F}_K$	ϵ	DK $\overline{CT}$ FK	δe_L $\dot{e}_L$	D_o G_f	σF $\delta\bar{F}_K$	$\delta\bar{F}_{K_o}$	$\bar{k}_{AVE}$ $\bar{c}_{AVE}$	δe_L $\delta\dot{e}_L$	$\bar{P}_{UF}$	ECT	MPT	$\delta P_{(\phi)}U$
Routine / File No.	N16 (Core)	N9	N1	N16	N13	N3	N4	N9	N17	N10	N2	N17	N14	N19	N5	N10	N6	N7	N3	N6	N15	N20	N8	N11	N12	N12	N18
PREEIG			R RW			R	R RW										W RW				W RW						R
EIGSOL																											
FIFDEF	C				R RW												R RW	W RW	W RW					R RW	R		
PØLSØL																										R RW	
PREBAL		R	R RW	R	R RW		R RW		C	W	W RW	W	W RW	W RW	W RW			R RW		W RW	R RW	W RW		R RW	R RW	C	
IMBAL 1	Test equilibrium imbalance; if satisfactory, go to IMBAL 2 ; if unsatisfactory, re-execute PREEIG, EIGSØL, FIFDEF, PØLSØL, PREBAL, AND IMBAL as follows.																										
PREEIG						R											W RW					R RW					
EIGSØL																											
FIFDEF	R RW														R RW		R RW	R RW	R RW				W RW	R RW	R RW		
PØLSØL																											
PREBAL	C				R RW		R RW		C				W RW	W- RW	W RW					W RW	R RW	R RW	R RW	R RW	R RW		
IMBAL 2								R RW	W RW			R			R RW	W RW				R RW							C

Notes: C denotes core resident
R denotes read
W denotes write
RW denotes rewind

the original design source code, calls were made to this routine to initiate printing. As noted in Table 8.4, many of these calls which were eliminated can be reactivated for checkout purposes. Routine CHKPNT is called exclusively by routine PREBAL at preset points in the code. Depending on an array of flags in routine CHKPNT corresponding to each of these points in routine PREBAL, specific arrays are output to aid in debugging the solution process. The actual printing in this instance is done by routine PLØP, a general print routine for any real or integer array.

Design and Functional Improvements

Even though the current version of the code and its approach may not be the ultimate method adopted for geometric nonlinearity, the redundancies and inefficiencies now present and planned strategies for implementation of material nonlinearity should be discussed so that they be considered in any future development effort.

Obviously, all unnecessary code relative to original theoretical approach should be eliminated. This includes all calculations relative to $\hat{a}$ and passing of certain data on external files. That data identified in Tables 8.1 and 8.2 is the only data presently required for the incremental solution. All other data processing on external files in the final code is unnecessary. Some routines included in the source code of the final version are no longer used, i.e. FØRMAT and PØLSØL.

In the iterative solution for geometric nonlinearity, routine EIGSØL recomputes the eigenvalues and eigenvectors of the A matrix at each iteration. This is unnecessary since the A matrix is constant during the iteration. Only that part of the transformed load matrix due fictitious forces and deformations changes at each iteration necessitating the solution of only those modal response equations which involve the load matrix $\bar{P}$.

Further, in the last iteration for geometric nonlinearity, routine FIFDEF need only recompute the final element forces $\bar{\bar{F}}_K$ which are a function of the input data to FIFDEF from EIGSØL. The calculations for fictitious force and deformation effects in FIFDEF are unnecessary since their only use is as input to routine EIGSØL which will not be called again for the current increment.

That portion of code in routine PREBAL relative to material nonlinearity, which was deleted in modification step 1 under *IDENT TEMP2 in Table 8.4, contained computations, logic, file processing, and calls to routines consistent with the original design for material nonlinearity. Included in this sequence of code is a call to routine PREBST. This routine was to be the executive routine for the updating of element material properties and regeneration of element stiffness and damping matrices. Under control routine PREBST, the stress/strain state of the element would be computed and tested to determine whether or not regeneration of stiffness and damping were necessary. The testing would be based on the amount of change in the following parameters since the element was last updated.

1) Temperature - recompute material properties

2) Equivalent stress in plastic range - recompute Young's Modulus, E

3) Equivalent strain - use new element geometry.

At this point in the processing sequence, the data necessary for testing and regeneration would be available from the following sources:

1) Array A(IDEBCL), which is core resident, would identify previously failed elements.

2) Matrix MPT, the core resident array of material property coefficients, would be used to compute new material properties as a function of temperature.

3) Array A(IEVT), which is core resident, would provide element material properties and other data parameters from the time of last update, e.g. temperature, equivalent stress/strain, etc.

4) Current element geometry would be available in core from the previous execution of routine PREBGM.

The following new code and modifications would be required in order to make this approach to element updating operational.

1) All necessary data is currently available with the exception of some data in array A(IEVT), e.g. increment number of last update, temperature at time of last update, change in temperature since last update, equivalent stress/strain at time of last update, etc.

2) Routine PREBST, which is a dummy routine in the original version of the source code, has to be designed and coded as do the routines required for the testing to determine if updating is necessary. Routine PREBST calls the testing routines and, if necessary, calls the regeneration routines.

3) Those routines necessary for the actual computation of new material properties and the regeneration of the compliance, stiffness, and damping matrices must be provided. Nearly all these routines are available from the Initial Generator. The routines for generation of element geometry (routines LPBG, LPMG, LPMFFB, etc.,under routine PREBGM in Figure 8.1) were also obtained from the Initial Generator and implemented in a manner simulating the program structure of the Initial Generator. This was done in anticipation of implementing the other Initial Generator routines for compliance, stiffness, and damping matrix regeneration.

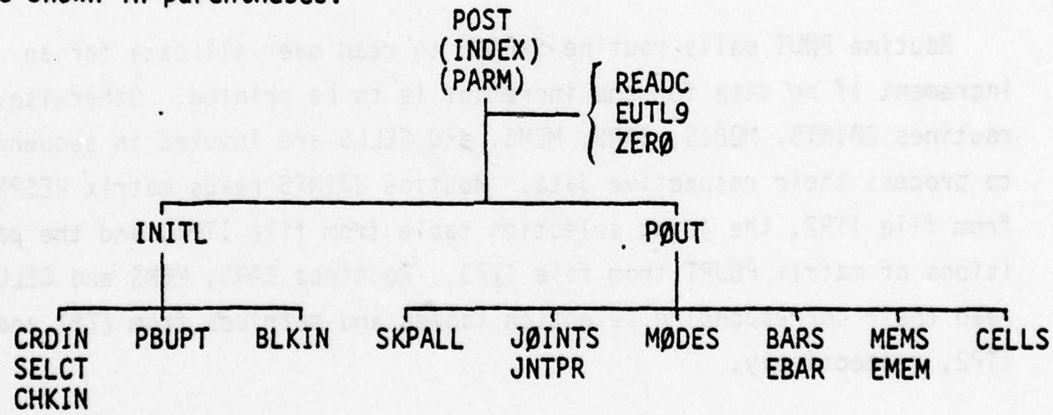
SECTION IX

POSTPROCESSOR

OVERVIEW

This program accepts output data from the linear incremental or nonlinear incremental solution and selectively prints data according to user input criteria. The card input prepared by the user is described in Part 2 of this report. The format of the output data from either linear or nonlinear incremental solution is described in Section VII of Part 3. Detailed descriptions of each routine and labeled common block in this program are given in Appendix F of this document.

The core required by this program without the use of overlay or segmentation is 60000 octal words. In addition to the system input/output files 5 and 6, three external files are required. These three files, referenced as integer variables ITP1, ITP2, and ITP3, are used for program generated selection tables, incremental solution input data, and selected partitions of input matrix PBUPT, respectively. Organization of all routines in this program is shown below. Labeled common blocks are shown in parentheses.



A singularly dimensioned array A in blank common is used for all data storage. Two labeled common blocks, PARM and INDEX, are used by the program. PARM is used to store problem parameters from input matrix CØNST as well as program generated selectivity flags. INDEX is used to store beginning locations or indices of partitions within blank common array A.

All initialization takes place under routine INITL. Initialization consists of allocating array space, processing all card input, assembling selection tables for joints, bars, membranes, and cells, assembling required partitions of input matrix PBUPT, and initializing line headers and print flags for the forces, stresses, strains, and equivalent stresses for each type of element.

Routine INITL reads matrices CØNST, TIME and UZERØ from file ITP2 and stores this data in core. Array space is then allocated followed by a call to routine CRDIN to process all card input data. Matrix DPBPHI is read over by routine INITL as is matrix PBUPT if no joint data is requested for printing. If joint data is requested, matrix PBUPT is read and processed by routine PBUPT. Routine BLKIN is then evoked to initialize all line headers and print flags for each type element.

Routine PØUT calls routine SKPALL to read over all data for an increment if no data for the increment is to be printed. Otherwise, routines JØINTS, MØDES, BARS, MEMS, and CELLS are invoked in sequence to process their respective data. Routine JØINTS reads matrix RESPNS from file ITP2, the joint selection table from file ITP1, and the partitions of matrix PBUPT from file ITP3. Routines BARS, MEMS and CELLS read their corresponding selection tables and matrices from ITP1 and ITP2, respectively.

FILE UTILIZATION

All card input is read from file 5. All printed output is written to file 6. The three scratches files required, ITP1, ITP2 and ITP3, are initialized in the main program as files 1, 2 and 3, respectively.

File ITP1 is used to store joint, bar, membrane, and cell selection tables. A selection table is assembled and written to ITP1 only if that type data exists in the model and at least one of the subsets of that type data has been requested for printing. Routine SELCT reads the card input for the selection tables which are present, assembles the table accordingly, and writes the table onto ITP1.

File ITP2 contains the incremental solution output data consisting following FØRMAT matrices:

CØNST	Matrix of problem constants
TIME	Matrix of incremental time
UZERØ	Matrix of original joint coordinates
DPBPHI	Matrix of incremental loads
PBUPT	Modal to T degree of freedom transformation matrix

and for each increment:

RESPNS	Modal response matrix
BARS	Bar element response matrix
MEMBRN	Membrane element response matrix
CELLS	Cell element response matrix

If bar, membrane, or cell element data does not exist in the modal, the corresponding matrix will not be present in the incrmental solution output.

File ITP3 is used to store partitions of matrix PBUPT. Routine PBUPT assembles the partitions according to the joints selected for printing coordinate, displacement, velocity, and/or acceleration data.

CORE UTILIZATION

All array space is allocated into a singularly dimensioned blank common array A. The beginning location of each partition within array A is stored in labeled common block INDEX. Labeled common block

PARM is used to store problem parameters and print selection flags. Both labeled common blocks, PARM and INDEX, are used by all principle routines.

Some of the more pertinent problem parameters stored in labeled common block PARM are described below.

NJTS	number of joints
NBARS	number of bars
NMEMS	number of membranes
NCELS	number of cells
NMØDS	number of modes
NTVLS	number of time intervals

LIMITATIONS

Sufficient space must be available in blank common array A to accommodate the problem. The size of array A required for a given problem is a function of the problem parameters stored in labeled common block PARM. If IWØRK is the length of array A, and defining L_1 as

$$6*\text{NMØDS} \leq L_1 \geq 60$$

and L_2 as the larger of NJTS, NBARS, NMEMS, and NCELS, then

$$\text{IWØRK} \geq 640 + L_1 + L_2 + 2*\text{NTVLS} + 3*\text{NJTS} + 3*\text{NMØDS}$$

The integer IWØRK is the first word of blank common and is equal to the size of array A in blank common. The integers ILNMAX and ICLMAX are stored in labeled common block INDEX and are equal to the maximum number of lines and columns printed on each page. These three variables, IWØRK, ILNMAX and ICLMAX, are initialized to 10000, 60, and 8, respectively. However, the user may override these values on the first input card provided the following is true of the user specified values:

$$IWØRK \geq 10000$$

$$ILNMAX \geq 20$$

$$8 \geq ICLMAX \geq 1$$

SECTION X
PROJECTED IMPROVEMENTS

EFFICIENCY

The following modifications to the final version of the program code are recommended in order to reduce the machine resources required for execution and to reduce execution time.

1) Increase the dimensioned size of the arrays in labeled common blocks JØRT and WRK in the Initial Generator so that they take advantage of the available space defined by labeled common blocks EDØF and ISEQ, respectively (see Section V, LIMITATIØNS).

2) Where possible, take advantage of symmetry in the generation of $\bar{k}$, $\bar{s}$, m and ϵ_σ in the Initial Generator.

3) Use SEGLØAD in the implementation of the Linear and Nonlinear Incremental Solution programs.

4) Delete excess files declared in the PROGRAM statements of the Linear and Nonlinear Incremental Solution programs.

5) Delete unnecessary copies of master input data to scratch files in routine MATIN of the Linear and Nonlinear Incremental Solution programs.

6) In the Linear Incremental Solution, eliminate the allocation of space for the joint coordinate data since it is not required.

7) Eliminate all unnecessary processing in the Nonlinear Incremental Solution program left over from the original design (see Section VIII, FUTURE DEVELOPMENT).

8) Eliminate all unnecessary printing from the Nonlinear Incremental

Solution program left over from program check out.

9) Provide the logic and processing commands to store joint coordinate data on an external file during the eigen problem solution (EIGSØL) in the Nonlinear Incremental Solution.

10) Perform the functions of routine PREEIG (regeneration of $\bar{K}$, $\bar{C}$) in routine PREBAL of the Nonlinear Incremental Solution.

EXTENDED CAPABILITY

The following modifications to the final version of the program code are recommended in order to extend basic capabilities already in existance.

1) Development convergence criteria and necessary code to end the iteration process through routines EIGSØL and FIFDEF for geometric nonlinearity in the Nonlinear Incremental Solution program.

2) Introduce the cosine load variation or impedence matrix to the Linear Incremental Solution program.

3) Modify the processing matrix $\delta\bar{e}_T$ in routine PREEIG of the Nonlinear Incremental Solution to account for the fact that $\delta\bar{e}_T$ is presently assumed constant.

Introduction of additional capabilities to the Nonlinear Incremental Solution such as material nonlinearity and restart will be contingent on theoretical development. Guidelines for program modifications to accommodate material nonlinearity are discussed in Section VIII under FUTURE DEVELOPMENT. In this particular area, consideration should be given to the use of random access files for external storage of matrices $\bar{k}$, $\bar{c}$, $\delta\bar{e}_T$, and EVT in element partitions. This would provide for rewriting these matrices only for those elements whose properties change rather than for all elements as the present design requires.

APPENDIX A

LAMINATE GENERATOR ROUTINES

APPENDIX A
LAMINATE GENERATOR ROUTINES

This appendix contains detailed descriptions of all routines in this program. Table A gives page number references within this document for documentation of each routine.

The detailed description of each routine is divided into the following subheadings:

Algorithm — verbal flow chart of routine logic and data flow

Input/Output — description of all external data set input/output

Argument List — name, type, and description of each argument

Labeled Common — list of all labeled common blocks declared

Subroutines Called — list of all routines called

Error Detection — description of tests made for errors and action taken

TABLE A. INDEX TO LAMINATE GENERATOR ROUTINES

MAIN PROGRAM LAMGEN

The main program allocates core and calls the input, calculation, and output subroutines.

Algorithm

Two arrays are set up -- A and N -- and every element in the A array is initialized to 10^{30}. The first subroutine called is IØXS to read and write Data Code 2 data (if any). The next subroutine called is RLAYER. The element and joint numbering increments read by this subroutine are required by MAIN to allocate sufficient core for generated cell and joint data. The message
'LAYERS READ NO. LAYERS = nnnn'
is printed after a successful return from RLAYER. The call to RIDCEL is next. The number of quadrilaterals and the joint numbers of their corners read in by this subroutine are required to allocate sufficient core to contain the generated joint data. After calculating the total number of cells that will have to be handled the message
'CELLS HAVE BEEN READ NO. CELLS = nnnnn NO. QUADS = nnnnn'
is printed. The subroutine that reads the joint data, RJØINT, is called next, and after the largest joint number to be handled is calculated, the message
'JOINTS HAVE BEEN READ NO. JOINTS = nnnnn'
is printed. The subroutine CELGEN, called next, sets up an array of quadrilateral corner joint numbers required by RTVARI and JTGEN, which are called next. The subroutines WJØINT, WNØRM, and WCELL follow, with calls to IØXS interspersed to keep the numerical order of the Data Codes on the output files intact.

Input/Output

There is no input, all output is printed on file 6.

Argument List

None

Labeled Common

None

Subroutines Called

IØXS, RLAYER, RIDCELL, RJØINT, CELGEN, RTVARI, JTGEN, WJØINT, WNØRM, WCELL

Error Detection

If an error flag is received from one of the reading subroutines (RLAYER, RIDCEL, or RJØINT) indicating necessary data was not read, the message
'JOB TERMINATED'
is written and execution of the program is stopped.

FUNCTION ABSVAL

This routine finds and returns the length of a vector.

Algorithm

The three components of the input vector are squared, the square root of the sum of the squares is the length of the vector.

Input/Output

None

Argument List

C	A real array of components of a vector

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CELGEN

This routine generates the data required to form the cells of the laminate and sets up an array of joint numbers that correspond to the corners of quadrilaterals.

Algorithm

Data for the cells is generated for a stack of cells under each quadrilateral, one stack at a time. First, the stress orientation angle and material number for each cell in the stack are stored in the cell array. Then, for each corner of the quadrilateral, the number of the joint at the corner is stored in an array in numerical order (i.e., no duplication of joint numbers), a cross product is formed of the two sides of the quadrilateral forming the corner, a quarter of the magnitude of which is added to the running sum in the area array for that joint, and the unit vector of which is added to a running sum in the surface normal array, and finally, the numbers of the joints in the stack and stored in the cell array in the correct location for each cell in the stack. The joint coordinates in the stack are checked to see if any of the joints has been already defined by the user. If a user-defined joint does exist below the quadrilateral corner, the joint number of the corner is made negative as an indicator for later subroutines (JTGEN & RTVARI).

Input/Output

None

Argument List

VJØINT A real array of coordinates of joints stored by joint number

IDCELL An integer array of numbers of the joints on the eight corners and the material number of each cell, stored by cell number

ZETA A real array of stress orientation angles

ML An integer array cell number increments, joint number increments, and material number of each layer

JØINTP An integer array of joint numbers corresponding to the corners of quadrilaterals, stored in numerical order

JQUADP An integer array of numbers of cells that have only four defined corners (quadrilaterals) stored in order in which they were read

VNØRM A real array of running sums of unit vectors at each quadrilateral corner, stored in the same order as the joints in JØINTP

AREA A real array of areas associated with the joints on the quadrilateral corners

NJØINT An integer scalar defining the total number of joints

NJT An integer scalar defining the number of joints that define quadrilateral corners

NCELLS An integer scalar defining the total number of cells

NQUAD An integer scalar defining the number of quadrilaterals

NL An integer scalar defining the number of layers

Labeled Common

None

Subroutines Called

CRØSSQ, UVEC, ABSVAL

Error Detection

None

SUBROUTINE CRØSSQ

This routine forms and returns the cross product of two input vectors.

Algorithm

The mathematics of forming a cross product are performed and the resultant returned through the argument list.

Input/Output

None

Argument List

U	A real array for the first input vector
V	A real array for the second input vector
W	A real array for the resultant vector

Labeled Common

None

Subroutines Called

None

Error Detection

None

FUNCTION DØT

This routine calculates the dot product of two input vectors.

Algorithm

The pairs of x, y, and z components of the two input vectors are multiplied and added.

Input/Output

None

Argument List

U	A real array for the first vector
V	A real array for the second vector

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE IØXS

This routine reads selected data and directly writes it back out on files 6 and JTAPE.

Algorithm

Given beginning and ending Data Code numbers for a series of Data Codes, all data with Data Codes in the series is read from ITAPE and written in the same order on file 6 and, if JTAPE ≠ 6, on file JTAPE.

Input/Output

Input is read from file ITAPE; output is written on files 6 and JTAPE.

Argument List

ITAPE	An integer scalar defining the number of the input file
JTAPE	An integer scalar defining the number of one of the output files
JCØDE	An integer scalar defining the starting Data Code number
LCØDE	An integer scalar defining the ending Data Code number

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE JTGEN

This routine generates the joint coordinates for the joints in the laminate.

Algorithm

The first step is to unitize the normal vectors created by CELGEN. The coordinates of the joints within the laminate are generated either along these normal vectors or along lines connecting user-defined joints in the laminate. The coordinates are generated for the stack of joints under each quadrilateral corner. If the coordinates of the first joint in the stack are other than the input values, the new coordinates are generated by multiplying the local normal vector by the input offset and adding that vector to the input location of the first joint. If the joint number stored in the array of quadrilateral corner joints is negative, indicating there is a user-defined joint in the stack of joints, there is a branch to the segment of the subroutine that constructs unit vectors between user defined joints in the stack and calculates the intervening joint coordinates along these unit vectors. If the joint number at the top of the stack is positive all the coordinates of joints in the stack are generated along the local surface normal.

Input/Output

None

Argument List

VJØINT — A real array of coordinates of each joint

ML — An integer array of layer information: element number increments, joint number increments, and material number

TL — A real array of layer thickness, including the offset of the true outer laminate surface from the input laminate surface joints

TVARI A real array of layer thickness for each stack of joints that does not have internal user-defined joints

JØINTP An integer array of joint numbers corresponding to quadrilateral corners stored in numerical order

VNØRM A real array of surface normal vectors corresponding to quadrilateral corners, stores in the same order as the joints in JØINTP

NJT An integer scalar defining the number of layer

Labeled Common

None

Subroutines Called

UVEC, DØT

Error Detection

None

SUBROUTINE RIDCEL

This routine reads the input file searching for Data Code 40 data (cell data) which it stores.

Algorithm

The input file is read until a Data Code 40 card is encountered. The file is then backspaced one record and the first cell data is read and stored in an array in a location corresponding to the cell number read. If the input number used to define the fifth corner is zero, the cell is called a quadrilateral and its number is stored in another array. The file is read and the data stored until a cell number of 9999 is read or the Data Code on the card is not 40.

Input/Output

The input is read from file ITAPE.

Argument List

IDCELL	An integer array of numbers of joints on the eight cell corners (four corners for quadrilaterals) and the material number stored according to cell number
ZETA	A real array of stress orientation angles stored according to cell number
IDQUAD	An integer array of numbers of the cells that only have four defining corner joints input (quadrilaterals) stored in the order in which they are read
IFLAG	An integer scalar flag signalling the main program to terminate execution if no cell data are found
NCELLS	An integer scalar defining the largest cell number read
JQUAD	An integer scalar defining the number of quadrilaterals encountered
MAXQ	An integer scalar defining the largest cell number among the quadrilaterals

MAXJT An integer scalar defining the largest joint number used to define a quadrilateral corner

ITAPE An integer scalar defining the number of input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no cell data cards are found the message 'NO CELLS (DATA CODE 40) ENCOUNTERED IN INPUT' is printed and IFLAG is set equal to 1 to cause the main program to terminate execution.

SUBROUTINE RJØINT

This routine reads the input file searching for Data Code 2 data (joint data) which it stores.

Algorithm

The input file is read until a Data Code 2 card is read. The file is backspaced one record and the coordinates are stored in an array in a location corresponding to the joint number. The file is read and the contents stored until a joint number of 9999 is read or a card without a Data Code of 2 is encountered.

Input/Output

The input is read from file ITAPE.

Argument List

VJØINT	A real array of coordinates of input joints stored by joint number
IFLAG	An integer scalar flag signalling the main program to terminate execution of no joint
NJØINT	An integer scalar defining the largest joint number read
ITAPE	An integer scalar defining the number of input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no joint data cards are found, the message 'NO JOINTS (DATA CODE 2) ENCOUNTERED IN INPUT' is written and IFLAG is set equal to 1 to cause the main program to terminate.

SUBROUTINE RLAYER

This routine reads the input file searching for Data Code 7 (layer data) which it stores.

Algorithm

The input file is read until a Data Code 7 card is found. The file is then backspaced one record and the layer data is stored in an array in a location corresponding to the layer number plus one. The zeroth layer (which really contains only the distance from the loft line to the top layer) is thus stored in the first column of the layer data. The input file continues to be read and layer data stored until a layer identification of 9999 is read or until a card that does not carry a Data Code of 7 is read.

Input/Output

The input is read from file ITAPE.

Argument List

TL — A real array of thicknesses of the layers

ML — An integer array of material number, element number increment, and joint number increment for each layer

IFLAG — An integer scalar for signalling the main program to determine if no layer data is read

NL — An integer scalar defining the largest layer number read plus one

ITAPE — An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no layer data cards are read the message 'NO LAYERS (DATA CODE 7) ENCOUNTERED IN INPUT' is printed and IFLAG is set equal to 1 to cause the main program to terminate.

SUBROUTINE RTVARI

This routine determines the thickness of every layer at every joint in the laminate using the thicknesses read in RLAYER for each layer as the default values and reading revised thickness data to calculate and store the non-default thicknesses.

Algorithm

First all the thicknesses read in by RLAYER are stored in an array in locations corresponding to layer number and quadrilateral corner number so that each quadrilateral corner has a stack of thicknesses associated with it. Then the input file is read, searching for revised thickness data (Data Code 8). When the data is found it is stored in the appropriate location in the array created earlier, replacing default values. The reading and storing continues until a joint number of 9999 is encountered or the card does not carry a Data Code of 8.

Input/Output

The input is read from file ITAPE.

Argument List

TVARI A real array of thicknesses of layers beneath each joint on a quadrilateral corner

TL A real array of thicknesses of layers stored by layer number by RLAYER

JØINTP An integer array of joint numbers corresponding to quadrilateral corners

NL An integer scalar defining the number of quadrilateral corners

ITAPE An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If any joint number input through Data Code 8 data has a user-defined joint in the stack of joints below it the message
'JOINT nnnn LISTED IN DATA CODE 8 IS ABOVE A USER-DEFINED JOINT -- USER-DEFINED JOINT WILL REMAIN UNCHANGED'
is printed but execution continues since the user-defined joint will take precedence later on in the program.

If the revised thickness data duplicates the existing thickness data, the message
'CARD ENCOUNTERED IN DATA CODE 8 DUPLICATES EXISTING LAMINATE DATA'
is printed as a caution to the user and execution continues.

If any input joint is not found in the list of joint numbers corresponding to quadrilateral corners the message
'JOINT NO. nnnn LISTED IN DATA CODE 8 IS NOT FOUND IN LIST OF QUAD CORNER JØINTS -- BAD DATA MAY RESULT'
is printed and execution is continued although revised thicknesses are liable to be applied to the wrong quadrilateral corners.

If no revised thickness data is found the message
'NO REVISED THICKNESSES (DATA CODE 8) ENCOUNTERED IN INPUT'
is printed and execution proceeds, assuming all thicknesses to be as they were read by RLAYER.

SUBROUTINE UVEC

This routine forms and returns a unit vector from an input vector.

Algorithm

The length of the input vector is found and the input vector's components are divided by the length. The resulting unit vector is returned through the argument list.

Input/Output

None

Argument List

I An integer scalar defining the location of input vector in array C

J An integer scalar defining the location of output vector in array C

C A real array of vectors

Labeled Common

None

Subroutines Called

None

Error Detection

If the length of the vector to be unitized is zero, the message
'ATTEMPT TO UNITIZE ZERO VECTOR'
is printed and the zero vector is divided by a length of one and returned.

SUBROUTINE WCELL

This routine outputs the cell data in the format for Data Code 40 on files 6 and JTAPE.

Algorithm

The cell data (cell number, corner joint numbers, material number, and stress orientation angle) are written on file 6 and, if JTAPE ≠ 6, on file JTAPE as well. The format for the JTAPE output does not include carriage control characters. After all the cell data has been written a cell number of 9999 is output to mark the end of the cell data.

Input/Output

The output is written on files 6 and JTAPE.

Argument List

IDCELL	An integer array of cell data: eight joint numbers defining the corners and the material number for each cell
ZETA	A real array of stress orientation angles
NCELLS	An integer scalar defining the number of cells
JTAPE	An integer scalar defining the number of one of the output files

Labeled Common

None

Subroutines Called

None

Error Detection

If arrav IDCELL has data greater than 10^5, indicatino no data was read or qenerated for the cell, the message 'GAP IN DATA' is outout and execution continues with the next cell for which the program has valid data.

SUBROUTINE WJØINT

This routine outputs the joint data in the Data Code 2 format on files 6 and JTAPE.

Algorithm

The joints data (joint number and coordinates) is written on file 6 and, if JTAPE ≠ 6, on file JTAPE as well. The format for the JTAPE output has no carriage control characters. After all the joint data has been output, a joint number of 9999 is output as a flag indicating the end of the data.

Input/Output

The output is written on files 6 and JTAPE.

Argument List

VJØINT	A real array of coordinates of the joints
NJØINT	An integer scalar defining the number of joints
JTAPE	An integer scalar defining the number of one of the output files

Labeled Common

None

Subroutines Called

None

Error Detection

If the x-coordinate of a joint is 10^{30}, indicating that no data for the joint was read or generated, the message

'GAP IN SEQUENCE'

is output and execution continues at the next joint whose coordinates are known.

SUBROUTINE WNØRM

This routine outputs the surface normal data in the Data Code 6 format on files 6 and JTAPE.

Algorithm

The normals and areas which have been generated for every joint on a quadrilateral corner are output on file 6 and JTAPE ≠ 6, on file JTAPE as well. The format for JTAPE has no carriage control characters. When all normals have been written a joint number of 9999 is output to signal the end of the surface normal and area data.

Input/Output

The output is written on files 6 and JTAPE.

Argument List

VNØRM A real array of the unitized surface normals for every joint on a corner of an input quadrilateral

AREA A real array of the areas associated with every joint on a corner of an input quadrilateral

JØINTP An integer array of the joints on the corners of the input quadrilaterals

NJT An integer array of numbers of joints on corners of input quadrilaterals

JTAPE An integer scalar defining the number of one of the output files

Labeled Common

None

Subroutines Called

None

Error Detection

None

APPENDIX B

INITIAL GENERATOR ROUTINES AND LABELED COMMON BLOCKS

APPENDIX B
INITIAL GENERATOR ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table B gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the doucmentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/ output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usage	list of all routines which contain declarations for the labeled common block

TABLE B. INDEX TO INITIAL GENERATOR ROUTINES AND LABELED COMMON BLOCKS

TABLE B. INDEX TO INITIAL GENERATOR ROUTINES AND LABELED COMMON BLOCKS (Continued)

TABLE B. INDEX TO INITIAL GENERATOR ROUTINES AND LABELED COMMON BLOCKS (Continued)

TABLE B. INDEX TO INITIAL GENERATOR ROUTINES AND LABELED COMMON BLOCKS (Continued)

MAIN PROGRAM BIRDG1

This routine is the overall executive or supervisory routine for the windshield technology program Initial Generator. It controls the sequence of input, the calling of subprograms or modules, and the coordination of data output. For the derivation of element matrices and additional mathematical formulations, refer to Appendix H of Part 1 of this report.

Algorithm

The program begins by initializing various counters and flags. Then Data Code 1 is read and checked for validity. Subroutines JZERO and JTEMP are invoked to read Data Codes 2 and 3 building the joint coordinate and joint temperature tables. These are printed out by subroutine JTPRT.

Next, a tape header is written on tape 1 using subroutine TAPEHD. The joint coordinate table is written onto tape 1, with the header and trailer records written by routines MATHD and MATTR. Direction numbers and constraints, Data Codes 4 and 5, are read and the reaction table created and written onto tape 16 by execution of subroutine CNCSTN. Next Data Code 6 is skipped over by calling routine SKDATA. Material properties, Data Code 10, are processed for future use and written onto tape 1 by module MTLMOD. Again SKDATA is used to skip over unnecessary input Data Codes 11 and 12, if present.

The program now goes into a loop for processing the various elements. One element is processed each time through the loop. First bar elements, then membranes, cells and point mass elements are processed in order. Subroutine ELEMENT is used to input and validate the element definition input, Data Codes 20, 30, 40, and 50. The joint temperature differentials are calculated and the material properties are then obtained by invoking routine ACCPRP. Print flags are checked and the compliance matrix is formed according to the type of element. Subroutine LPBAR, LPCAP and LPMAP are invoked to generate the various element matrices which are saved as partitions of their respective matrices representing the total structure by routines LPBSM, LPCSM, LPMSM, and, in the case of point mass elements, by main program.

After processing all elements, the edges are processed by routine EDGDØF. Using the edge data, the ETC table and some program constants, the assembly module PASSM generates the PRUPT and the PRUF matrices writing them out to tape 1. Next, routine WTAPE1 is used to generate the K, M, KBAR, CBAR, FFBAR, SIGMA FBAR, EPSILØN SIGMA, EBØT, and EVT matrices. These matrices along with the CØNST table are written onto tape 1. Finally, a tape trailer record is written onto tape 1 using routine TAPTR called from BIRDG1.

If a dump of tape 1 is requested by the user, it is dumped using the routine DUMPMT. When this is completed, the program constants are printed by subroutine PRTCØN and the run is wrapped up.

Input/Output

Tape 1 is the output tape for binary matrix data and is formatted for use as a FØRMAT matrix master input tape. Tapes 5 and 6 are used for system input and output, respectively. Tapes 7 through 18 are used as scratch units to store data passed between modules within the program. The data temporarily stored on these tapes is as follows:

Tape	Data
7	Element variable table (ECT)
8	Element damping matrix ($\bar{c}$)
9	Element thermal matrices ($\bar{e}_{T\Delta}$ and $\delta\bar{e}_T$)
10	Element mass matrix (m)

Tape	Data
11	Element stress/strain transform (ϵ_σ)
12	Element force transform ($F_{\overline{F}}$)
13	Element stiffness matrix ($\overline{k}$)
14	Element stiffness components (KLPB, DLPM and KLPC)
15	Element force/stress transform ($\sigma_{\overline{F}}$)
16	Global/oblique constraint tables
17	Edge degree of freedom table
18	Element constant table (ECT)

Argument List

None

Labeled Common

CØNST, IERRØR, JØRT, LPEP, MTRL

Subroutines Called

ACCPRP, CNSTRN, DUMPMT, EDGDØF, EDGES, ELEMNT, JTEMP, JTPRT, JZERØ, LPBAR, LPBSM, LPCAP, LPCC, LPCSM, LPMAP, LPMC, LPMSM, MATHD, MATTR, MTLMØD, PASSM, PRTCØN, SKDATA, TAPEHD, TAPETR, WTAPE1

Error Detection

A check is made for the Data Code 1 delimiter card being missing or out of sequence. If this card is missing the run is aborted. If the base temperature is not input, the program sets the IERRØR flag equal to zero aborting the run but continues execution to allow a scan of the remaining input data for additional errors.

LABELED COMMON ABC

This common block is used by the sorting module to store sort control parameters.

Declaration

```
CØMMØN /ABC  / INIHI,INILØ,IP,ISWAP,ICNTR,KEYFLD
```

Contents

INIHI	Pointer to last record in a file to be partitioned
INILØ	Pointer to first record in a file to be partitioned
IP	Pointer to record that is the partitioning agent
ISWAP	Record interchange counter
ICNTR	Pointer to next subfile in the subfile stack
KEYFLD	Record element that contains the sort key

Usage

PARTN, QKSØRT, STACK, SWAP

LABELED COMMON CØFCLC

This common block is used to store data and program parameters associated with the computation of material properties.

Declaration

```
CØMMØN /CØFCLC/ TVPA(10,10),RTMP(10,3),ITMP(10,3),VPT(10),
                AP(10),ASC(10,10),BSC(10)
```

Contents

TVPA	An array of temperatures for a property where the rows are temperatures associated with a property in VPT and the columns are monotonically increasing powers of the temperature
RTMP	Scratch space required by routine SID, must be at least 3*number of rows
IMPT	Same as RTMP
VPT	Input property values associated with temperatures in TVPA
AP	Temporary storage for the calculated coefficients
ASC	Scratch storage used in the calculations
BSC	Same as ASC

Usage

CØLALC, PRØP

LABELED COMMON CØNST

This common block is used to store coefficients and counters describing the overall model size and problem constants.

Declaration

```
DIMENSION CØNST(30)
CØMMON /CØNST/ NJ, NC, NM, NBAR, NMEM, NCELL, NPTM, NEDØF, NMØDES,
               NPLJ, NØR, TBASE, HJ, NMC, ND, MP1, MP1NM, GC, GE
EQUIVALENCE (CØNST(1),NJ)
```

Contents

NJ	Number of joints
NC	Number of constraints
NM	Number of materials
NBAR	Number of bar elements
NMEM	Number of membrane elements
NCELL	Number of cell elements
NPTM	Number of point mass elements
NEDØF	Number of edge degrees of freedom
NMØDES	Number of modes
NPLJ	Number of pressure loaded joints
NØR	Number of oblique reactions
TBASE	Base temperature of the structure
HJ	Mechanical equivalent of heat
NMC	Total number of material property coefficients
ND	Total number of material property words
MP1	Maximum number of properties per material + 1
MP1NM	MP1*NM

GC Cell stiffness matrix suppression coefficient

GE Bar compliance matrix suppression coefficient

<u>Usage</u>

BIRDG1, JTEMP, JZERØ, MTLMØD, PASSM, PASSM1, PASSM2, PASSM4, PASSM5, PRTCØN, WTAPE1

LABELED COMMON EDØF, IEDGE, AND ISEQ

These common blocks are always used as a group by those modules associated with the computation of overall structural degree of freedom row formats.

Declaration

```
CØMMØN /EDØF/ IDØF(7200) /IEDGE/ IEDG(7200) /ISEQ/ ISEQ(7200)
```

Contents

IDØF	Edge degrees of freedom in same sequence as ISEQ
IEDG	Sorted pairs of joints defining edges
ISEQ	Index to original sequence of IEDG

The content of these common regions varies between modules. Source code listings should be consulted for specific information regarding content.

Usage

EDGDØF, PASSM

LABELED COMMON IDEN

This common block is used to store data and program parameters associated with reading material property input data, Data Code 10.

Declaration

```
CØMMØN /IDEN/ MR,IP,NPC,MAP,MRØLD,IPØLD,NCØLD,MAPØLD
```

Contents

MR	Material reference number of current record divided by 100
IP	Property identification number of current record
NPC	Number of coefficients input for current property
MAP	Number of coefficients to be computed for current property
MRØLD	MR for the previous record
IPØLD	IP for the previous record
NCØLD	NPC for the previous record
MAPØLD	MAP for the previous record

Usage

CØCALC, MATDES, MTLMØD, PRØP

LABELED COMMON IERRØR

This common block is used to store the main error flag and print control flags.

Declaration

```
CØMMØN /IERRØR/ IERRØR,LNPAGE,MTDUMP,NPASSM
```

Contents

IERRØR Program fatal error flag

LNPAGE Lines per page print limit, defaults to 45 lines

MTDUMP Tape 1 dump flag which if 0 means no dump, if 1 dump header records only, and if 2 dump complete tape

NPASSM Print flag for PRUPT and PRUF matrix assembly

Usage

BIRDG1, ACCPRP, CNSTRN, CØCALC, DRCNUM, EDGDØF, EDGES, ELEMNT, JTEMP, JZERØ, LPCKC, MATDES, MTLMØD, ØVER3, PASSM, PASSM1, PASSM2, PASSM4, PASSM5, PRØP, PRTCØN, RAMØSG, WTAPE1

LABELED COMMON JORT

This common block is used to store joint coordinate and temperature data.

Declaration

```
CØMMØN /JORT/ JO(3,1200), TEMP(1200)
REAL JO
```

Contents

JO	Joint coordinate table
TEMP	Joint temperature table

Usage

BIRDG1, ACCJZE, JTEMP, JTPRT, JZERØ, LPBAR, LPBPRT, LPCAP, LPCG, LPCPRT, LPMAP, LPMG, LPMPRT

LABELED COMMON LIMITS

This common block is used to store size limitation parameters associated with material property tabular data, Data Code 10.

Declaration

```
CØMMØN /LIMITS/ MXMTRL,MXPRØP,MXCØFS,MXDES,MXPCØ,MXPVL
```

Contents

MXMTRL Maximum number of materials that space is provided for

MXPROP Maximum number of properties that space is provided for

MXCOFS Maximum number of coefficients for all properties for all materials that space is provided for

MXDES Maximum number of packed words (10 characters per word) of material description for all materials

MXPCO Maximum number of coefficients for a property

MXPVL Maximum number of values for a property

Usage

CØCALC, MATDES, MTLMØD, PRØP

LABELED COMMON LPBEM

This common block is used to store geometry and output matrix data for bar elements.

Declaration

```
CØMMØN /LPBEM/ CBAR(2,2), EBARTD(2),  EBOT(2)  ,
                EMASS(6,6), EPSSIG    , FFBAR(7,2),
                 KLPB(7,7), KBAR(2,2), SFBAR(2)  ,
                   AB(4)  ,     PQ(4)
REAL KBAR, KLBP
```

Contents

CBAR	Element damping matrix
EBARTD	Unassembled element deformations due to unit temperature change
EBOT	Initial element thermal deformations
EMASS	Element mass matrix
EPSSIG	Element strain matrix
FFBAR	Unassembled to global transformation matrix
KLPB	Transformed element stiffness matrix
KBAR	Element stiffness matrix
SFBAR	Element stress matrix
AB	X, Y, Z components of unit vector from P to Q
PQ	Bar length and X, Y, Z components of length

Usage

LPBAR, LPBPRT, LPBSM

LABELED COMMON LPCEM

This common block is used to store cell element output matrices.

Declaration

```
CØMMØN /LPCEM/  CBAR(30,30), EBARTD(30)  ,  EBOT(30),
               EMASS(24,24), EPSSIG(6,12), FFBAR(36,30),
                KBAR(30,30), KLPC(36,36) , SFBAR(12,30)
REAL KBAR, KLPC
```

Contents

CBAR	Element damping matrix
EBARTD	Unassembled element deformations due to unit temperature change
EBOT	Deformations due to the difference between element joint temperature and the structural base temperature
EMASS	Element mass matrix
EPSSIG	Element strain matrix
FFBAR	Unassembled to global transformation matrix
KBAR	Element stiffness matrix
KLPC	Transformed element stiffness matrix
SFBAR	Element stress matrix

Usage

LPCAP, LPCEBT, LPCK, LPCKC, LPCRM, LPCSM, LPCZM

LABELED COMMON LPCV

This common block is used to store cell element geometry and vector data.

Declaration

```
CØMMØN /LPCV/ PQ(4.8)   , RQ(4,8)   , RS(4,8)   , PS(4,8)   ,
              AB(4,8)   , BB(4,8)   , CB(4,8)   , DB(4,8)   ,
              EB(4,8)   , FB(4,8)   , FK(5,8)   ,
              LA(8)     , LB(8)     , LT(8)     , THETA(8) ,
              TPB(4)    , TQB(4)    , TRB(4)    , TSB(4)    ,
              TPQ(4)    , TRQ(4)    , TRS(4)    , TPS(4)    ,
              V(8)      , ZETA(8)
REAL LA, LB, LT
```

Contents

PQ	Edge vector components and magnitude
RQ	Edge vecgor components and magnitude
RS	Edge vector components and magnitude
PS	Edge vector components and magnitude
AB	Unit vector components
BB	Unit vector components
CB	Unit vector components
DB	Unit vector components
EB	Unit vector components
FB	Unit vector components
FK	K factor
LA	Length of cell sub-element
LB	Width of cell sub-element
LT	Average thickness of cell sub-element
THETA	Corner angles THETAPO, THETAQO, THETARO, THETASO, THETAP1, THETAQ1, THETAR1, and THETAS1

TPB	Thickness vector components and magnitude
TQB	Thickness vector components and magnitude
TRB	Thickness vector components and magnitude
TSB	Thickness vector components and magnitude
TPQ	Thickness vector components and magnitude
TRQ	Thickness vector components and magnitude
TRS	Thickness vector components and magnitude
TPS	Thickness vector components and magnitude
V	T*A*B*SIN(THETA) for cell sub-element
ZETA	Orientation angle of cell sub-element

<u>Usage</u>

LPCAP, LPCEBT, LPCFFB, LPCG, LPCKC, LPCPRT, LPCSM

LABELED COMMON LPEP

This common block is used to store material properties, joint connectivity data, and other information for an element.

Declaration

```
DIMENSION CK(3,3), JMN(4)
COMMON /LPEP/  A         , ALPHAT   , CK1(6,6), CK2(6,6), E     ,
               EC        , ET       , H       , JCN(8)  , MR    ,
               MT        , NU       , PM(3)   , PRFLAG  , RHØ   ,
               T         , TD(8)    , VØL     , ZETAPQ
EQUIVALENCE (CK1(1,1),CK(1,1)),(JCN(1),JMN(1)),
            (JP ,JCN(1)),(JQ ,JCN(2)),(JR ,JCN(3)),(JS ,JCN(4)),
            (JPO,JCN(1)),(JQO,JCN(2)),(JRO,JCN(3)),(JSO,JCN(4)),
            (JP1,JCN(5)),(JQ1,JCN(6)),(JR1,JCN(7)),(JS1,JCN(8)),
REAL NU, MT
INTEGER PRFLAG
```

Contents

A	Cross sectional area
ALPHAT	Coefficient of thermal expansion
CK	Compliance matrix
CK1	Compliance matrix for upper half of LPC element
CK2	Compliance matrix for lower half of LPC element
E	Elastic modulas
EC	Compression gap
ET	Tension gap
H	Damping/stiffness ratio
JCN	Array containing cell joint numbers JPO, JQO, JRO, JSO, JP1, JQ1, JR1, and JS1
JMN	Array containing membrane joint numbers JP, JQ, JR, and JS
MR	Material properties reference code

MT	Total element mass
NU	Poisson's ratio
PM	Point mass
PRFLAG	Print flag
RHØ	Mass density
T	Membrane thickness
TD	Array containing unit temperature differential at joints JPO, JQO, JRO, JSO, JP1, JQ1, JR1, and JS1
VØL	Element volume
ZETAPQ	Material or stress orientation angle

Usage

BIRDG1, ELEMNT, LPBAR, LPBPRT, LPBSM, LPCAP, LPCC, LPCEBT, LPCFFB, LPCG, LPCKC, LPCPRT, LPMPRT, LPCSM, LPMSM, LPMAP, LPMC, LPMFFB, LPMEBT, LPMG

LABELED COMMON LPMDSZ

This common block is used to store membrane element geometry data and intermediate matrix data.

Declaration

```
COMMON /LPMDSZ/ AØTP   , AØTQ   , AØTR   , AØTS   ,
                DP(3,3), DQ(3,3), DR(3,3), DS(3,3),
                SP(3)  , SQ(3)  , SR(3)  , SS(3)  ,
                ZP(3,3), ZQ(3,3), ZR(3,3), ZS(3,3)
```

Contents

AØTP	A*B*sin (THETA)/t for corner p
AØTQ	A*B*sin (THETA)/t for corner q
AØTR	A*B*sin (THETA)/t for corner r
AØTS	A*B*sin (THETA)/t for corner s
DP	$\tilde{D}$ matrix for corner p
DQ	$\tilde{D}$ matrix for corner q
DR	$\tilde{D}$ matrix for corner r
DS	$\tilde{D}$ matrix for corner s
SP	Skew stress matrix for corner p
SQ	Skew stress matrix for corner q
SR	Skew stress matrix for corner r
SS	Skew stress matrix for corner s
ZP	Global translation matrix for corner p
ZQ	Global translation matrix for corner q
ZR	Global translation matrix for corner r
ZS	Global translation matrix for corner s

Usage

LPMAP, LPMKC, LPMPRT, LPMSZD, LPMS1, LPMS2

LABELED COMMON LPMEM

This common block is used to store membrane element output matrices.

Declaration

```
CØMMØN /LPMEM/  CBAR(9,9)   , EBARTD(9)   ,  EBØT(9)    ,
               EMASS(12,12), EPSSIG(3,3), FFBAR(16,9),
                KBAR(9,9)   , KLPM(16,16), SFBAR(3,9)
REAL KBAR, KLPM
```

Contents

CBAR	Element damping matrix
EBARTD	Unassembled element deformations due to unit temperature change
EBØT	Deformations due to the difference between element joint temperature and the structural base temperature
EMASS	Element mass matrix
EPSSIG	Element strain matrix
FFBAR	Unassembled to global transformation matrix
KBAR	Element stiffness matrix
KLPM	Transformed element stiffness matrix
SFBAR	Element stress matrix

Usage

LPMAP, LPMFFB, LPMEBT, LPMPRM, LPMSM, LPMSZD, LPMZM

LABELED COMMON LPMV

This common block is used to store membrane element geometry and vector data.

Declaration

```
COMMON  /LPMV/  PQ(4)  , RQ(4)  , RS(4)  , PS(4)  ,
                AB(4)  , BB(4)  , CB(4)  , DB(4)  , EB(4) ,
                FK1    , FK2    , FK3    , FK4    , FK5   ,
                THETAP , THETAQ , THETAR , THETAS
```

Contents

PQ	Edge vector components and magnitude
RQ	Edge vector components and magnitude
RS	Edge vector components and magnitude
PS	Edge vector components and magnitude
AB	Unit vector components
BB	Unit vector components
CB	Unit vector components
DB	Unit vector components
EB	Unit vector components
FK1	K factor
FK2	K factor
FK3	K factor
FK4	K factor
FK5	K factor
THETAP	Corner angle
THETAQ	Corner angle
THETAR	Corner angle
THETAS	Corner angle

Usage

LPMAP, LPMFFB, LPMEBT, LPMG, LPMPRT, LPMSM, LPMSZD

LABELED COMMON MTRL

This common block is used to store the material property tables in their entirety.

Declaration

```
CØMMØN /MTRL/ CØEFS(660),LENCØF,LENDES,LNGTHS(14,20),MPTR(240),
              MRDES(240), PRPVAL(13)
```

Contents

CØEFS — Coefficients for all properties for all materials

LENCØF — Length of coefficients

LENDES — Length of MRDES

LNGTHS — Number of coefficients for each property and number of packed material description words for each material

MPTR — Pointers to first coefficients for all properties of all materials as stored in array COEFS, and pointers to the first packed material description word for all materials as stored in array MRDES

MRDES — Packed material description for all materials

PRPVAL — Temporary storage for properties for a material

PRPVAL(1)	Elastic modulas
PRPVAL(2)	Secant modulas at point A, Approximate yield point
PRPVAL(3)	True stress at point A
PRPVAL(4)	True stress at rupture
PRPVAL(5)	True strain at rupture
PRPVAL(6)	Poisson's ratio
PRPVAL(7)	Coefficient of thermal expansion
PRPVAL(8)	Mass density
PRPVAL(9)	Specific heat
PRPVAL(10)	Damping/stiffness ratio
PRPVAL(11)	Ramberg-Osgood coefficient
PRPVAL(12)	Slope tangent to Ramberg-Osgood curve at the origin
PRPVAL(13)	Elastic limit stress

Usage

BIRDG1, ACCPRP, LPBSM, LPCSM, MATDES, MTLMØD, PRØP, RAMØSG

LABELED COMMON WRK

This common block is used as scratch space by several routines.

Declaration

The largest extent declared for labeled common block WRK occurs in subroutine CNSTRN. The declaration in that routine is given here.

```
COMMON /WRK/ DIREC(3)    , DIRTBL(50,3), DUM(600),
             ICON(600,3), OBLIQ(300,3)
```

Contents

The content of this common region varies between modules. Source code listings should be consulted for specific information regarding content.

Usage

CNSTRN, DUMPMT, LPCK, LPCKC, LPMAP

SUBROUTINE ACCJZE

This routine computes the difference of the X, Y and Z coordinates of two joints.

Algorithm

The coordinates of the two joints are accessed in the joint coordinate table, array JØ, and the X, Y, Z coordinate differences, representing the components of a vector directed from the first joint to the second, are stored in array DIREC.

Input/Output

None

Argument List

JNTM	An integer scalar defining the first joint number
JNTN	An integer scalar defining the second joint number
DIREC	A real array for the direction vector

Labeled Common

JØRT

Subroutines Called

None

Error Detection

None

SUBROUTINE ACCPRP

This routine computes properties for a material as a function of a given temperature and computes the Ramberg-Osgood coefficients.

Algorithm

The first 10 values of the PRPVAL array are computed using subrouting HØRNER. The last three are computed by routine RAMØSG. IF the IERRØR flag is set non-zero prior to entry to this routine, no material properties are computed.

Input/Output

Diagnostics are written on tape 6.

Argument List

MR	An integer scalar defining the material reference number
NM	An integer scalar defining the number of materials
TMPT	A real scalar defining the temperature

Labeled Common

MTRL

Subroutines Called

HØRNER, RAMØSG

Error Detection

If a material reference number is greater than the maximum allowed, a diagnostic is printed, the IERRØR flag is set equal one, and a return is made to the calling routine.

SUBROUTINE CNSTRN

This routine reads the direction cosine data, Data Code 4, the constraint data, Data Code 5, from which tables are generated to indicate constrained degrees of freedom that are oblique as well as parallel to the global directions.

Algorithm

Data Code 4 is processed and the direction table is generated by calling subroutine DRCNUM.

Constraints, Data Code 5, are processed next one at a time. A data card is read and an ECHØ of the values are printed on tape 6. The data is checked for validity and if errors are found, the IERRØR flag is set and the program continues to check the remainder of the input. Subroutine ACCJZE is used to access the joint coordinate table and determine the direction cosines. The direction cosines are normalized and the direction vector checked for zero. The constraint number and the joint number are stored in the constraint table ICØN. Next the components are checked to see if the constraint is in the global directions. If so, the procedure is continued for the next constraint until all constraints are processed.

If the constraint is an oblique constraint, the direction cosines are stored into the oblique direction table ØBLIQ and the rest of constraints processed as above.

After all of the constraint data has been processed the constraint table is sorted using subroutine QKSØRT. A check is made for more than three entries per joint using routine ØVER3. The constraints and oblique table are written onto tape 16.

Input/Output

User input constraint data is read from tape 5, and the compiled constraint table is written onto tape 16. Tape 6 is used for the output of diagnostics.

Argument List

MAXJNT An integer scalar defining the maximum joint number in the structure

NCT An integer scalar defining the number of constraints

NØB An integer scalar defining the number of oblique reactions

LNPAGE An integer scalar defining the lines per page print limit

Labeled Common

MTRL

Subroutines Called

ACCJZE, DRCNUM, ØVER3, QKSØRT

Error Detection

Checks are made for any direction vectors equal to zero, invalid joint numbers, data out of sequence, invalid Data Code, number of constraints exceeding the program maximum limitation, constraint data missing, and the number of oblique constraints exceeding the maximum. A diagnostic is written for all errors, the IERRØR flag is set equal zero, and a return is made to the calling routine.

SUBROUTINE CØCALC

This routine produces a column matrix of material properties coefficients.

Algorithm

The MVP temperature/value pairs stored in arrays TVPA and VPT are used in the following equation to compute material property coefficients.

$$AP = (TVPA^T * TVPA)^{-1} * (TVPA^T * VPT)$$

Input/Output

Diagnostics are written on tape 6.

Argument List

MVP	An integer scalar defining the number of temperature/value pairs

Labeled Common

CØFCLC, IDEN, IERRØR, LIMITS

Subroutines Called

LTRMLT, MLTMTR, SID

Error Detection

If $(TVPA^T * TVPA)^{-1}$ is singular, a diagnostic is printed, the IERRØR flag is set equal to zero, and a return is made to the calling routine.

SUBROUTINE DRCNUM

This routine reads the direction cosine data, Data Code 4, and builds the direction table.

Algorithm

Data Code 4 cards are read and checked for valid data entry. An echo of the input values is printed on tape 6.

Subroutine ACCJZE is used to access the joint coordinate table and determine the direction cosines. A check is made of the direction vector to see if it is zero. If it is zero, the IERRØR flag is set and the program continues to scan the input for errors. Next the direction cosines are normalized and stored into the direction table DIRTBL. This procedure is continued until all Data Code 4 input has been processed.

Input/Output

Direction cosine data are read from tape 5 and diagnostics are written onto tape 6.

Argument List

DIRTBL	A real array for the direction table
MAXJNT	An integer scalar defining the maximum joint number in the structure
NDR	An integer scalar defining the number of records in the direction table
JERR4	An integer scalar error condition flag for the direction table

Labeled Common

IERRØR

Subroutines Called

ACCJZE

Error Detection

A check is made for all zero direction vectors, joint numbers out of range, direction cosine records not in sequence, invalid data code, and direction numbers exceeding the maximum allowable. For all errors a diagnostic is printed, the IERRØR flag is set to zero, and the input data scanned for further errors.

SUBROUTINE DUMPMT

This routine dumps all or part of the data on a standard FØRMAT tape.

Algorithm

The information contained on tape IN is read according to standard FØRMAT matrix data format and then printed on tape 6.

Input/Output

Data is read from tape IN and then printed on tape 6

Argument List

IN An integer scalar defining the logical unit number of the tape

IFLAG An integer scalar control flag which, when zero, specifies dump header records only and, when non-zero, specifies complete dump

Labeled Common

WRK

Subroutines Called

None

Error Detection

None

SUBROUTINE EDGDØF

This routine generates an insort table of unique edges and a table of edge degrees of freedom corresponding to the original order of the unsorted unique edge table.

Algorithm

If the IERRØR flag is set non-zero prior to entry to this routine, it is not executed. An immediate return is made to the calling routine.

Tape 17 containing the pairs of joints defining each edge is rewound and read into array IEDG. The edges are sorted according to the first joint using routine QKSØRT. The edges are compared one to another for duplication and only the unique edges are saved in array IEDG. The degree of freedom number for the edge is saved in array IDØF. The final table of edges, array IEDG, is sorted then back into the original sequence.

Input/Output

Tape 6 is used to output diagnostics, and edge data is read from tape 17.

Argument List

NWRT	An integer scalar defining the number of elements processed
NPRS	An integer scalar defining the number of edges processed
IUNQ	An integer scalar defining the number of unique edges

Labeled Common

EDØF, IEDGE, ISEQ

Subroutines Called

QKSØRT

Error Detection

If the number of edge pairs requested is not equal to the number of pairs read from tape, a diagnostic is printed, the IERRØR flag is set equal one, and a return is made to calling routine.

SUBROUTINE EDGES

This routine assembles the joint pairs that define the edges of an element.

Algorithm

The number of edges are determined according to the type of element. Each joint pair defining an edge is packed into a single word of array IEDGE according to the formula

IEDGE = J1 * 10000 + J2

where J1, the joint number of joint one, is less than J2, the joint number of joint two. When all edges for the element have been processed, they are written on tape 17 as one record for subsequent use by subroutine EDGDØF.

Input/Output

Tape 6 is used to output diagnostics. The edge data is output onto tape 16.

Argument List

ICØDE — An integer scalar defining the data code of the element
20 = Bar
30 = Membrane
40 = Cell
50 = Point Mass

JCN — An integer array containing element joint numbers

NWRT — An integer scalar defining the number of elements whose edges have been processed

NPRS — An integer scalar defining the number of joint pairs or edges processed

Labeled Common

IERRØR

Subroutine Called

None

Error Detection

If an invalid Data Code number is detected, the IERRØR flag is set equal to one, a diagnostic printed, and a return is made to the calling routine.

SUBROUTINE ELEMNT

This routine reads and checks the validity of element definition data.

Algorithm

The user input defining the elements, Data Codes 20, 30, 40 and 50, is read and the data checked for validity. If an error is found , appropriate error flags are set. The run is aborted only after all the input data is scanned for errors.

Input/Output

Tape units 5 and 6 are used for card input data and output diagnostic messages, respectively. An echo of the input is also output on tape 6.

Argument List

ICØDE	An integer scalar defining the Data Code number
NE	An integer scalar defining the element numbers
MAXJ	An integer scalar defining the number of joints
IEØD	An integer scalar flag set equal to one when the end of element data is encountered
LNPAGE	An integer scalar defining the lines per page print limit

Labeled Common

LPEP, IERRØR

Subroutines Called

None

Error Detection

A check is made for element numbers out of sequence, element type code out of sequence, zero area for bar elements, zero thickness for membrane elements, joint numbers out of range, and for joints not defined. For each error condition, a diagnostic is printed. The IERRØR flag is set equal to one but the scan of input data is allowed to continue.

SUBROUTINE HØRNER

This routine solves a monotonically increasing polynomial equation by Horner's method.

Algorithm

The polynomial is evaluated in the following manner:

$$V_1 = \text{CØEF(INUM)}*\text{TMPT} + \text{CØEF(INUM-1)}$$

$$\text{VAL} = \sum_{i=2}^{\text{INUM-1}} V_{i-1}\ \text{TMPT} + \text{CØEF(INUM-}i\text{)}$$

Input/Output

None

Argument List

CØEF	A real array of coefficients describing the polynomial
INUM	An integer scalar defining the number of coefficients
TMPT	A real scalar defining the independent variable
VAL	A real scalar defining the dependent variable

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE JTEMP

This routine reads joint temperature data, Data Code 3, and modifies the joint temperature table.

Algorithm

Temperature data is read and checked for validity, then the temperature table stored in array TEMP is updated accordingly.

Input/Output

User specified joint temperature data is read from tape 5 and diagnostics are written onto tape 6.

Argument List

None

Labeled Common

JØRT

Subroutines Called

None

Error Detection

Checks are made for input of a data code other than 3 and for the omission of the joint number if temperature data is present. If an error is detected, the IERRØR flag is set equal zero and a return is made to the calling routine.

SUBROUTINE JTPRT

This routine prints the contents of the joint coordinate table and the joint temperature table.

Algorithm

The contents of arrays JØ and TEMP in labeled common JZERØ are printed.

Input/Output

The joint coordinate and temperature data are written onto tape 6.

Argument List

LNPAGE An integer scalar defining the lines per page print limit

NJT An integer scalar defining the total number of joints

Labeled Common

JØRT

Subroutines Called

None

Error Detection

None

SUBROUTINE JZERØ

This routine reads user input joint coordinate and temperature data, Data Code 2, and forms the joint coordinate table and the temperature table.

Algorithm

Joint coordinate and temperature input data cards are read and checked for validity. The joint coordinate and the temperature tables are formed in arrays JØ and TEMP, respectively.

Input/Output

Joint coordinate and joint temperature data are read from tape 5 and diagnostics are written onto tape 6.

Argument List

None

Labeled Common

CØNST, IERRØR, JØRT

Subroutines Called

None

Error Detection

Checks are made for input of a data code other than 2 and for the omission of the joint number if coordinate data is present. If an error is detected, the IERRØR flag is set equal zero and a return is made to the calling routine.

SUBROUTINE LPBAR

This routine forms the matrices for lumped parameter bar elements.

Algorithm

Using the element geometry, physical properties and material properties stored in labeled common, the following matrix equations are evaluated. Matrix size is given in parentheses at the left.

(3 x 1) $$\overline{a} = \frac{\overline{pq}}{|\overline{pq}|}$$

(2 x 2) $$\overline{k} = \frac{2A}{C_k\,|pq|} \left[\begin{array}{c|c} 1.0 & \\ \hline & 1.0 \end{array}\right]$$

(2 x 2) $$\overline{c} = h\,\overline{k}$$

(7 x 2) $$F_{\overline{F}} = \left[\begin{array}{c|c} -a_x & \\ -a_y & \\ -a_z & \\ \hline & a_x \\ & a_y \\ & a_z \\ \hline 1.0 & -1.0 \end{array}\right]$$

(1 x 2) $$\sigma_{\overline{F}} = \left[\begin{array}{c|c} \frac{1}{2A} & \frac{1}{2A} \end{array}\right]$$

(1 x 1) $$\epsilon_\sigma = C_k$$

(2 x 1) $$\overline{e}_{T\Delta} = \frac{\alpha T\ |pq|}{8} \left[\begin{array}{c} 4.0 \\ \hline 4.0 \end{array}\right]$$

(2 x 1) $$\delta\bar{e}_T = \frac{\alpha T\ |pq|}{8}\ \frac{3(T_{JP}-T_B) + (T_{JQ}-TB)}{(T_{JP}-T_B) + 3(T_{JP}-T_B)}$$

(1 x 1) $$V = A\ |\overline{pq}|$$

(1 x 1) $$m_T = \rho\ V$$

(7 x 7) $$m = \frac{m_T}{6}\left[\begin{array}{ccc|ccc} 2 & & & 1 & & \\ & 2 & & & 1 & \\ & & 2 & & & 1 \\ \hline 1 & & & 2 & & \\ & 1 & & & 2 & \\ & & 1 & & & 2 \end{array}\right]$$

Input/Output

None

Argument List

TBASE A real scalar defining the base temperature of the structure

Labeled Common

JØRT, LPEP, LPBEM

Subroutines Called

LPBPRT, LPMK, VECT

Error Detection

None

AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3
AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)
DEC 77 R C MORRIS F33615-75-C-3105
UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

3 OF 6
ADA
063741

(2 x 1) $\delta\bar{e}_T = \frac{\alpha T\ |pq|}{8}\ \frac{3(T_{JP}-T_B) + (T_{JQ}-TB)}{(T_{JP}-T_B) + 3(T_{JP}-T_B)}$

(1 x 1) $V = A\ |\overline{pq}|$

(1 x 1) $m_T = \rho\ V$

(7 x 7) $m = \frac{m_T}{6} \left[\begin{array}{ccc|ccc} 2 & & & 1 & & \\ & 2 & & & 1 & \\ & & 2 & & & 1 \\ \hline 1 & & & 2 & & \\ & 1 & & & 2 & \\ & & 1 & & & 2 \end{array}\right]$

Input/Output

None

Argument List

TBASE A real scalar defining the base temperature of the structure

Labeled Common

JØRT, LPEP, LPBEM

Subroutines Called

LPBPRT, LPMK, VECT

Error Detection

None

SUBROUTINE LPBPRT

This routine prints the input and intermediate calculations for lumped parameter bar elements.

Algorithm

This routine is executed only in runs made for the purpose of program checkout. The intermediate calculations for the elements are printed as are the contents of labeled common LPEP and LPBEM.

Input/Output

Intermediate data is output on tape 6.

Argument List

None

Labeled Common

JØRT, LPBEM, LPEP

Subroutines Called

None

Error Detection.

None

SUBROUTINE LPBSM

This routine saves lumped parameter bar element data on external files.

Algorithm

The matrix partitions representing the contribution of this element to each of the output matrices are written onto the appropriate scratch tape.

Input/Output

The data output to scratch tapes is as follows:

Tape	Data
7	Element variable table
8	CBAR
9	EBØT
10	EMASS
11	EPSSIG
12	FFBAR
13	KBAR
14	KLPC
15	SFBAR
18	Element constant table

Argument List

NBAR An integer scalar defining the bar element number

TMPAVG A real scalar defining the element average temperature

Labeled Common

LPBEM, LPEP, MTRL

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCAP

This routine acts as the executive routine for computing the lumped parameter cell (approximate parallel piped) matrices.

Algorithm

Routine LPCG is invoked to compute cell element geometry. Routines LPCKC, LPCK and LPCFFB are then called to compute element matrices $\overline{k}$, k and $F_{\overline{F}}$, respectively. Using data stored in labeled common, the following matrix equations are then evaluated. Matrix size is given in parentheses at the left.

(12 x 12)

$$\varepsilon_{\sigma} = \left[\begin{array}{c|c} C_{k_1} & \\ \hline & C_{k_2} \end{array}\right]$$

(30 x 30)

$$\overline{c} = h\,\overline{k}$$

(36 x 36)

$$m = \frac{\rho v}{216}\left[\begin{array}{c|c} 2m_{LPM} & m_{LPM} \\ \hline m_{LPM} & 2m_{LPM} \end{array}\right]$$

Note: See routine LPMM for definition of m_{LPM}.

Routine LPCEBT is then called to compute matrices $\delta\overline{e}_T$ and $\overline{e}_{T\Delta}$.

Input/Output

None

Argument List

GC A real scalar defining the cell stiffness matrix suppression coefficient

Labeled Common

JØRT, LPCEM, LPCV, LPEP

Subroutines Called

LPCEBT, LPCFFB, LPCG, LPCK, LPCKC, LPCPRM, LPCPRT, LPCZM, LPMM

Error Detection

None

SUBROUTINE LPCC

This routine forms the compliance matrix for lumped parameter cell elements.

Algorithm

The compliance matrix for the upper and lower surfaces of the cell element are formed independently. These two matrices each of order 6 x 6 are then assembled as diagonal partitions of the 12 x 12 compliance matrix for the elemtnt. Each partition is formed as shown below where E is Young's Modulus and V is Poisson's Ratio.

$$C_K = \frac{1}{E}\begin{bmatrix} 1 & -V & -V & & & \\ -V & 1 & -V & & & \\ -V & -V & 1 & & & \\ & & & 2(1+V) & & \\ & & & & 2(1+V) & \\ & & & & & 2(1+V) \end{bmatrix}$$

Input/Output

None

Argument List

None

Labeled Common

LPEP

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCE3T

This routine calculates the unassembled element deformations due to unit temperature change at the joints defining a cell element and the initial thermal deformations for the element.

Algorithm

The theoretical definition of matrix $\bar{e}_{T\Delta}$ as a function of geometry, element K factors, and the coefficient of thermal expansion is as follows:

$$\bar{e}_{T\Delta} = \begin{array}{c} \begin{array}{cccccccc} p_o & q_o & r_o & s_o & p_1 & q_1 & r_1 & s_1 \end{array} \\ \left[\begin{array}{c|c} \begin{array}{c} E_{11} \\ (9 \times 4) \end{array} & \begin{array}{c} E_{12} \\ (9 \times 4) \end{array} \\ \hline \begin{array}{c} E_{21} \\ (9 \times 4) \end{array} & \begin{array}{c} E_{22} \\ (9 \times 4) \end{array} \\ \hline \multicolumn{2}{c}{E_3 \ (4 \times 8)} \\ \hline \multicolumn{2}{c}{E_4 \ (8 \times 8)} \end{array} \right] \end{array}$$

where the partitions E_{12}, E_{21}, and E_3 are null and

$$E_{11} = \begin{bmatrix} b_1 \cos\theta_1 & b_2 \cos\theta_2 & K_{3,6} b_3 \cos\theta_3 & K_{3,6} b_4 \cos\theta_4 \\ & a_2 & & \\ a_1 & & & \\ & & a_3 & \\ & & & a_4 \\ & b_2 & & \\ & & b_3 & \\ b_1 & & & \\ & & & b_4 \end{bmatrix}$$

$$E_{22} = \begin{bmatrix} b_5 \cos \theta_5 & b_6 \cos \theta_6 & K_{3,7} b_7 \cos \theta_7 & K_{3,7} b_8 \cos \theta_8 \\ & a_6 & & \\ a_5 & & & \\ & & a_7 & \\ & & & a_8 \\ & b_6 & & \\ & & b_7 & \\ b_5 & & & \\ & & & b_8 \end{bmatrix}$$

$$E_4 = \begin{bmatrix} t_1 & & & & & & & \\ & & & & t_5 & & & \\ & t_2 & & & & & & \\ & & & & & t_6 & & \\ & & t_3 & & & & & \\ & & & & & & t_7 & \\ & & & t_4 & & & & \\ & & & & & & & t_8 \end{bmatrix}$$

However, the matrices $\bar{e}_{T\Delta}$ and $\delta\bar{e}_T$ used by the code are single column matrices. Matrix $\delta\bar{e}_T$, initial thermal deformations, is formed by multiplying the terms in each column shown above by the initial thermal gradient of the referenced joints and summing the columns. Matrix $\bar{e}_{T\Delta}$ used by the code represents unassembled element deformations due to an average unit joint temperature change for the element. It is formed by simply summing the columns shown above. Subsequently, this column matrix is scaled by the average joint temperature change of the element.

Input/Output

None

Argument List

None

Labeled Common

LPCEM, LPCV, LPEP

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCFFB

This routine generates the force transformation matrix for lumped parameter cell elements.

Algorithm

The transformation matrix $F_{\bar{\bar{F}}}$ for a cell element is of dimension 36 x 30 and is given by the matrix equation

$$F_{\bar{\bar{F}}} = (C_{F'}\ F'_{\hat{F}} + C_{Q'}\ Q'_{\hat{F}})\ \hat{F}_{\bar{\bar{F}}}$$

Matrices $C_{F'}$, $F'_{\hat{F}}$, $C_{Q'}$, $Q'_{\hat{F}}$, and $\hat{F}_{\bar{\bar{F}}}$ are functions of the unit edge vectors a, b, c, d, e, f, the thickness proportionality factors α, β, γ, and the panel K factors. Matrix $F_{\bar{\bar{F}}}$ is assembled directly from this data as opposed to assembling these matrices and solving the matrix equation given above.

For the sake of presentation so that each element of $F_{\bar{\bar{F}}}$ can be shown, we can partition $F_{\bar{\bar{F}}}$ as follows

$$F_{\bar{\bar{F}}} = \left[\begin{array}{c|c} F_{\bar{\bar{F}}_1} & F_{\bar{\bar{F}}_2} \\ (24 \times 15) & (24 \times 15) \\ \hline F_{\bar{\bar{F}}_3} & F_{\bar{\bar{F}}_4} \\ (12 \times 15) & (12 \times 15) \end{array}\right]$$

where

$$F_{\overline{F}_1} = \begin{bmatrix}
 & & -a_{o_x}{}^{\alpha} & & & & & -d_{o_x}{}^{\alpha} & & & & -a_{o_x}{}^{\beta} & & & \\
 & & -a_{o_y}{}^{\alpha} & & & & & -d_{o_y}{}^{\alpha} & & & & -a_{o_y}{}^{\beta} & & & \\
 & & -a_{o_z}{}^{\alpha} & & & & & -d_{o_z}{}^{\alpha} & & & & -a_{o_z}{}^{\beta} & & & \\
e_{o_x}K_{10}{}^{\alpha} & a_{o_x}{}^{\alpha} & & & & b_{o_x} & & & & & a_{o_x}{}^{\beta} & & & & b_{o_x}{}^{\beta} \\
e_{o_y}K_{10}{}^{\alpha} & a_{o_y}{}^{\alpha} & & & & b_{o_y}{}^{\alpha} & & & & & a_{o_y}{}^{\beta} & & & & b_{o_y}{}^{\beta} \\
e_{o_z}K_{10}{}^{\alpha} & a_{o_z}{}^{\alpha} & & & & b_{o_z}{}^{\alpha} & & & & & a_{o_z}{}^{\beta} & & & & b_{o_z}{}^{\beta} \\
 & & & -c_{o_x}{}^{\alpha} & & & -b_{o_x}{}^{\alpha} & & & & & & -a_{o_x}{}^{\beta} & & \\
 & & & -c_{o_y}{}^{\alpha} & & & -b_{o_y}{}^{\alpha} & & & & & & -a_{o_y}{}^{\beta} & & \\
 & & & -c_{o_z}{}^{\alpha} & & & -b_{o_z}{}^{\alpha} & & & & & & -a_{o_z}{}^{\beta} & & \\
f_{o_x}K_{40}{}^{\alpha} & & & & c_{o_x}{}^{\alpha} & & & & d_{o}{}^{\alpha} & & & & & c_{o_x}{}^{\beta} & \\
f_{o_y}K_{40}{}^{\alpha} & & & & c_{o_y}{}^{\alpha} & & & & d_{o_y}{}^{\alpha} & & & & & c_{o_y}{}^{\beta} & \\
f_{o_z}K_{40}{}^{\alpha} & & & & c_{o_z}{}^{\alpha} & & & & d_{o_z}{}^{\alpha} & & & & & c_{o_z}{}^{\beta} & \\
 & & -a_{1_x}{}^{\beta} & & & & & -d_{1_x}{}^{\beta} & & & & -a_{1_x}{}^{\alpha} & & & \\
 & & -a_{1_y}{}^{\beta} & & & & & -d_{1_y}{}^{\beta} & & & & -a_{1_y}{}^{\alpha} & & & \\
 & & -a_{1_z}{}^{\beta} & & & & & -d_{1_z}{}^{\beta} & & & & -a_{1_z}{}^{\alpha} & & & \\
e_{o_x}K_{11}{}^{\beta} & a_{1_x}{}^{\beta} & & & & b_{1_x}{}^{\beta} & & & & e_{o_x}K_{11}{}^{\alpha} & a_{1_x}{}^{\alpha} & & & & b_{1_x}{}^{\alpha} \\
e_{o_y}K_{11}{}^{\beta} & a_{1_y}{}^{\beta} & & & & b_{1_y}{}^{\beta} & & & & e_{o_y}K_{11}{}^{\alpha} & a_{1_y}{}^{\alpha} & & & & b_{1_y}{}^{\alpha} \\
e_{o_z}K_{11}{}^{\beta} & a_{1_z}{}^{\beta} & & & & b_{1_z}{}^{\beta} & & & & e_{o_z}K_{11}{}^{\alpha} & a_{1_z}{}^{\alpha} & & & & b_{1_z}{}^{\alpha} \\
 & & & -c_{1_x}{}^{\beta} & & & & -b_{1_x}{}^{\beta} & & & & & c_{1_x}{}^{\alpha} & & \\
 & & & -c_{1_y}{}^{\beta} & & & & -b_{1_y}{}^{\beta} & & & & & c_{1_y}{}^{\alpha} & & \\
 & & & -c_{1_z}{}^{\beta} & & & & -b_{1_z}{}^{\beta} & & & & & c_{1_z}{}^{\alpha} & & \\
f_{o_x}K_{41}{}^{\beta} & & & & c_{1_x}{}^{\beta} & & & & d_{1_x}{}^{\beta} & f_{o_x}K_{41}{}^{\alpha} & & & & c_{1_x}{}^{\alpha} & \\
f_{o_y}K_{41}{}^{\beta} & & & & c_{1_y}{}^{\beta} & & & & d_{1_y}{}^{\beta} & f_{o_y}K_{41}{}^{\alpha} & & & & c_{1_y}{}^{\alpha} & \\
f_{o_z}K_{41}{}^{\beta} & & & & c_{1_z}{}^{\beta} & & & & d_{1_z} & f_{o_y}K_{41}{}^{\alpha} & & & & c_{1_y}{}^{\alpha} &
\end{bmatrix}$$

$$F_{\overline{F}_2} = \left[\begin{array}{ccccccccccccccc}
 & -d_{o_x}\beta & & & & & -a_{o_x}K_{15}\gamma & d_{2x} & & & & & & & \\
 & -d_{o_y}\beta & & & & & -a_{o_y}K_{15}\gamma & d_{2y} & & & & & & & \\
 & -d_{o_z}\beta & & & & & -a_{o_z}K_{15}\gamma & d_{2z} & & & & & & & \\
 & & & b_{o_x}K_{12}\gamma & & & & & & -e_{o_x} & & & & & \\
 & & & b_{o_y}K_{12}\gamma & & & & & & -e_{o_y} & & & & & \\
 & & & b_{o_z}K_{12}\gamma & & & & & & -e_{o_z} & & & & & \\
-b_{o_x}\beta & & & & -c_{o_x}K_{13}\gamma & & & & & & & b_{3x} & & & \\
-b_{o_y}\beta & & & & -c_{o_y}K_{13}\gamma & & & & & & & b_{3y} & & & \\
-b_{o_z}\beta & & & & -c_{o_z}K_{13}\gamma & & & & & & & b_{3z} & & & \\
 & & d_{o_x}\beta & & & d_{o_y}K_{14}\gamma & & & & & & & & -f_{o_x} & \\
 & & d_{o_y}\beta & & & d_{o_y}K_{14}\gamma & & & & & & & & -f_{o_y} & \\
 & & d_{o_z}\beta & & & d_{o_z}K_{14}\gamma & & & & & & & & -f_{o_z} & \\
 & -d_{1_x}\alpha & & -d_{1_x}K_{42}\gamma & & & & & d_{2_x} & & & & & & \\
 & -d_{1_y}\alpha & & -d_{1_y}K_{42}\gamma & & & & & d_{2_y} & & & & & & \\
 & -d_{1_z}\alpha & & -d_{1_z}K_{42}\gamma & & & & & d_{2_z} & & & & & & \\
 & & & & a_{1_x}K_{43}\gamma & & & & & & e_{o_x} & & & & \\
 & & & & a_{1_y}K_{43}\gamma & & & & & & e_{o_y} & & & & \\
 & & & & a_{1_z}K_{43}\gamma & & & & & & e_{o_z} & & & & \\
 & b_{1_x}\alpha & & -b_{1_x}K_{42}\gamma & & & & & & & & & -b_{3_x} & & \\
 & b_{1_y}\alpha & & -b_{1_y}K_{42}\gamma & & & & & & & & & -b_{3_y} & & \\
 & b_{1_z}\alpha & & -b_{1_z}K_{42}\gamma & & & & & & & & & -b_{3_z} & & \\
 & & d_{1_x}\alpha & & & & c_{1_x}K_{45}\gamma & & & & & & & & f_{o_x} \\
 & & d_{1_y}\alpha & & & & c_{1_x}K_{45}\gamma & & & & & & & & f_{o_y} \\
 & & d_{1_z}\alpha & & & & c_{1_z}K_{45}\gamma & & & & & & & & f_{o_z}
\end{array}\right]$$

$$F_{\bar{F}_3} = \begin{bmatrix}
\alpha & -\alpha & \alpha & & & & & & & \beta & -\beta & \beta & & & \\
\alpha K_{20} & & & & & -\alpha & \alpha & & & \beta K_{20} & & & & & -\beta \\
\alpha K_{30} & & & \alpha & -\alpha & & & & & \beta K_{30} & & & \alpha & -\alpha & \\
\alpha K_{50} & & & & & & & \alpha & -\alpha & \beta K_{50} & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
\beta & -\beta & \beta & & & & & & & \alpha & -\alpha & \alpha & & & \\
\beta K_{21} & & & & & -\beta & \beta & & & \alpha K_{21} & & & & & -\alpha \\
\beta K_{31} & & & \beta & -\beta & & & & & \alpha K_{31} & & & -\alpha & \alpha & \\
\beta K_{51} & & & & & & & \beta & -\beta & \alpha K_{51} & & & & &
\end{bmatrix}$$

$$F_{\bar{F}_4} = \begin{bmatrix}
 & & & \gamma & & & & & & & & & & & \\
\beta & & & & -\gamma & & & & & & & & & & \\
 & & & & & \gamma & & & & & & & & & \\
 & \beta & -\beta & & & & -\gamma & & & & & & & & \\
 & & & -\gamma K_{52} & & & \gamma K_{25} & -1 & 1 & & & & & & \\
 & & & \gamma K_{22} & -\gamma K_{53} & & & & & -1 & 1 & & & & \\
 & & & & \gamma K_{23} & -\gamma K_{54} & & & & & & -1 & 1 & & \\
 & & & & & \gamma K_{24} & -\gamma K_{55} & & & & & & & -1 & 1 \\
 & & & -\gamma K_{32} & & & & & & & & & & & \\
\alpha & & & & \gamma K_{33} & & & & & & & & & & \\
 & & & & & -\gamma K_{34} & & & & & & & & & \\
 & \alpha & -\alpha & & & & \alpha K_{35} & & & & & & & &
\end{bmatrix}$$

Input/Output

None

Argument List

FFBAR	A real array for the output transformation matrix for the element

Labeled Common

LPCV, LPEP

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCG

This routine computes vector geometry for lumped parameter cell (approximate parallelpiped) elements.

Algorithm

Thickness Vectors

$$\overline{t}_p = \overline{p}_1 - \overline{p}_0$$

$$\overline{t}_q = \overline{q}_1 - \overline{q}_0$$

$$\overline{t}_r = \overline{r}_1 - \overline{r}_0$$

$$\overline{t}_s = \overline{s}_1 - \overline{s}_0$$

$$\overline{t}_{pq} = \overline{t}_q - \overline{t}_p$$

$$\overline{t}_{rq} = \overline{t}_q - \overline{t}_r$$

$$\overline{t}_{rs} = \overline{t}_s - \overline{t}_r$$

$$\overline{t}_{ps} = \overline{t}_s - \overline{t}_p$$

Surface 0	Surface 1
$\overline{pq}_0 = \overline{q}_0 - \overline{p}_0$	$\overline{pq}_1 = \overline{q}_1 - \overline{p}_1$
$\overline{rq}_0 = \overline{q}_0 - \overline{r}_0$	$\overline{rq}_1 = \overline{q}_1 - \overline{r}_1$
$\overline{rs}_0 = \overline{s}_0 - \overline{r}_0$	$\overline{rs}_1 = \overline{s}_1 - \overline{r}_1$
$\overline{ps}_0 = \overline{s}_0 - \overline{p}_0$	$\overline{ps}_1 = \overline{s}_1 - \overline{p}_1$
$\overline{a}_0 = \overline{pq}_0 / \lvert\overline{pq}_0\rvert$	$\overline{a}_1 = \overline{pq}_1 / \lvert\overline{pq}_1\rvert$

Surface 0

$\overline{b}_0 = \overline{rq}_0 \,/\, |\overline{rq}_0|$

$\overline{c}_0 = \overline{rs}_0 \,/\, |\overline{rs}_0|$

$\overline{d}_0 = \overline{ps}_0 \,/\, |\overline{ps}_0|$

$\overline{e}_0 = \overline{t}_q \,/\, |\overline{t}_q|$

$\overline{f}_0 = \overline{t}_s \,/\, |\overline{t}_s|$

Surface 1

$\overline{b}_1 = \overline{rq}_1 \,/\, |\overline{rq}_1|$

$\overline{c}_1 = \overline{rs}_1 \,/\, |\overline{rs}_1|$

$\overline{d}_1 = \overline{ps}_1 \,/\, |\overline{ps}_1|$

$\overline{e}_1 = \overline{e}_0$

$\overline{f}_1 = \overline{f}_0$

Surface 2

$\overline{pq}_2 = \overline{pq}_0$

$\overline{rq}_2 = -\overline{t}_q$

$\overline{rs}_2 = -\overline{pq}_1$

$\overline{ps}_2 = -\overline{t}_p$

$\overline{a}_2 = \overline{a}_0$

$\overline{b}_2 = -\overline{e}_0$

$\overline{c}_2 = -\overline{a}_1$

$\overline{d}_2 = \overline{t}_p \,/\, |\overline{t}_p|$

$\overline{e}_2 = \overline{b}_0$

$\overline{f}_2 = -\overline{d}_1$

Surface 3

$\overline{pq}_3 = -\overline{rq}_0$

$\overline{rq}_3 = -\overline{t}_r$

$\overline{rs}_3 = \overline{rq}_1$

$\overline{ps}_3 = \overline{t}_q$

$\overline{a}_3 = -\overline{b}_0$

$\overline{b}_3 = \overline{t}_r \,/\, |\overline{t}_r|$

$\overline{c}_3 = \overline{b}_1$

$\overline{d}_3 = -\overline{e}_0$

$\overline{e}_3 = -\overline{c}_0$

$\overline{f}_3 = \overline{a}_1$

Surface 4

$$\begin{aligned}
\overline{pq}_4 &= \overline{rs}_0 \\
\overline{rq}_4 &= -\bar{t}_s \\
\overline{rs}_4 &= -\overline{rs}_1 \\
\overline{ps}_4 &= \bar{t}_r \\
\bar{a}_4 &= \bar{c}_0 \\
\bar{b}_4 &= -\bar{f}_0 \\
\bar{c}_4 &= -\bar{c}_1 \\
\bar{d}_4 &= -\bar{f}_3 \\
\bar{e}_4 &= \bar{d}_0 \\
\bar{f}_4 &= -\bar{b}_s
\end{aligned}$$

Surface 5

$$\begin{aligned}
\overline{pq}_5 &= -\overline{pr}_0 \\
\overline{rq}_5 &= -\bar{t}_0 \\
\overline{rs}_5 &= \overline{ps}_1 \\
\overline{ps}_5 &= \bar{t}_s \\
\bar{a}_5 &= -\bar{d}_0 \\
\bar{b}_5 &= -\bar{d}_2 \\
\bar{c}_5 &= \bar{d}_1 \\
\bar{d}_5 &= -\bar{f}_0 \\
\bar{e}_5 &= -\bar{a}_0 \\
\bar{f}_5 &= \bar{c}_p
\end{aligned}$$

Surface 6

$$\begin{aligned}
\overline{pq}_6 &= \overline{pq}_0 + \beta\bar{t}_{pq} \\
\overline{rq}_6 &= \overline{rq}_0 + \beta t_{rq} \\
\overline{rs}_6 &= \overline{rs}_0 + \beta t_{rs} \\
\overline{ps}_6 &= \overline{ps}_0 + \beta t_{ps} \\
\bar{a}_6 &= \overline{pq}_6 \; / \; |\overline{pq}_6|
\end{aligned}$$

Surface 7

$$\begin{aligned}
\overline{pq}_7 &= \overline{pq}_0 + \alpha t_{pq} \\
\overline{rq}_7 &= \overline{rq}_0 + \alpha t_{rq} \\
\overline{rs}_7 &= \overline{rs}_0 + \alpha t_{rs} \\
\overline{ps}_7 &= \overline{ps}_0 + \alpha t_{ps} \\
\bar{a}_7 &= \overline{pq}_7 \; / \; |\overline{pq}_7|
\end{aligned}$$

Surface 6 Surface 7

$$\overline{b}_6 = \overline{rq}_6 \;/\; |\overline{rq}_6| \qquad \overline{b}_7 = \overline{rq}_7 \;/\; |\overline{rq}_7|$$

$$\overline{c}_6 = \overline{rs}_6 \;/\; |\overline{rs}_6| \qquad \overline{c}_7 = \overline{rs}_7 \;/\; |\overline{rs}_7|$$

$$\overline{d}_6 = \overline{ps}_6 \;/\; |\overline{ps}_6| \qquad \overline{d}_7 = \overline{ps}_7 \;/\; |\overline{ps}_7|$$

$$\overline{e}_6 = \overline{e}_0 \qquad \overline{e}_7 = \overline{e}_0$$

$$\overline{f}_6 = \overline{f}_0 \qquad \overline{f}_7 = \overline{f}_0$$

Sub-element Geometry

$$l_{a1} = 0.5 \; |\overline{pq}_6| \qquad l_{b1} = 0.5 \; |\overline{ps}_6|$$

$$l_{a2} = 0.5 \; |\overline{pq}_6| \qquad l_{b2} = 0.5 \; |\overline{rq}_6|$$

$$l_{a3} = 0.5 \; |\overline{rs}_6| \qquad l_{b3} = 0.5 \; |\overline{rq}_6|$$

$$l_{a4} = 0.5 \; |\overline{rs}_6| \qquad l_{b4} = 0.5 \; |\overline{ps}_6|$$

$$l_{a5} = 0.5 \; |\overline{pq}_7| \qquad l_{b5} = 0.5 \; |\overline{ps}_7|$$

$$l_{a6} = 0.5 \; |\overline{pq}_7| \qquad l_{b6} = 0.5 \; |\overline{rq}_7|$$

$$l_{a7} = 0.5 \; |\overline{rs}_7| \qquad l_{a7} = 0.5 \; |\overline{rq}_7|$$

$$l_{a8} = 0.5 \; |\overline{rs}_7| \qquad l_{b8} = 0.5 \; |\overline{ps}_7|$$

$$t_1 = t_5 = \frac{1}{32}\left(9\overline{t}_p + 3\overline{t}_q + \overline{t}_r + 3\overline{t}_s\right)$$

$$t_2 = t_6 = \frac{1}{32}\left(3\overline{t}_p + 9\overline{t}_q + 3\overline{t}_r + \overline{t}_s\right)$$

$$t_3 = t_7 \quad \frac{1}{32}\left(\overline{t}_p + 3\overline{t}_q + 9\overline{t}_r + 3\overline{t}_s\right)$$

$$t_4 = t_8 = \frac{1}{32}\left(3\overline{t}_p + \overline{t}_q + 3\overline{t}_r + 9\overline{t}_s\right)$$

K Factors

$$K_{1,i} = \frac{(\bar{a}_i \times \overline{rq}_i) \cdot \bar{c}_i}{(\bar{e}_i \times \overline{rq}_i) \cdot \bar{c}_i}$$

$$K_{2,i} = \frac{-(\bar{a}_i \times \overline{ps}_i) \cdot \bar{f}_i - K_{1,i}\,[\bar{e}_i \times (\overline{rs}_i - \overline{rs}_i)] \cdot \bar{f}_i}{(\bar{b}_i \times \overline{rs}_i) \cdot \bar{f}_i}$$

$$K_{3,i} = \frac{-K_{1,i}(\bar{e}_i \times \overline{pq}_i) \cdot \bar{f}_i - K_{2,i}(\bar{b}_i \times \overline{pq}_i) \cdot \bar{f}_i}{(\bar{c}_i \times \overline{ps}_i) \cdot \bar{f}_i}$$

$$K_{4,i} = \frac{-K_{3,i}(\bar{c}_i \times \overline{ps}_i) \cdot \bar{a}_i}{(\bar{f}_i \times \overline{ps}_i) \cdot \bar{a}_i}$$

$$\begin{aligned} K_{5,i} = \; & -(\bar{a}_i \cdot \bar{d}_i) - K_{1,i}(\bar{e}_i \cdot \bar{d}_i) - K_{2,i}(\bar{b}_i \cdot \bar{d}_i) \\ & -K_{3,i}(\bar{c}_i \cdot \bar{d}_i) - K_{4,i}(\bar{f}_i \cdot \bar{d}_i) \end{aligned}$$

where $i = 0,1,2,\cdots 7$

Corner Angles

$$\theta_1 = \cos^{-1}(-\bar{a}_6 \cdot \bar{d}_6)$$

$$\theta_2 = \cos^{-1}(\ \bar{a}_6 \cdot \bar{b}_6)$$

$$\theta_3 = \cos^{-1}(-\bar{c}_6 \cdot \bar{b}_6)$$

$$\theta_4 = \cos^{-1}(\ \bar{c}_6 \cdot \bar{d}_6)$$

$$\theta_5 = \cos^{-1}(-\bar{a}_7 \cdot \bar{d}_7)$$

$$\theta_6 = \cos^{-1}(\ \bar{a}_7 \cdot \bar{b}_7)$$

$$\theta_7 = \cos^{-1}(-\bar{c}_7 \cdot \bar{b}_7)$$

$$\theta_8 = \cos^{-1}(\bar{c}_7 \cdot \bar{d}_7)$$

$$\zeta_1 = \zeta_2 = \zeta_5 = \zeta_6 = \zeta_{pq}$$

$$\zeta_3 = \zeta_4 = \zeta_{pq} = (\theta_1 = \theta_4)$$

$$\zeta_5 = \zeta_6 = \zeta_{pq} = (\theta_5 = \theta_8)$$

Volumn and Mass

$$V = \frac{1}{6}[(\overline{pq}_0 \times \overline{ps}_0) \cdot \bar{t}_p + (\overline{rs}_0 \times \overline{rq}_0) \cdot \bar{t}_r$$
$$+ (\overline{rq}_1 \times \overline{pq}_1) \cdot \bar{t}_q + (\overline{ps}_1 \times \overline{rs}_1) \cdot \bar{t}_s$$
$$+ (\overline{pq}_0 - \overline{ps}_0) \times (\bar{t}_s - \overline{rs}_1)] \cdot (\bar{t}_s - \overline{ps}_1)$$

$$M = \rho V$$

Input/Output

None

Argument List

None

Labeled Common

JØRT, LPCV, LPEP

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPCK

This routine computes the transformed element stiffness matrix for lumper parameter cell elements.

Algorithm

The following matrix equation is evaluated using data stored in labeled common. The size of the resulting matrix is given in parentheses on the left.

(36 x 36) $$k = F_{\bar{F}} \bar{k} F_{\bar{F}}^T$$

Input/Output

None

Argument List

None

Labeled Common

LPCEM, WRK

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCKC

This routine forms the lumped parameter cell element stiffness and stress matrices.

Algorithm

The following matrix equations are evaluated using data stored in labeled common. Matrix dimensions are given in parentheses at the left.

$$(30 \times 30) \qquad \bar{K} = \left[\sum_{k=1}^{4} (v_k \, \sigma_{\bar{F}}^{T} \, C_{k1} \, \sigma_{\bar{F}}) + \sum_{k=5}^{8} (v_k \, \sigma_{\bar{F}}^{T} \, C_{k2} \, \sigma_{\bar{F}}) \right]^{-1}$$

$$(12 \times 30) \qquad \sigma_{\bar{F}} \quad \begin{bmatrix} \left(\sum_{k=1}^{4} v_k \, \sigma_{\bar{F}k} \right) \Big/ \sum_{k=1}^{4} v_k \\ \left(\sum_{k=5}^{8} v_k \, \sigma_{\bar{F}k} \right) \Big/ \sum_{k=5}^{8} v_k \end{bmatrix}$$

Input/Output

Diagnostics are written onto tape 6.

Argument List

KB A real array used to store the LPC stiffness matrix

GC An real scalar defining the cell stiffness matrix suppression coefficient

KCFLAG An integer scalar control flag which, if zero, signals generation of matrices KBAR and SPBAR, and if one, signals generation of CBAR

Labeled Common

IERRØR, LPCEM, LPCV, LPEP, WRK

Subroutines Called

LPCPRT, SID, WRMAT

Error Detection

If SID cannot invert KBAR, a diagnostic is printed, intermediate calculations dumped, and the IERRØR flag set equal one. Return is made to the calling routine. The loss of significant digits is checked and, if less than two, a warning diagnostic is printed and execution is allowed to continued.

SUBROUTINE LPCPRM

This routine prints the output matrices for lumped parameter cell elements.

Algorithm

Matrix partitions for cell elements and the contents of labeled common LPCEM are printed.

Input/Output

Matrix data for cell elements are written onto tape 6.

Argument List

None

Labeled Common

LPCEM

Error Detection

None

SUBROUTINE LPCPRT

This routine prints the input and intermediate calculations for lumped parameter cell elements.

Algorithm

Intermediate cell element calculations and the contents of labeled common LPEP and LPCV are printed.

Input/Output

Intermediate cell element data are written onto tape 6.

Argument List

None

Labeled Common

LPEP, LPCV, JØRT

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCSM

This routine saves lumped parameter cell element data on external files.

Algorithm

The matrix partitions representing the contribution of this element to each of the output matrices are written onto the appropriate scratch tape.

Input/Output

The data output to scratch tapes is as follows:

Tape	Data
7	Element variable table
8	CBAR
9	EBØT
10	EMASS
11	EPSSIG
12	FFBAR
13	KBAR
14	KLPC
15	SFBAR
18	Element constant table

Argument List

NCELL An integer scalar defining the bar element number

TMPAVG A real scalar defining the element average temperature

Labeled Common

LPCEM, LPCV, LPEP, MTRL

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCZM

This routine initializes the matrix arrays for lumped parameter cell elements.

Algorithm

All arrays in labeled common block LPCEM are set to zero.

Input/Output

None

Argument List

None

Labeled Common

LPCEM

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMAP

This routine acts as the executive routine for computing the lumped parameter membrane (approxiate parallelogram) matrices.

Algorithm

Routine LPMG is called to compute membrane element geometry. Routine LPMSZD is then invoked to form the p, q, r, and s components of matrices S, Z and $\tilde{D}$. Routines LPMKC, LPMK and LPMFFB are then called to assemble matrices $\overline{k}$, k and $F_{\overline{F}}$, respectively.

Using the compliance and stiffness matrices C_k and $\overline{k}$, the code forms the matrices ε_σ and $\overline{c}$. Routine LPMM is then called to compute the element mass matrix, m. Finally, routine LPMEBT is called to form matrices $\delta\overline{e}_T$ and $\overline{e}_{T\Delta}$.

Input/Output

None

Argument List

None

Labeled Common

JØRT, LPEP, LPMDSZ, LPMEM, LPMV

Subroutines Called

LPMEBT, LPMFFB, LPMG, LPMK, LPMKC, LPMM, LPMPRM, LPMPRT, LPMSZD, LPMZM

Error Detection

None

SUBROUTINE LPMC

This routine forms the compliance matrix for lumped parameter membrane elements.

Algorithm

The compliance matrix C_k of order 3 x 3 is formed from the material properties of the element. Using Young's Modulus, E, and Poisson's Ratio, NU, matrix C_k is assembled as shown below.

$$C_k = \frac{1}{E} \begin{bmatrix} 1 & -NU & \\ -NU & 1 & \\ & & 2(1 + NU) \end{bmatrix}$$

Input/Output

None

Argument List

None

Labeled Common

LPEP

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMD

This routine generates the $\tilde{D}$ matrices for lumped parameter parallelogram elements.

Algorithm

The following matrix equation is evaluated to form the matrix $\tilde{D}$ of order 3 x 3.

$$\tilde{D} = \frac{A}{t} (S^T \; Z^T \; C_k \; ZS)$$

Input/Output

None

Argument List

AØT	A real scalar equal to A*B*SIN(THETA)/T
C	A real array for the compliance matrix
D	A real array for the $\tilde{D}$ matrix
S	A real array for the skew stress matrix
Z	A real array for the global translation matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMEBT

This routine calculates unassembled element deformations due to unit temperature change at the joints defining a membrane element and the initial thermal deformations of the element.

Algorithm

The theoretical definition of matrix $\bar{e}_{T\Delta}$ as a function of geometry and the coefficient of thermal expansion is as follows:

$$\bar{e}_{T\Delta} = \frac{\alpha T}{8} \begin{array}{c} \begin{array}{cccc} p & q & r & s \end{array} \\ \left[\begin{array}{cccc} L_2 & 3L_2 & & \\ 3L_2 & L_2 & & \\ & & 3L_4 & L_4 \\ & & L_4 & 3L_4 \\ & 3L_6 & L_6 & \\ & L_6 & 3L_6 & \\ 3L_8 & & & L_8 \\ L_8 & & & 3L_8 \end{array}\right] \end{array}$$

where

$$L_2 = |\overline{pq}|$$
$$L_4 = |\overline{rs}|$$
$$L_6 = |\overline{rq}|$$
$$L_8 = |\overline{ps}|$$

However, matrices $\bar{e}_{T\Delta}$ *and* $\delta\bar{e}_T$ *used by the code are single column* matrices. Matrix $\delta\bar{e}_T$, initial thermal deformations, is formed by multiplying the terms in each column shown above by the initial thermal gradient at the referenced joints and summing the columns. Matrix

$\overline{e}_{T\Delta}$ used by the code represents unassembled element deformations due to an average unit joint temperature change for the element. It is formed by simply summing the columns shown above. Subsequently, this column matrix is scaled by the average joint temperature change for the element.

Input/Output

None

Argument List

None

Labeled Common

LPEP, LPMEM, LPMV

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMFFB

This routine generates the transformation matrix for the lumped parameter membrane element.

Algorithm

The transformation matrix $F_{\overline{F}}$ of order 16 x 9 is assembled as shown below.

$$F_{\overline{F}} = \begin{bmatrix}
 & & -a_x & & & & & -d_x & \\
 & & -a_y & & & & & -d_y & \\
 & & -a_z & & & & & -d_z & \\
e_xK_1 & a_x & & & & -b_x & & & \\
e_xK_1 & a_y & & & & -b_y & & & \\
e_zK_1 & a_z & & & & -b_z & & & \\
 & & & -c_x & & & -b_x & & \\
 & & & -c_y & & & -b_y & & \\
 & & & -c_z & & & -b_z & & \\
e_xK_4 & & & & c_x & & & & d_x \\
e_yK_4 & & & & c_y & & & & d_y \\
e_zK_4 & & & & c_z & & & & d_z \\
f_1 & -f_1 & f_1 & & & & & & \\
f_2K_2 & & & & & -f_2 & -f_3 & & \\
f_3K_3 & & & f_3 & -f_3 & & & & \\
f_4K_5 & & & & & & & f_4 & -f_4
\end{bmatrix}$$

Input/Output

None

Argument List

None

Labeled Common

LPEP, LPMEM, LPMV

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMG

This routine computes vector geometry for lumped parameter membrane parallelogram elements.

Algorithm

Edge Vectors

$\overline{pq} = \overline{q} - \overline{p}$ and $\overline{qp} = -\overline{pq}$

$\overline{rq} = \overline{q} - \overline{r}$ and $\overline{qr} = -\overline{rq}$

$\overline{rs} = \overline{s} - \overline{r}$ and $\overline{sr} = -\overline{rs}$

$\overline{ps} = \overline{s} - \overline{p}$ and $\overline{sp} = -\overline{ps}$

$\overline{a} = \overline{pq} / |\overline{pq}|$

$\overline{b} = \overline{rq} / |\overline{rq}|$

$\overline{c} = \overline{rs} / |\overline{rs}|$

$\overline{d} = \overline{ps} / |\overline{ps}|$

$\overline{e} = \overline{ps} \times \overline{pq} / |\overline{ps} \times \overline{pq}|$

K Factors

$K_1 = -(\overline{a} \times \overline{qr}) \cdot \overline{c} / (\overline{e} \times \overline{qr}) \cdot \overline{c}$

$K_2 = -(\overline{a} \times \overline{ps}) \cdot \overline{e} / (\overline{b} \times \overline{rs}) \cdot \overline{e}$

$K_3 = -K_2 (\overline{b} \times \overline{qp}) \cdot \overline{e} / (\overline{c} \times \overline{sp}) \cdot \overline{e}$

$K_4 = -K_3 (\overline{c} \times \overline{sp}) \cdot \overline{a} / (\overline{e} \times \overline{sp}) \cdot \overline{a}$

$K_5 = -(\overline{a} \cdot \overline{d}) - K_2 (\overline{b} \cdot \overline{d}) - K_3 (\overline{c} \cdot \overline{d})$

Corner Angles

$\theta_p = \cos^{-1} (-\overline{a} \cdot \overline{d})$

$\theta_s = \cos^{-1} (\overline{c} \cdot \overline{d})$

$\theta_q = \cos^{-1} (\overline{a} \cdot \overline{b})$

$$\theta_r = \cos^{-1}\,(-\bar{c} \cdot \bar{b})$$

Volumn and Mass

$$V = \frac{t}{2}\,(\overline{pq} \times \overline{rq} + \overline{rs} \times \overline{ps})$$

$$M = \rho\, V$$

Input/Output

None

Argument List

None

Labeled Common

JØRT, LPEP, LPMV

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPMK

This routine computes the transformed element stiffness matrix for lumped parameter membrane elements.

Algorithm

The following matrix equation is evaluated where k is of order 16 x 16.

$$k = F_{\bar{F}} \bar{k} F_{\bar{F}}^{T}$$

Input/Output

None

Argument List

KLPM A real array for the element k matrix

KBAR A real array for the element stiffness matrix

FFBAR A real array for the unassembled to global transformation matrix

NK An integer scalar defining the number of rows and columns of array KLPM

NKB An integer scalar defining the number of rows and columns of array KBAR

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMKC

This routine forms the lumped parameter membrane stiffness matrix.

Algorithm

The flexibility matrix, d, for the membrane element is first formed by evaluating the matrix equation

$$d = (T_p{}^T \tilde{D}_p T_p + T_q{}^T \tilde{D}_q T_q + T_r{}^T \tilde{D}_r T_r + T_s{}^T \tilde{D}_s T_s)$$

where the actual assembly of matrix d is done element by element knowing the configuration of the T and $\tilde{D}$ matrices.

After the assembly of matrix d, routine SID is called to obtain the inverse which is the element stiffness matrix $\bar{k}$.

Input/Output

Diagnostics are written onto tape 6.

Argument List

FK	A real scalar defining the element
JMN	An integer array containing membrane joint numbers JP, JQ, JR, JS
KBAR	A real array for the LPM stiffness matrix

Labeled Common

LPMDSZ

Subroutines Called

LPMPRT, SID

Error Detection

If SID cannot invert KBAR, a diagnostic is printed, intermediate calculations dumped, and the IERRØR flag set equal one. Return is then made to the calling routine. The loss of significant digits is also checked, and, if less than 2, a warning diagnostic is printed and execution is allowed to continue.

SUBROUTINE LPMM

This routine forms the mass matrix for lumped parameter membrane elements.

Algorithm

The element mass matrix m of order 16 x 16 is assembles as shown below.

$$m = \frac{\rho v}{36} = \begin{bmatrix}
4 & & & 2 & & & 1 & & & 2 & & \\
& 4 & & & 2 & & & 1 & & & 2 & \\
& & 4 & & & 2 & & & 1 & & & 2 \\
2 & & & 4 & & & 2 & & & 1 & & \\
& 2 & & & 4 & & & 2 & & & 1 & \\
& & 2 & & & 4 & & & 2 & & & 1 \\
1 & & & 2 & & & 4 & & & 2 & & \\
& 1 & & & 2 & & & 4 & & & 2 & \\
& & 1 & & & 2 & & & 4 & & & 2 \\
2 & & & 1 & & & 2 & & & 4 & & \\
& 2 & & & 1 & & & 2 & & & 4 & \\
& & 2 & & & 1 & & & 2 & & & 4
\end{bmatrix}$$

Input/Output

None

Argument List

EMASS	A real array in which the mass matrix is to be generated
N	An integer scalar defining the maximum dimensions of the array EMASS
PV	A real scalar defining the quantity ρv

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMPRM

This routine prints the output matrices for lumped parameter membrane elements.

Algorithm

Matrix partitions for membrane elements and the contents of labeled common LPMEM are printed.

Input/Output

Matrix data for membrane elements are printed on tape 6.

Argument List

None

Labeled Common

LPMEM

Subroutines Called

WRMAT

Error Detection

None

SUBROUTINE LPMPRT

This routine prints the input and intermediate calculations for lumped parameter membrane.

Algorithm

Intermediate membrane element calculations and the contents of labeled common LPEP, LPMV and LPMDSZ are printed.

Input/Output

Intermediate membrane element data are printed on tape 6.

Argument List

None

Labeled Common

JØRT, LPEP, LPMDSZ, LPMV

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMSM

This routine saves lumped parameter membrane element data on external files.

Algorithm

The matrix partitions representing the contribution of this element to each of the output matrices are written onto the appropriate scratch tape.

Input/Output

The data output to scratch tapes is as follows:

Tape	Data
7	Element variable table
8	CBAR
9	EBØT
10	EMASS
11	EPSSIG
12	FFBAR
13	KBAR
14	KLPC
15	SFBAR
18	Element constant table

Argument List

NMEM	An integer scalar defining the membrane element number
TMPAVG	A real scalar defining the element average temperature

Labeled Common

LPEP, LPMEM, LPMV, MTRL

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMSZD

This routine computes the area matrices (Sp, Sq, Sr, Ss), the global transformation matrices (Zp, Zq, Zr, Zs), and the D-tilde matrices ($\tilde{D}p$, $\tilde{D}q$, $\tilde{D}r$, $\tilde{D}s$) used to compute the lumped parameter membrane element damping, stiffness, and strain matrices.

Algorithm

The corner angles, θp, θq, θr, and θs, are tested for equality which would indicate the element is a parallelogram. If they are not equal, the following six steps are executed for each of the four element sides using the corresponding edge vector and corner angle.

1) $a = \frac{1}{2}\ |\overline{ps}|$ (constant for all sides)

2) $b = \frac{1}{2}\ |\overline{pq}|$ (substituting $\overline{rq}$, $\overline{rs}$ and $\overline{ps}$)

3) $\frac{A}{t} = a\ b \sin \theta p$ (substituting θq, θr and θs)

4) $Sp = \begin{bmatrix} \frac{1}{b} & & \\ & \frac{1}{a} & \\ & & \frac{1}{a} \end{bmatrix}$

5) Call routine LPMZ to compute matrix Z

6) Call routine LPMD to compute matrix $\tilde{D}$

If the element is a parallelogram, the S, Z and $\tilde{D}$ matrices corresponding to each side of the element are equal. In this case, the S, Z and $\tilde{D}$ matrices are computed for the first side and copied for the other three sides.

After completing the above procedure, if the element is a parallelogram, routine LPMS1 is called to form matrix σ_F. If the element is not a parallelogram, routine LPMS2 generates matrix σ_F.

Input/Output

None

Argument List

CK A real array for the compliance matrix

T A real scalar defining the membrane thickness

ZETAPQ A real scalar defining the material or stress orientation angle

Labeled Common

LPMDSZ, LPMEM, LPMV

Subroutines Called

LPMD, LPMS1, LPMS2, LPMZ

Error Detection

None

SUBROUTINE LPMS1

This routine forms the lumped parameter membrane stress matrix for parallelogram elements.

Algorithm

The stress matrix of order 3 x 9 for a parallelogram membrane element is given by the matrix equation

$$\sigma_{\overline{F}} = \frac{ZS}{2t} (T_p + T_q + T_r + T_s)$$

or, more explicitly, in terms of the elements of matrix Z, the vector magnitudes a and b, the thickness t, and the element K factor is given by

$$\sigma_{\overline{F}} = \frac{1}{2t} \begin{bmatrix} c_1 & \frac{Z_{11}}{b} & \frac{Z_{11}}{b} & \frac{Z_{11}}{b} & \frac{Z_{11}}{b} & \frac{Z_{12}}{a} & \frac{Z_{12}}{a} & \frac{Z_{12}}{a} & \frac{Z_{12}}{a} \\ c_2 & \frac{Z_{21}}{b} & \frac{Z_{21}}{b} & \frac{Z_{21}}{b} & \frac{Z_{21}}{b} & \frac{Z_{22}}{a} & \frac{Z_{22}}{a} & \frac{Z_{22}}{a} & \frac{Z_{22}}{a} \\ c_3 & \frac{Z_{31}}{b} & \frac{Z_{31}}{b} & \frac{Z_{31}}{b} & \frac{Z_{31}}{b} & \frac{Z_{32}}{a} & \frac{Z_{32}}{a} & \frac{Z_{32}}{a} & \frac{Z_{32}}{a} \end{bmatrix}$$

where

$$c_1 = Z_{13} (1 + K)/a$$

$$c_2 = Z_{23} (1 + K)/a$$

$$c_3 = Z_{33} (1 + K)/a$$

Input/Output

None

Argument List

FK	A real scalar defining the K factor

SFBAR A real array for the element stress matrix

THICK A real scalar defining the membrane thickness

Labeled Common

LPMDSZ

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMS2

This routine forms the lumped parameter membrane stress matrix for approximate parallelogram elements.

Algorithm

The stress matrix of order 3 x 9 for an approximate parallelogram element is given by the matrix equation

$$\sigma_{\underline{F}} = \frac{1}{4t}\left(Z_pS_pT_p + Z_qS_qT_q + Z_rS_rT_r + Z_sS_sT_s\right)$$

or, more explicitly, in terms of the elements of matrix Z for each edge, the vector magnitudes a and b for each edge, the thickness t, and the element K factor is given by:

$$\sigma_{\underline{F}} = \frac{1}{4t}\begin{bmatrix} c_1 & \frac{Z_{q11}}{b_q} & \frac{Z_{p11}}{b_p} & \frac{Z_{r11}}{b_r} & \frac{Z_{s11}}{b_s} & \frac{Z_{q12}}{a_q} & \frac{Z_{r12}}{a_r} & \frac{Z_{p12}}{a_p} & \frac{Z_{s12}}{a_s} \\ c_2 & \frac{Z_{q21}}{b_q} & \frac{Z_{p21}}{b_p} & \frac{Z_{r21}}{b_r} & \frac{Z_{s21}}{b_s} & \frac{Z_{q22}}{a_q} & \frac{Z_{r22}}{a_r} & \frac{Z_{p22}}{a_p} & \frac{Z_{s22}}{a_s} \\ c_3 & \frac{Z_{q31}}{b_q} & \frac{Z_{p31}}{b_p} & \frac{Z_{r31}}{b_r} & \frac{Z_{s31}}{b_s} & \frac{Z_{q32}}{a_q} & \frac{Z_{r32}}{a_r} & \frac{Z_{p32}}{a_p} & \frac{Z_{s32}}{a_s} \end{bmatrix}$$

where

$$c_1 = \frac{1}{2}\left[\frac{Z_{p13}}{a_p} + \frac{Z_{q13}}{a_q} + K\left(\frac{Z_{r13}}{a_r} + \frac{Z_{s13}}{a_s}\right)\right]$$

$$c_2 = \frac{1}{2}\left[\frac{Z_{p23}}{a_p} + \frac{Z_{q23}}{a_q} + K\left(\frac{Z_{r23}}{a_r} + \frac{Z_{s23}}{a_s}\right)\right]$$

$$c_3 = \frac{1}{2}\left[\frac{Z_{p33}}{a_p} + \frac{Z_{q33}}{a_q} + K\left(\frac{Z_{r33}}{a_r} + \frac{Z_{s33}}{a_s}\right)\right]$$

Input/Output

None

Argument List

FK A real scalar defining the K factor

SFBAR A real array for the element stress matrix

THICK A real scalar defining the membrane thickness

Labeled Common

LPMDSZ

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMZ

This routine forms the lumped parameter membrane global translation matrix.

Algorithm

Matrix Z of order 3 x 3 is formed as shown below:

$$Z = \frac{1}{\sin(\mathrm{THETA})} \begin{bmatrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \\ z_{31} & z_{32} & z_{33} \end{bmatrix}$$

where

$$z_{11} = \cos^2(\mathrm{ZETA})$$

$$z_{21} = \sin^2(\mathrm{ZETA})$$

$$z_{31} = -\sin(\mathrm{ZETA})\ \cos(\mathrm{ZETA})$$

$$z_{12} = \cos^2(\mathrm{THETA-ZETA})$$

$$z_{22} = \sin^2(\mathrm{THETA-ZETA})$$

$$z_{32} = \sin(\mathrm{THETA-ZETA})\ \cos(\mathrm{THETA-ZETA})$$

$$z_{13} = 2\ \cos(\mathrm{THETA-ZETA})\ \cos(\mathrm{ZETA})$$

$$z_{23} = -2\ \sin(\mathrm{THETA-ZETA})\ \sin(\mathrm{ZETA})$$

$$z_{33} = \sin(\mathrm{THETA-ZETA})\cos(\mathrm{ZETA})-\cos(\mathrm{THETA-ZETA})\sin(\mathrm{ZETA})$$

Input/Output

None

Argument List

THETA	A real scalar defining the corner angle

Z A real array for the global translation matrix

ZETA A real scalar defining the material orientation angle

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMZM

This routine initializes the matrix arrays for lumped parameter membrane elements.

Algorithm

All arrays in labeled common block array LPMEM are set to zero.

Input/Output

None

Argument List

None

Labeled Common

LPMEM

Subroutines Called

None

Error Detection

None

SUBROUTINE LTRMLT

This routine performs a transpose multiply matrix operation.

Algorithm

The following matrix operation is performed:

$$C = A^T B$$

Input/Output

None

Argument List

A	A real array for the matrix to be transposed
M	An integer scalar defining the number of rows of A
N	An integer scalar defining the number of columns of A
MAXRA	An integer scalar defining the max rows of A
MAXCA	An integer scalar defining the max columns of A
B	A real array for matrix B
I	An integer scalar defining the number of columns of B
C	A real array for the product matrix C

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MATDES

This routine processes the material description input data records for a particular material in the material property tables, Data Code 10.

Algorithm

The first card of the material description has been read by routine MTLMØD and is stored in array REC. The routine transfers information from REC to array MRDES until a full word of blanks is encountered. The routine reads, checks, processes and echos the input for continuation cards of the material description. When the description has been completely processed, the pointers pertaining to the beginning and ending of the description words stored in array MRDES are set.

Input/Output

Material descriptions are read from tape 5 and diagnostics are written on tape 6.

Argument List

REC	An alphanumeric array containing the input data record
CØNT	An alphanumeric scalar containing the continuation field of input record, any non-blank character will indicate a continuation

Labeled Common

IDEN, IERRØR, LIMITS, MRTL

Subroutines Called

None

Error Detection

Checks are made for material reference numbers being out of sequence, for invalid Data Code numbers, and for insufficient space allocated for input. If an error is detected, IERRØR is set equal one, a diagnostic is printed, and a return is made to the calling routine.

SUBROUTINE MATHD

This routine writes a FØRMAT matrix header record.

Algorithm

A record, twelve words in length, is written on tape TAPENØ as shown below.

Word	Data	Type	Description
1	-1	integer	matrix header code
2	0	integer	compression code
3	9	integer	words remaining in record
4	X.........	BCD	matrix name (1 leading character with blank fill)
5	X.........	BCD	
6	X.........	BCD	
7	X.........	BCD	
8	X.........	BCD	
9	X.........	BCD	
10	MF	integer	matrix subscript
11	I	integer	number of rows
12	J	integer	number of columns

Input/Output

A matrix header record is written on tape TAPENØ.

Argument List

TAPENØ	An integer scalar defining the logical tape number
NAME	An alphanumeric array for the matrix name (6 characters)
MF	An integer scalar defining the matrix subscript
I	An integer scalar defining the number of rows in the matrix
J	An integer scalar defining the number of columns in the matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MATTR

This routine writes a FØRMAT matrix trailer record.

Algorithm

A record, four words in length, is written on tape TAPENØ as shown below.

Word	Data	Type	Description
1	-2	integer	matrix trailer code
2	0	integer	compression code
3	1	integer	words remaining in record
4	0	integer	dummy data

Input/Output

A matrix trailer record is written on tape TAPENØ.

Argument List

TAPENØ	An integer scalar defining the logical tape number

Labeled Common

None

Subroutines Called

None

Error Detected

None

SUBROUTINE MLTMTR

This routine forms the product of two matrices.

Algorithm

(C) = (A) (B)

Input/Output

None

Argument List

A	A real array for the first input matrix
M	An integer scalar defining the usable rows in matrix A
N	An integer scalar defining the usable columns in matrix A
MAXR	An integer scalar defining the dimensioned row size of matrix A
MAXC	An integer scalar defining the dimensioned row size of matrix B
B	a real array for the second input matrix
I	A integer scalar defining the usable columns of matrix B
C	A real array for the output matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MLTMØD

This is the executive routine for the module which reads and processes the material property table, Data Code 10.

Algorithm

First the lengths of the coefficient table and the materials description table are initialized.

A material property data card is read, checked for errors, and an echo of the data is printed. If the card is descriptive text for the material, subroutine MATDES is called to process it and any continuation cards. If the card defines a property of the material, subroutine PRØP is called to process the input. This procedure is repeated until all input for all materials is processed.

The necessary constants are saved in labeled common CONST and the assembled material property data is written to tape 1 in the form of a standard FØRMAT matrix. Finally, a report of the output coefficients is printed on tape 6.

Input/Output

The material property table is input on tape 5. Material property coefficients and diagnostics are output on tape 6. The material property data also output on tape 1 for use by the analysis modules following the initial generator.

Argument List

None

Labeled Common

CØNST, IDEN, IERRØR, LIMITS, MTRL

Subroutines Called

MATDES, PRØP

Error Detection

Checks are made for omissions in the input for a material, for a material reference number out of range for which space has been allocated, for invalid Data Code, and for material reference numbers out of sequence. If an error is detected, appropriate diagnostics are printed, the IERRØR flag is set equal one, and a return is made to the calling routine.

SUBROUTINE ØVER3

This routine searches the constraint table for more than three entries per joint.

Algorithm

It is required that the system of constraints at a joint for translation be statically determinate. Therefore, for translation, the table is searched to determine if there are more than three input constraints acting at a joint.

Input/Output

Diagnostics are written onto tape 6.

Argument List

ICØN An integer array for the constraint table

NCT An integer scalar defining the number of records in the constraint table

JERRS An integer scalar error condition flag for the constraint table

Labeled Common

IERRØR

Subroutines Called

None

Error Detection

If more than three constraints are found at a joint, a diagnostic is printed, the IERRØR is set equal one, and a return is made to the calling routine.

SUBROUTINE PARTN

This routine partitions a file which is being sorted by routine QKSØRT.

Algorithm

Pointers are set to the top of the file and to the bottom of the file. A pointer is set to the pivot record, that is, the record that is to serve as the partitioning element. The purpose is to end up with all the larger records above the pivot element and all the smaller records below. The number of records in the file is calculated, then a comparison is made between the pivot record (which is initially at the bottom) and the highest record. If the pivot record is smaller, then the pointer to the highest record is moved down to the next record. If the pivot is not smaller, then a swap is made between the pivot record and the compared record. The pointer to the lowest record is moved up and comparisons begin between the pivot record and the records below it. The file is partitioned when the upper and lower pointers are adjacent.

Input/Output

Error messages, if any, are written to file 6.

Argument List

REC	An integer array of records to be sorted
CHGSEQ	An integer array to specify the original positions of the sorted records
NREC	An integer scalar specifying the number of records in array REC
LENREC	An integer scalar specifying the length of the records in array REC

Labeled Common

ABC

Subroutines Called

SWAP

Error Detection

A check is made at the end of the partitioning to be sure that the file pointers are correct, that is, the ending low pointer should be immediately after the ending high pointer. If this is not true, then the low, high and pivot pointers are printed along with a diagnostic and the records of the file passed to PARTN for partitioning. Execution is then terminated.

SUBROUTINE PASSM

This is the driver routine for the module which assembles output matrices PRUPT, PRUF and ECT.

Algorithm

After initializing the intermediate error flag, NERR, and rewinding files 16, 17 and 18, routines PASSM1, PASSM2, PASSM4, and PASSM5 are called to assemble matrices ECT, PRUPT and PRUF. File 16 contains constraint description data, file 18 contains element joint numbers as well as other element constants, and file 17 is subsequently used as scratch.

Input/Output

None

Argument List

None

Labeled Common

CØNST, EDØF, IEDGE, IERRØR, ISEQ

Subroutines Called

PASSM1, PASSM2, PASSM4, PASSM5

Error Detection

If NERR is returned as non-zero by PASSM2, then IERROR is set equal one and a return is made to the calling routine.

SUBROUTINE PASSM1

This routine assembles and outputs matrix ECT and assembles intermediate matrix PFT.

Algorithm

After writing matrix headers on files 17 and 1 for matrices PTF and ECT, the bar data is processed followed by membranes and cells. The same procedure takes place in each case.

The element constants for the next element are read from file 18 into array IECT. Using the edge degree of freedom numbers stored in array IDØF, array IECT is augmented and output to file 1. Array NPF is then set to the degree of freedom numbers corresponding to each of the element forces and output to file 17.

In the case of point mass elements, no data is output to file 1 for matrix ECT. However, array NPF is assembled and output to file 17 in a similar manner as the other elements.

Input/Output

Matrix ECT is output to tape 1. Tape 17 is used as a scratch unit to hold the PTF matrix temporarily. ECT data from the routines LPBSM, LPCSM and LPMSM is input on tape 18. Tape 6 is used to output intermediate calculations if flagged.

Argument List

IEDGE An integer array of joint pairs defining each edge DØF

IDØF An integer array relating each edge element force to an edge DØF

IECT An integer array of element constants

NPF An integer array used to store a column of PFT

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHD, MATTR

Error Detection

None

SUBROUTINE PASSM2

This routine assembles intermediate matrix PUPT.

Algorithm

After writing a matrix header on file 18 for matrix PUPT, constraint data is read from file 16 into arrays ICØN and ØBLIQ. The three columns of ICØN contain the constraint number, the joint at which the constraint acts, and the direction description. Values of 1, 2 and 3 for direction description indicate the global X, Y and Z directions, respectively. Values greater than 3 are pointers to array ØBLIQ which contains the direction cosines of oblique constraints.

Using arrays ICØN and ØBLIQ, matrix PUPT is assembled by omitting rows corresponding to global degrees of freedom which are constrained and placing unity in remaining rows at the corresponding column. In the case of oblique constraints, routine PASSM3 is called to transform global X, Y and Z unit vectors into contributions in the global and/or oblique unconstrained degrees of freedom.

Array IDØF is used to flag constrained degrees of freedom as zero, unconstrained degrees of freedom as one, and degrees of freedom with duplicate constraints as less than zero.

Input/Output

Constraint data is input from tape 16. Matrix PUPT is output to tape 18. Tape 6 is used for printing diagnostics and intermediate calculations if flaged.

Argument List

IEDGE An integer array of joint pairs defining each edge DØF

IDØF An integer array used to flag constrained TDØF and to count and detect duplicate global constraints

ICØN An integer array defining each constraint created by the constraint processor module and written to unit 16

ØBLIQ A real array of direction cosines of oblique constraints also on unit 16

NERR An integer scalar used as an error flag

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHD, MATTR, PASSM3

Error Detection

A test is made for duplicate constraints, and, if detected, the NERR flag is set equal one and a return is made to the calling routine. A diagnostic is printed on tape 6.

SUBROUTINE PASSM3

This routine assembles contributions to intermediate matrix PUPT for joints which have oblique reactions.

Algorithm

The equation $[A][B]^{-1} = [C]$ is solved to obtain the contributions to matrix PUPT for a joint with oblique constraints. [A] is an identity of order 3 and [B] contains the direction cosines of from 1 to 3 oblique constraints. Both [A] and [B] are stored in array PF. The code initializes the leading 3 x 3 partition of array PF as an identity.

After solving for [C], the rows corresponding to unconstrained degrees are assembled from each column in arrays P and IP in compressed form. Each column of matrix PUPT so obtained is then output to file 18.

Input/Output

Columns of matrix PUPT are output to tape 18. Tape 6 is used for printing intermediate calculations if flaged.

Argument List

IRF — An integer array which is the partition of array IDØF for the joint being processed

PF — A real array where columns 1 to 3 contain unit vectors in the X, Y, Z global directions and the remaining columns contain the components of reactions acting at this joint

NR — An integer scalar defining the number of reactions at this joint

NERR — An integer scalar used as an error flag

ICØL — An integer scalar defining the column location in PUPT of the last matrix element output

IRØW — An integer scalar defining the row location in PUPT of the last matrix element output

NPASSM An integer scalar used to signal printing of intermediate data

Labeled Common

None

Subroutines Called

None

Error Detection

A test is made for duplicate constraints, and, if detected, the NERR flag is set equal one and a return is made to the calling routine.

SUBROUTINE PASSM4

This routine determines the unconstrained degree of freedom (UDØF) row format, the reordered UDØF for SEQWF, prints these row formats, and outputs intermediate matrix PRP.

Algorithm

Initially, array IEDGE contains pairs of joint numbers identifying each edge degree of freedom. Two joint numbers are stored in a single word by scaling the first joint number by 10000 and adding the second joint number.

Arrays IEDGE and ISEQ are restructured to contain the identification of each unconstrained degree of freedom. The identification for joint degrees of freedom consists of a joint number in array IEDGE and the integer 1, 2 or 3 in array ISEQ indicating global X, Y or Z degree of freedom, respectively. The identification for an edge degree of freedom consists of the first joint number in array IEDGE and the negative of the second joint number in array ISEQ.

The joint number pairs initially contained in array IEDGE are extracted and stored in arrays IEDGE and ISEQ at locations corresponding to their proper positions in the unconstrained degree of freedom row format. Then using the array IDØF containing flags identifying constrained and unconstrained joint degrees of freedom, the partition of arrays IEDGE and ISEQ are set to appropriate values.

Arrays IEDGE and ISEQ are then scanned to determine the reordered unconstrained degree of freedom format. The reordered position of entries in arrays IEDGE and ISEQ is stored in array IDØF.

Input/Output

Reports of the unconstrained degrees of freedom and the reordered UDØF row formats are printed on tape 6.

Argument List

IEDGE	An integer array of joint pairs defining each edge DØF
IDØF	An integer array of flags identifying joint constrained and unconstrained DØF
ISEQ	An integer array used for intermediate storage

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHD, MATTR, PASSM3

Error Detection

None

SUBROUTINE PASSM5

This routine assembles and outputs matrices PRUF and PRUPT from data stored in array IDØF and intermediate matrices PUPT and PTF which are on tape.

Algorithm

Array IEDGE is initialized as zero for a length equal to the total number of degrees of freedom. Matrix PUPT is then read from file 18 one column at a time. Using the reordered row positions stored in array IDØF, the reordered elements of matrix PRUPT are stored in arrays IEDGE and ISEQ. Array IEDGE contains a positive or negative integer at the row location corresponding to the column of matrix PUPT read from file 18. A positive integer indicates a single value of unity for that column of matrix PRUPT at the row position equal to the integer value. A negative integer indicates more than one value is present in the column. This occurs only at joints where oblique constraints are present. In this case, the absolute value of the integer points to a location in array ISEQ containing the number of values present in the column. Immediately following in array ISEQ, are pairs of values and reordered row locations of that column.

After sorting the value/location pairs for a column in array ISEQ, the column of matrix PRUPT is output to file 1 using the data contained in arrays IEDGE and ISEQ.

The coefficients in arrays ISEQ are then made negative for use in conjunction with the assembly of matrix PRUF. Matrix PFT is then read from file 17 one column at a time. Each column of matrix PTF contains the total degree of freedom number for each force for an element. Using these numbers as pointers into array IEDGE, the columns of matrix PRUF are assembled and output to file 1.

Input/Output

Matrices PRUPT and PRUF are output onto tape 1. Matrix PTF is input from tape 17 and matrix PUPT is input from tape 18. Intermediate calculations are printed on tape 6 if flaged.

Argument List

IEDGE An integer array containing the reordered row positions of each UDØF

IDØF An integer array used to store reordered row positions of each TDØF

ISEQ An integer array used to store values and positions of elements of matrix PUPT when a column contains other than a single value of unity

IPFT An integer array used to store columns of matrix PRP

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHHD, MATTR, PASSM6

Error Detection

None

SUBROUTINE PASSM6

This routine sorts array ISEQ columnwise into ascending order based on the values in the second row of array ISEQ.

Algorithm

Array ISEQ contains matrix element values in the first row and matrix element row locations in the second row. The columns are sorted based on the values in the second row. Note that the number of columns in array ISEQ must be two or more.

Input/Output

None

Argument List

ISEQ	An integer array used to store values and positions of elements of matrix PUPT
KNUM	An integer scalar defining the number of columns in ISEQ

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE PRØP

This routine processes the material property value input data records for a particular material in the material property tables, Data Code 10.

Algorithm

The first material property card has been read by routine MTLMØD and is stored in array REC. The parameters in labeled common IDEN have also been initialized.

The next material property card is read, checked for correctness of input and to determine if it is a continuation card associated with the property. If coefficients are input, they are saved in array AP. If temperature-value pairs are input, the temperature-value arrays are set up to compute the coefficients. The coefficients are then computed by subroutine CØCALC.

The coefficients for this property are moved from array AP to array CØEFS, the pointers and counters updated, and the data for the next property is read and processed similarly.

Input/Output

Material property data is read from tape 5. Diagnostics are printed on tape 6.

Argument List

REC	A real array containing the current input data record

Labeled Common

CØFCLC, IDEN, IERRØR, LIMITS, MTRL

Subroutines Called

CØCALC

Error Detection

Checks are made to insure that there is sufficient space allocated for the materials input, that the number of coefficients read is equal to the number requested, that the number of coefficients to be calculated is not greater than the number of values input, that the number of coefficients will not overflow the space allocated for all coefficients, that there are not more properties to be input than space is allocated for, that there is not more coefficients to be input than space is allocated for, that a wrong material reference number was input, or that a bad Data Code was input.

SUBROUTINE PRTCØN

This routine prints the contents of labeled common block CONST.

Algorithm

The contents of labeled common CØNST, consisting of model size data and problem constants, is printed.

Input/Output

Problem constants and sizing variables are printed on tape 6.

Labeled Common

CØNST, IERRØR

Subroutines Called

None

Error Detection

None

SUBROUTINE QKSØRT

This is the driver routine for the module that accomplishes file sorting by the infile Quick Sort technique (Reference 17).

Algorithm

The upper and lower records of the target file are identified. The file is partitioned into 2 subfiles relative to the key field of a specified record. The subfiles are identified by a stack of pointers. Subfiles are removed from the stack and partitioned until there are no subfiles remaining. If a subfile consists of only 2 records, then it is ordered and need not be stacked with the other subfiles. When no subfiles remain, then the file is sorted.

Input/Output

None

Argument List

REC An integer array of records to be sorted

CHGSEQ An integer array to specify the original positions of the sorted records

NREC An integer scalar specifying the number of records in array REC

LENREC An integer scalar specifying the length of each record in array REC

ITØP An integer scalar pointing to the last record in array REC

IBØT An integer scalar pointing to the first record in array REC

KYFLD An integer scalar pointing to the record element containing the sort key

Labeled Common

ABC

Subroutines Called

PARTN, STACK, SWAP

Error Detection

None

SUBROUTINE RAMØSG

This routine calculates the Ramberg-Osgood coefficient, the slope tangent to the Ramberg-Osgood curve at the origin, and the elastic limit stress.

Algorithm

First, the input properties are tested to determine if the material has a plastic range. If $E_A = E$, $\bar{\sigma}_A = \bar{\sigma}_r$ and $\bar{\epsilon}_r = \frac{\bar{\sigma}r}{E}$, the material is assumed to be totally elastic. The program sets $n = 1$, $\bar{E} = E$, and $\bar{\sigma}_L = \bar{\sigma}_r$ and returns to the calling routine.

If the material has a plastic range, the Ramberg-Osgood coefficient, n, the initial slope of the stress/strain curve, $\bar{E}$, and the elastic limit stress, $\bar{\sigma}_L$, are solved for using an iterative approach. Initializing

$$r = \frac{E_A}{E}$$

$$\text{and} \qquad n_0 = 50.0$$

the following equations are evaluated in an iterative manner until n has converged or the iterative count, k, reaches 50.

$$\text{CALC1} = 1.0 - r^{n_{k-1}-1}$$

$$R_{k-1} = \frac{\frac{E_A}{E} - r^{n_{k-1}-1}}{\text{CALC1}}$$

$$\text{CALC2} = \epsilon_r - R_{k-1}\frac{\bar{\sigma}_r}{E_A}$$

$$\text{CALC4} = \frac{\bar{\sigma}_A}{E_A}$$

$$\text{CALC5} = \frac{\bar{\sigma}_r}{\sigma_A}$$

$$n_k = \frac{\ln\,(CALC2) - \ln\,(1-R_{k-1}) - \ln\,(CALC4)}{\ln\,(CALC5)}$$

When n has converged, processing is completed by evaluating the following equations:

$$R = \frac{\frac{E_A}{\overline{E}} - r^{n-1}}{1 - r^{n-1}}$$

$$\overline{E} = \frac{E_A}{R}$$

$$\sigma_L = r\,\overline{\sigma}_A$$

If convergence does not take place, a fatal error condition occurs causing subsequent failure of the run.

Input/Output

Diagnostics are written on tape 6.

Argument List

MR	An integer scalar defining the material reference number

Labeled Common

IERRØR, MTRL

Subroutines Called

None

Error Detection

Checks are made to detect various denominators being equal to zero, non-convergence, and an attempt to extract the logorithm of zero. The errors associated with each of the potential errors codes printed in the diagnostic message are listed below:

Error Code	Error
1	$E = 0.0$
2	$E_A = 0.0$
3	$\bar{\sigma}_A = 0.0$
4	no convergence
5	CALC1 = 0.0
6	$R = 1.0$
7	CALC2 = 0.2
9	CALC4 = 0.0
10	CALC5 = 0.0

SUBROUTINE SID

This routine inverts a real non-singular square matrix (A), and if desired, will also solve linear system(s) of simultaneous equations, (A)(X) = (B) where (B) may have any number of columns. Every column of (B) must have at least one non-zero element.

Algorithm

The numerical method is basically the Gauss-Jordan elimination with selection of maximum pivotal elements (full pivoting). Special procedures are present to improve accuracy and also minimize the frequency of overflow and underflow when the magnitudes of any of the elements in the (A) or (B) matrices are large or small.

Input/Output

None

Argument List

A — A two-dimensional real array which contains (A). (A) will be replaced by inverse (A) during the execution of SID.

N — An integer which denotes the number of rows in (A)

NDRØW — An integer scalar which denotes the maximum number rows which may be stored in the A array. Note that the matrix (A) may have fewer rows and/or columns than the array A which contains it.

NDCØLA — An integer scalar which denotes the maximum number of columns which may be stored in the A array

B — A real array which contains (B). If (B) is present, the B array must have exactly NDRØW rows. (B) will be replaced by (X) during the execution of SID. If no (B) is present, B may be any variable or constant of any type.

M — An integer scalar which denotes the number of columns in (B). If there is no (B), use M = 0.

NDCØLB An integer which denotes the maximum number of columns which may be stored in the B array. If no (B) is present, NDCØLB may have any value.

SIGDIG A real scalar which will be set equal to an estimate of the number of significant digits in the elements in the inverse. This estimate is based on the assumption that the elements of (A) are all accurate to eight significant digits.

IERRØR An integer scalar used to flag errors. IERRØR = -1 indicates both of the following:
1) neither $(A)^{-1}$ nor (X) could be computed
2) (A) was singular or nearly so.
IERRØR = 1 indicates no errors were detected. Division by zero cannot occur in this routine. No test is made for floating point overflow.

PIVØT A real array containing at lease 3*N elements

INDEX An integer array containing at least 3*N elements

SCALEB A real array containing at least M elements

Labeled Common

None

Subroutines Called

None

Error Detection

See IERRØR and SIGDIG under Argument List above.

SUBROUTINE SKDATA

This routine skips over Data Codes in the card input tables which are not used by the initial generator.

Algorithm

The input file is read and each card is checked to determine if it is the end card for the Data Code to be skipped over. When the end card is encountered, a return is made to the calling routine. If a Data Code is encountered which is less than the Data Code to be skipped over, an error exit occurs. If the Data Code is larger, the input pointer is backed up and a normal exit is made.

Input/Output

Input is read from tape 5 and error diagnostics are written on tape 6.

Argument List

ICØDE	An integer scalar defining the Data Code number to be read over

Labeled Common

None

Subroutines Called

None

Error Detection

If the data cards are out of sequence, the IERRØR flag is set equal one and control is returned to the calling routine.

SUBROUTINE STACK

This subroutine stacks pointers to subfiles of a file that is being sorted by routine QKSØRT.

Algorithm

The lengths of the lower and upper subfiles are calculated. The largest subfile is stacked first. If they are of equal length, then the upper file is stacked first. Subfiles of 2 records are not stacked.

Input/Output

None

Argument List

REC	An integer array of records to be sorted
NREC	An integer scalar specifying the number of records in array REC
LENREC	An integer scalar specifying the length of the records in array REC

Labeled Common

ABC

Subroutines Called

None

Error Detection

None

SUBROUTINE SWAP

This routine switches the positions of two records in a table that is being sorted by routine QKSØRT.

Algorithm

Two rows of length LENREC in array REC are interchanged as are two entrys in array CHGSEQ. The counter ISWAP is incremented for each interchange made in array REC.

Input/Output

None

Argument List

I An integer scalar defining the first row of array REC

J An integer scalar defining the second row of array REC

CHGSEQ An integer array of original row positions of array REC

NREC An integer scalar defining the number of rows in array REC

LENREC An integer scalar defining the number of columns in array REC

REC An integer array for the table to be sorted

Labeled Common

ABC

Subroutines Called

None

Error Detection

None

SUBROUTINE TAPEHD

This routine writes a FORMAT tape header record.

Algorithm

A record ten words in length is written on tape TAPENØ as shown below.

Word	Data	Type	Description
1	-10	integer	tape header code
2	0	integer	compression code
3	7	integer	words remaining in record
4	X	BCD	tape name (1 leading character with blank fill)
5	X	BCD	
6	X	BCD	
7	X	BCD	
8	X	BCD	
9	X	BCD	
10	MOD	integer	tape modifier

Input/Output

A tape header is written on tape TAPENØ.

Argument List

TAPENØ	An integer scalar defining the logical tape number
NAME	An alphanumeric array for the tape (6 characters)
MØD	An integer scalar defining the tape modifier

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE TAPETR

This routine writes a FØRMAT tape trailer record.

Algorithm

A record of four words in length is written on tape TAPENØ as shown below.

Word	Data	Type	Description
1	-20	integer	tape trailer code
2	0	integer	compression code
3	1	integer	words remaining in record
4	0	integer	dummy data

Input/Output

Writes a tape trailer on tape TAPENØ.

Argument List

TAPENØ	An integer scalar defining the logical tape number

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE VECT

This routine performs vector operations.

Algorithm

The following vector operation is performed according to the value of N.

N	Operation
1	$\overline{c} = \overline{B} - \overline{A}$
2	$\overline{c} = \overline{B} \times \overline{A}$
3	$\overline{c} = \overline{B} \cdot \overline{A}$
4	$\overline{c} = \overline{B} + \overline{A}$

The magnitude of the resultant vector $\overline{c}$ is then computed and stored in T. If normalization is requested, the components of the resultant vector $\overline{c}$ are divided by T for N equal to 1, 2, or 4.

Input/Output

None

Argument List

N	An integer scalar specifying which vector operation to perform
A	A real array for the first impact vector
B	A real array for the second impact vector
C	A real array for the resultant vector
T	A real scalar for the resultant vector magnitude
I	An integer scalar controlling normalization of the resultant vector

Labeled Common

None

AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3
AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)
DEC 77 R C MORRIS F33615-75-C-3105
UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

4 OF 6
ADA
063741

Subroutines Called

None

Error Detection

None

SUBROUTINE WPART

This routine writes a partition of a matrix onto tape in standard FØRMAT matrix data format.

Algorithm

Each column of the matrix partition A is written onto tape in the compressed mode. The origin of matrix partition A in the overall output matrix is at row IRØW and column ICØL. The row/column indices of the element of array A are incremented by IRØW and ICØL to obtain their proper location in the overall matrix.

Inout/Output

A matrix partition is written onto tape TAPENØ.

Argument List

TAPENØ An integer scalar defining the number of the last column output logical tape number

IRØW An integer scalar defining the number of last row output

ICØL An integer scalar defining the number of the last column output

A A real array for the core resident matrix partition A

M An integer scalar defining the number of rows of matrix partition A

N An integer scalar defining the number of columns of matrix partition A

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WRMAT

This routine prints a matrix residing in the core.

Algorithm

The maximum column dimension of the matrix to be printed is 50. If MAP is equal to zero, the elements of the matrix are printed. If MAP is not zero, a map of the non-zero elements of the matrix is printed. This routine is executed only in runs made for program checkout.

Input/Output

Matrix data is printed on tape 6.

Argument List

AMAT	A real array for the matrix to be printed
M	An integer scalar defining the number of rows in AMAT
N	An integer scalar defining the number of columns of AMAT
MAP	An integer scalar print control flag
LABEL	Matrix label
L	An integer scalar defining the number of words (10 char/word) in LABEL

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WTAPE1

This routine reads the element partitions for matrices KEL, MEL, KBAR, CBAR, FFBAR, SIGFB, EPSIG, DEBT, EVT, and CØNST, assembles the partitions into the final matrices, and outputs the matrices as standard FØRMAT matrix data.

Algorithm

The order in which the first nine output matrices are processed is KEL, MEL, KBAR, CBAR, FFBAR, SIGFB, DPSIG, DEBT, and EVT. In each case, the element data is read and processed in the order bars, membranes, and cells. The only matrix having contributions from point mass elements, which are processed last, is matrix MEL. However, trailing null partitions are also present in matrix KEL for point mass elements.

In all of these matrices with the exception of DEBT and EVT, the element partitions are positioned as diagonal or psuedo diagonal partitions. In matrices DEBT and EVT, each element partition is a single column each beginning in the first row.

Finally, array CØNST which contains problem constants is output as a single column matrix.

Input/Output

Matrix data is read from tapes 7 through 15 and all output data is written to tape 1.

Tape	Matrix
14	KEL
10	MEL
13	KBAR
8	CBAR
12	FFBAR

Tape	Matrix
15	SIGFB
11	EPSIG
9	DEBT
7	EVT

Argument List

None

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHD, MATTR, WPART

Error Detection

None

APPENDIX C

LOADS GENERATOR ROUTINES

APPENDIX C
LOADS GENERATOR ROUTINES

This appendix contains detailed descriptions of all routines in this program. Table C gives either page number references within this document or references to other documents for documentation of each routine. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

TABLE C. INDEX TO LOADS GENERATOR ROUTINES

MAIN PRØGRAM LØDGEN

The main program allocates core and calls subroutines to read, calculate, and output incremental loads data.

Algorithm

The subroutines READCØ and READCN are called and the average impact force is calculated. The load increment loop is then entered and READB and READJ are called. The current total load on the model and the location of the center of its footprint for this load increment are calculated, the largest component of the centroidal vector is determined, and the subroutines GENAB and MATMUL are called. If the load increment number is greater than one and less than or equal to the total number of load increments, subroutine DELTA is called. If the increment number is one, there is no need to calculate the change in loads from the preceding increment to this, since this change is the current load, and if the load increment number is equal to the number of increments plus one, the change in load is the negative of the preceding load. Subroutine ØUTPUT is then called and the current loads and the joints of their applications are stored at the end of the A array for use in the next load increment.

Input/Output

There is no input; all output is printed on file 6.

Argument List

None

Labeled Common

None

Subroutines Called

CRØSS, DØT, MATMUL, READB, READCØ, UVEC, DELTA, GENAB, ØUTPUT, READCN, READJ

Error Detection

If, in either of the checks on utilization of the A array, it is found that the available storage in the A array has been exceeded, the message
'INSUFFICIENT STORAGE nnnnn VS nnnnn'
is printed and the program is terminated.

SUBROUTINE DELTA

This routine compares the joint loads of the current load increment to the joint loads of the previous load increment and finds the change in load at each joint.

Algorithm

Since only non-zero joint loads for the current and previous load increments are stored in order of the joint numbers, there are three basic possibilities for each change in joint load. (1) The joint is loaded in the current increment, but not in the previous increment; (2) the joint is loaded in both the current and previous increments; or (3) the joint was loaded in the previous increment but not in the current increment. In any case the change in load on the joints loaded currently or in the previous load increments, are found and stored in order of the numbers of the joints.

Input/Out

None

Argument List

JØINTØ An integer array of the numbers of joints that received load in the previous load increment

PHIØLD A real array of loads on the joints in the previous load increment

JØINTN An integer array of the numbers of joints that receive load in this increment

PHINEW A real array of the loads on the joints in the current load increment

JTDEL An integer array of the numbers of joints that were loaded in the previous load increment or are loaded in the current increment

DELPHI A real array of the changes in joint loads from the previous load increment

NJTØ	An integer scalar defining the number of joints that received load in the previous load increment
NJTN	An integer scalar defining the number of joints that receive load in the current increment
NDELPH	An integer scalar defining the number of joints that receive load in either the current or previous load increments

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE GENAB

This routine generates the A and B matrices.

Algorithm

The B matrix's three members are calculated first from the cross product of the vector to the current footprint location and the unit vector in the direction of load application. The A array (dimension 3X NJTN) is created second, using the cross products of the vectors to the joints that are to receive load and the unit vector in the direction of load application.

Input/Output

None

Argument List

CØØRJ A real array of coordinates of all the joints

JØINTN An integer array defining numbers of joints that are to be loaded in this load increment

ABETA A real array defining the A array for this load increment

BBETA A real array defining the B array for this load increment

CBETA A real array of the coordinates of the center of the footprint

UVAPP A real array defining the unit vector in the direction of load application

NJTN An integer scalar defining the number of joints to be loaded in this load increment

IRØW2 An integer scalar defining the number of one of the principal axes in the reference plane

IRØW3 An integer scalar defining the number of the other principal axis in the reference plane

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE INVERT

This routine inverts a 3 x 3 matrix by the method of successive transformations.

Algorithm

An identity matrix is set up first, then, while the matrix to be inverted is turned into an identity matrix, the row operations to do so are also performed on the identity matrix. When the input matrix is an identity matrix, the identity matrix has been transformed into the inverse matrix.

Input/Output

None

Argument List

AAT	A real array defining the input matrix to be inverted
AATIN	A real array defining the inverted output matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MATMUL

This routine performs the matrix multiplication $A^T(AA^T)^{-1}BF$.

Algorithm

The first step is to obtain AA^T, then the matrix multiplication $A^T(AA^T)^{-1}$ is performed. The final two steps are the matrix multiplication $A^T(AA^T)^{-1}B$ and the scalar multiplication $A^T(AA^T)^{-1}BF$.

Input/Output

None

Argument List

ABETA	A real array defining the A matrix for this load increment
PREPRØ	A real array of storage area for the pre-product
PHINEW	A real array defining the final matrix product of loads on the joints
JØINT	An integer array defining the numbers of joints that are to be loaded in this load increment
BBETA	A real array defining the B matrix
F	A real scalar defining the total load for this increment
NJTN	An integer scalar defining the number of joints to receive load in this increment

Labeled Common

None

Subroutines Called

INVERT

Error Detection

None

SUBROUTINE ØUTPUT

This routine prints, punches and writes (on tape) the output data.

Algorithm

The output is produced in two forms. The first form is for the user to use for checking the input data. The constant data (mass velocity, duration of impact, average impact force, and the load application unit vector) are printed first, then the current load and the change in load that occurred between the preceding and current load increments for each joint that undergoes a change in load are printed. The second form of output is printed for the benefit of the user and written on tape for analysis programs downstream. The format used is that for a FØRMAT matrix whose rows are the degree of freedom numbers and whose columns are the load increment numbers. The change in load between the previous and current load increments are divided along the degrees of freedom and are written accordingly.

Input/Output

The output is written on files 6, JTAPE and KTAPE.

Argument List

DELPHI A real array of the change in joint loads from the previous to the current load increment

JØINT An integer array of the numbers of joints that undergo changes in load from the previous to the current load increment

PHINEW A real array of the loads on the joints in the current increment

JØINTN An integer array of the numbers of joints that receive load in the current load increment

IBETA An integer scalar defining the number of the load increment

F A real scalar defining the total load applied in the current increment

UVAPP A real array defining a unit vector in the direction of the applied load

NJTN An integer scalar defining the number of joints that received load in the current increment

NDELPH An integer scalar defining the number of joints that undergo changes in load between the current and preceding load increments

NDØF An integer scalar defining the total number of degrees of freedom

AMASS A real scalar defining the mass of the bird

VEL A real scalar defining the velocity of the bird

TIME A real scalar defining the duration of impact

FAVG A real scalar defining the average load

UVEL A real array defining a unit vector of the velocity of the bird

JTAPE An integer scalar defining the number of the output file that will contain the FØRMAT matrix card images

KTAPE An integer scalar defining the number of the output file that corresponds to the FØRMAT tape output

NBETAP An integer scalar defining the total number of load increments plus one

CØLM A real array used to assemble a column of the output matrix

Labeled Common

None

Subroutines Called

SQUEEZ

Error Detection

None

SUBROUTINE READB

This routine reads the footprint travel and load factor for the current load increment.

Algorithm

The input file is read until a Data Code 11 card is read. The Data Code 11 cards are read until a card with a flag of 6 is found. The cards with a flag of 6 are read until the card with the current increment number is read and the data is stored. The subroutine then returns to the main program.

Input/Output

The input is read from file ITAPE.

Argument List

DBETA A real scalar defining the distance the footprint travels from the initial impact point

FACTOR A real scalar defining the current position portion of the average impact load to be applied for the current load increment

IBETA An integer scalar defining the current increment number

ITAPE An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 11 cards are read the message 'NO CONSTANTS (DATA CODE 11) ENCOUNTERED IN INPUT'

is printed. If the card with data corresponding to the current load increment is not read the message
'INCREMENT (BETA) NUMBER nnn NOT FOUND IN DATA CODE 11 DATA'
is printed.

SUBROUTINE READCN

This routine reads the constants associated with the impact distribution.

Algorithm

The input file is read until a Data Code 11 card is read. The input file is backspaced one record and five cards are read and their data stored before the subroutine returns to the main program.

Input/Output

The input is read from file ITAPE.

Argument List

NDØF — An integer scalar defining the total number of degrees of freedom

AMASS — A real scalar defining the mass of the bird

VEL — A real scalar defining the velocity of the bird

ALENG — A real scalar defining the length of the bird

UVEL — A real array defining a unit vector in direction of bird motion

UNØRM — A real array defining a unit vector normal to the surface at the impact point

UFØØT — A real array defining a unit vector in the direction of impact footprint travel

CØØRI — A real array defining a vector from the origin to the impact point

NBETA — An integer scalar defining the number of load increments

ITAPE — An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 11 cards are read the message
'NO CONSTANTS (DATA CODE 11) ENCOUNTERED IN INPUT'
is printed. If there are not at least five consecutive Data Code 11 cards, the message
'MISSING DATA IN DATA CODE 11'
is printed.

SUBROUTINE READCØ

This routine reads the joint coordinate data.

Algorithm

The input file is read until a Data Code 2 card is found. The input file is backspaced one record and the coordinate data is read and stored until the data code is no longer 2 or until a joint number of 9999 is read.

Input/Output

The input is read from file ITAPE.

Argument List

CØØRJ	A real array of coordinates of joints stored by joint number
NJTØT	An integer scalar defining the largest joint number input
ITAPE	An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If the end of Data Code 2 data is encountered without encountering joint number 9999 the message
'A 29999 CARD WAS NOT FOUND'
is printed but execution continues. If no Data Code 2 data is found at all, the message
'NO JOINT COORDINATE DATA (DATE CODE 2) ENCOUNTERED IN INPUT'
is printed.

SUBROUTINE READJ

This routine reads the joint numbers that will receive load in the current load increment.

Algorithm

The input file is read until a Data Code 12 card is encountered. The file is backspaced one record and the data is read, checking the increment number. When the current increment number is read the joint numbers are read and stored for as many cards as have the current increment number.

Input/Output

The input is read from file ITAPE.

Argument List

JØINT An integer array defining the numbers of joints that are to receive load during the current increment

NJTN An integer scalar defining the number of joints that are contained in JØINT

IBETA An integer scalar defining the current load increment number

ITAPE An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 12 cards are read the message
'NO LOADED JOINTS DATA (DATA CODE 12) ENCOUNTERED IN INPUT'

is printed. If a card with data corresponding to the current load increment is not read the message
'JOINTS FOR INCREMENT (BETA) NUMBER nnnn NOT FOUND IN DATA CODE 12 DATA'
is printed.

APPENDIX D

LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

APPENDIX D
LINEAR INCREMENTAL ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table D gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usage	list of all routines which contain declarations for the labeled common block

TABLE D. INDEX TO LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

TABLE D. INDEX TO LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

MAIN PROGRAM RESPNS

This is the driver of the Linear Incremental Solution program.

Algorithm

The driver functions of this program consist of:

1) Defining 23 data sets used for input, output and scratch storage.
2) Delimiting the extents of 4 common blocks, the first of which, blank common, will be used for working storage.
3) Initializing via subroutine NITIAL.
4) Setting the stage for an incremental analysis by invoking subroutine PREEIG to generate matrices $\overline{M}$, $\overline{K}$ and $\overline{C}$ which are the components of the eigenvalue problem statement set up and solved via subroutine EIGSOL.
5) Performing the incremental analysis through the sequence of calls to subroutines LINEAR and IMBAL for each time interval.
6) Clocking the execution of each module and displaying elapsed time on output.

Input/Output

None

Arguments

None

Labeled Common

NDICES, TAPES, LIMITS

Subroutines Called

NITIAL, PREEIG, EIGSØL, LINEAR, IMBAL, TSETQ, TIMEQ

Error Detection

None

LABELED COMMON CLØCK

This common block is used to store timing information to obtain CPU time required for major modules within the program.

Declaration

```
CØMMØN /CLØCK/ ØRGTIM
```

Contents

ØRGTIM   CPU time remaining for the run obtained by system routine TIMREM

Usage

TIMEQ, TSETQ

LABELED COMMON LIMITS

This common block is used to store problem size information, problem constants, option flags, and other program parameters used by all principle routines.

Declaration

```
CØMMØN /LIMITS/NG,NK,NELEMS,NTRVLS,BETA,BETBAR,HEAT,NM,IØRGN,JX
,               KØLX,NF,MØREK,MØREC,NG2,NG2SQ,NJTS,AHAT,DEBCL,TBM1
,               TAU,NS,NØWRK,NSUPD,NFAIL,NØPBPU
LOGICAL BETBAR,NØBPU
INTEGER BETA
```

Contents

NG	Number of modes
NK	Number of lumped forces for an element
NELEMS	Total number of physical elements
NTRVLS	Number of time increments
BETA	Current increment number
BETBAR	Not used
HEAT	Option flag for element dissipated damping energy converted to heat
NM	(NG*NG+NG)/2
IØRGN	Pointer to unused trailing partition of blank common
JX	Pointer to a location in labelled common NDICES
KØLX	Matrix column counter
NF	Number of forces for an element
MØREK	Flag designating additional partitions of $\bar{k}$
MØREC	Flag designating additional partitions of $\bar{c}$
NG2	NG+NG
NG2SQ	NG2*NG2

NJTS	Number of joints
AHAT	Not used
DEBCL	Total damping energy
TBM1	Time increment of previous interval
TAU	Time increment of current interval
NS	Number of stresses for an element
NWØRK	Extent of blank common region
NSUPD	Not used
NFAIL	Not used
NØPBPU	Flag indicating the input $\delta\bar{P}_{(\phi)U}$ has been exhausted

Usage

RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR

LABELED COMMON NDICES

This common block is used to store pointers to partitions within blank common array A. All values in this common block are initialized in routine NITIAL. Partitions for elements are sized for cells which have the largest storage requirements.

Declaration

```
CØMMØN        A( 8000)
CØMMØN /NDICES/IVBDB,IVBB,IVBX,IDPBPU,IPBMUB,IDPPUB,IDBL,IVBL
,              IPBAR,IPBPHU,IKB,ICB,IMBAR,IMBARL,IDEBCL,ICØNST
,              ITIME,ISIGSS
,              IKBL,IZ,ICA,IQ,IVAL,IFTAU,IVEC,IEGSYS
,              IU,LU,IECT,IEVT,IMPT,IDEBO,IDFBKO,IDSEB
,              IFK,IFBK,IDEL,IDEDL,IFSFB,IFSFBB.IPBCU
,              IPBKU,ISIGFB,ISIGBH,IEPSIG,IPSLØN,ID,IDK
,              ISKB,ISKBB,ISCB,ISCBB,IPBUF,ISK,ITK,ICIB,IMEL
```

Contents

IVBDB	$\dot{\bar{v}}_{\beta}$, modal accelerations
IVBB	$\bar{v}_{\beta}$, modal velocity
IVBX	$\bar{v}_{\beta-1}$, modal velocity of previous increment
IDPBPU	$\delta\bar{P}_{U}$, incremental modal force imbalance
IPBMUB	$\bar{P}_{(M)U}$, modal inertia force
IDPPUB	$\delta\bar{P}_{(\phi)U}$, incremental modal applied load
IDBL	$\delta\bar{\Delta}_{L}$, incremental linear modal displacement
IVBL	$\bar{v}_{L}$, linear modal displacement
IPBAR	$\bar{P}$, total modal forces
IPBPHU	$\bar{P}_{(\phi)U}$, modal applied load
IKB	$\bar{K}$, modal stiffness
ICB	$\bar{C}$, modal damping
IMBAR	$\bar{M}$, modal mass

IMBARL	Cholesky decomposition of $\bar{M}$
IDEBCL	$\delta\bar{E}_{CL}$, incremental element dissipated damping energy
ICØNST	Problem constants
ITIME	Incremental time history
ISIGSS	$\sigma_{S\sigma}$, stress transform for cell elements
IKBL	Cholesky decomposition of $\bar{K}$
IZ	The modal matrix $Z = \bar{K}^{-1}_{\beta-1}\bar{P}_{\beta}$
ICA	The modal matrix $C_a = H^{-1} yo$
IQ	The modal matrix $Q = C_{a_D} F(\tau)$ where C_{a_D} is C_a diagonalized
IVAL	λ, eigenvalues
IFTAU	$F(\tau)$, a modal column matrix of $e^{\lambda\tau}$
IVEC	H, eigenvectors
IEGSYS	Eigenvalue problem work array
IU	U_o, original joint coordinates
LU	Not used
IECT	ECT, element constant table
IEVT	EVT, element variable table
IMPT	MPT, material property tables
IDEBO	$\delta\bar{e}_o$, incremental initial element deformations
IDFBKO	$\sigma\bar{F}_{K_o}$, incremental element forces due to initial deformations
IDSEB	$\delta\bar{e}$, incremental element deformations
IFK	The matrix $FK = \bar{F}_{K_{\beta-1}} + \delta\bar{F}_{K_o}$ for an element
IFBK	$\bar{F}_K$, element forces
IDEL	δe_L, incremental linear element displacement
IDEDL	$\delta\dot{e}_L$, incremental linear element velocity

IFSFB	$F_{\bar{F}}$, element force transform
IFSFBB	Not used
IPBCU	$\bar{P}_{(C)U}$, modal damping forces
IPBKU	$\bar{P}_{(K)U}$, modal stiffness forces
ISIGFB	$\sigma_{\bar{F}}$, element stress transform
ISIGBH	Not used
IEPSIG	ϵ_{σ}, element strain transform
IPSLØN	ϵ, element strains
ID	The matrix $D = \bar{P}_{UF} F_{\bar{F}}$ for an element
IDK	The matrix $DK = \bar{P}_{UF} F_{\bar{F}} \bar{k}$ for an element
ISKB	$\bar{k}$, unassembled element stiffness
ISKBB	Not used
ISCB	$\bar{c}$, unassembled element damping
ISCBB	Not used
IPBUF	$\bar{P}_{UF}$, modal transform
ISK	s_k, modal fictitious forces
ITK	t_k, modal fictitious deformations
ICIB	The matrix $\overline{CI} = \bar{P}_{UF} F_{\bar{F}} \bar{c} F_F^T \bar{P}_{UF}^T$ for an element
IMEL	m, element mass

Usage

RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR

LABELED COMMON TAPES

This common block is used to store the FORTRAN logical unit numbers of the external files used by the program.

Declaration

```
COMMON /TAPES /N1 ,N2 ,N3 ,N4 ,N5 ,N6 ,N7 ,N8 ,N9 ,N10
,              N11,N12,N13,N14,N15,N16,N17,N18,N19,N20
```

Contents

The values in the common block are initialized in routine NITIAL. The FORTRAN logical unit designations assigned are given here.

N1	1
N2	2
N3	3
N4	4
N5	8
N6	9
N7	10
N8	11
N9	12
N10	13
N11	14
N12	15
N13	16
N14	17
N15	18
N16	19
N17	20
N18	21
N19	22
N20	23

Usage

RESPNS, EIGSØL,IMBAL,LINEAR,NITIAL,ØUTPUT,PBARUF,PREEIG,PTMASS, READK, SIGFBR

SUBROUTINE ABSYM

This routine performs a matrix cross-product wherein the post-multiplier matrix is the upper half of a symmetric matrix.

Algorithm

The mode of storage of the post-multiplier matrix is b_{11}, b_{12} $\cdots$ b_{1n}, b_{22} $\cdots$. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix sum $C = C + A \cdot B$.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix in the mode of storage b_{11}, b_{12}, $\cdots$, b_{1n}, b_{22} $\cdots$

M An integer scalar defining the order of the pre-multiplier matrix

N An integer scalar defining the order of the post-multiplier symmetric matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ADD

This routine performs vector addition; C = A + B.

Algorithm

Vectors A and B are summed into C.

Input/Output

None

Argument List

C A real array accommodating the resulting vector

A A real array accommodating the first input vector

B A real array accommodating the second input vector

N An integer scalar defining the order of the vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ASYMB

This routine performs a matrix cross-product wherein the pre-multiplier matrix is the upper half of a symmetric matrix.

Algorithm

The mode of storage of the symmetric pre-multiplier matrix is $a_{11}, a_{12}, \cdots, a_{1n}, a_{22} \cdots$. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix sum $C = C + A \cdot B$.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix in the mode of storage $a_{11}, a_{12}, \cdots, a_{1n}, a_{22} \cdots$

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the order of the symmetric pre-multiplier matrix

P An integer scalar defining the column order of the pre-multiplier matrix

SUM A logical scalar which if true, $C = C + A \cdot B$; otherwise $C = A \cdot B$

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CCMULT

This routine computes the specified leading rows of the product of two complex matrices.

Algorithm

When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix product and summation $C = C + A*B$.

Input/Output

None

Argument List

C	A complex array accommodating the product matrix
A	A complex array accommodating the pre-multiplier matrix
B	A complex array accommodating the post-multiplier matrix
M	An integer scalar defining the row dimension of arrays C and A
N	An integer scalar defining the column dimension of array A, and the row dimension of array B
P	An integer scalar defining the column dimension of arrays B and C
ML	An integer scalar defining the computations concerning ML rows of C
SUM	A logical scalar which if true, $C = C + A*B$; otherwise $C = A*B$

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CHGSGN

This routine changes the algebraic sign of a vector.

Algorithm

The sign of the vector is reversed by being set to minus.

Input/Output

None

Argument List

A	A real array accommodating the vector whose sign is to be reversed
N	An integer scalar defining the order of the vector

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CHLSKY

This routine performs an in situ Cholesky decomposition, where the symmetric matrix is given in its triangular half.

Algorithm

Where $A = LL^T$, given the positive definite matrix A in its symmetric half, this routine transforms it to L. The mode of storage is a_{11}, a_{12}, $\cdots$ a_{1n}, a_{22} $\cdots$ a_{2n}, etc.

Input/Output

None

Argument List

A — A real array accommodating the symmetric half of the positive definite matrix on input, and its square root decomposition on output

N — An integer scalar defining the order of the problem

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE COPY

This routine copies an array from core to core.

Algorithm

Array A is copied into array B.

Input/Output

None

Argument List

A	A real array to be copied
B	A real array into which the copy is made
N	An integer scalar defining the order of the arrays

Labeled Common

None

Subroutined Called

None

Error Detection

None

SUBROUTINE CRMULT

This routine computes the specified leading rows of a matrix product wherein the pre-multiplier matrix is complex.

Algorithm

When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix product and summation C = C + A*B.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real complex array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the row dimension of arrays C and A

N An integer scalar defining the column dimension of array A and the row dimension of array B

P An integer scalar defining the column dimension of arrays B and C

ML An integer scalar defining the computations concerning ML rows of C

SUM A logical scalar which if true, C = C + A*B; otherwise, C = A*B

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE EIGSOL

This routine sets up, executes and disposes of the eigen problem.

Algorithm

The upper quadrants of array A are filled with the negative of symmetric modal damping and mass matrices, $\bar{C}$ and $\bar{M}$, reconstructed in their full forms. The lower left quadrant is set to the identity. Subroutine SYMSOL is then invoked, with the Cholesky decomposition of modal stiffness matrix $\bar{K}$ as argument, to arrive at:

$$\begin{bmatrix} -\bar{K}^{-1}\bar{C} & -\bar{K}^{-1}\bar{M} \\ I & 0 \end{bmatrix}$$

Subroutine RGEIG now performs the complete eigenvalue problem solution. The eigenvalues are reciprocated. In this process, equal eigenvalues corresponding to equal eigenvectors are recognized as being part of a singular system. An attempt to remedy this condition is made by scalar multiplying matrix $\bar{C}$ with 1.0001 and going through the complete procedure anew. Should this instability persist, the printing of an appropriate statement is followed by a call to subroutine HALT. Otherwise, the writing of the matrix of eigenvectors onto data set N8 is followed by their inversion through subroutine JØRDAN. The inverse is also output onto data set N8.

Input/Output

To conserve core space, the matrix of eigenvectors and its inverse are stored on data set N8.

Argument List

B	A real array accommodating the eigenvalue problem statement

X	A real array accommodating the matrix of eigenvectors
M	An integer which defines the order of the eigenvalue problem
N	An integer whose value is M/2

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

CHLSKY, CØPY, HALT, JØRDAN, RGEIG, SMULT, SYMFIL, SYMSØL, WRITE, ZERØ

Error Detection

Equal complex eigenvectors with corresponding equal eigenvalues are detected as singular. An attempt to remedy this condition is made by scalar multiplying matrix $\overline{C}$ with 1.0001. Should the unstable condition persist, the printing of an appropriate message is followed by the call to subroutine HALT.

SUBROUTINE HALT

This routine performs termination processing as necessary in the event of a fatal error.

Algorithm

In anticipation of certain processing requirements before job termination initiated by the detection of a fatal error, this routine and appropriate calls were provided. Subsequently, however, it was determined that no processing of this nature was necessary.

Input/Output

A job termination message is output to unit 6.

Argument List

None

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE IMBAL

For each physical element within each type of element, this routine computes the force, stress and strain matrices, $\overline{F}_K$, σ and ε, and writes them onto the output data set.

Algorithm

The output matrices concern bar elements, membrane elements and cell elements. Their respective FØRMAT matrix names are BARS, MEMBRN and CELLS. Each physical element is represented by a column whose contents are vectors $\overline{F}_K$, σ and ε. These quantities are computed as follows:

$$\overline{F}_K = \delta\overline{F}_{K_o} - DK^T\,\overline{\Delta}$$

where $$DK = \overline{P}_{UF}\;F_{\overline{F}}\;\overline{K}$$

$$\sigma = \sigma_{S\sigma}\;\sigma_{\overline{F}}\;\overline{F}_K \quad \text{for cells}$$

or $$\sigma = \sigma_{\overline{F}}\;\overline{F}_K \qquad \text{for bars and membranes}$$

$$\varepsilon = \varepsilon_\sigma\;\sigma$$

Input/Output

Tape 2 is the output data set. For each physical element, matrix $\sigma_{\overline{F}}$ is read from data set N14; the matrix quantity $\overline{P}_{UF}\;F_{\overline{F}}\overline{K}$ is read from data set N7; $\delta\overline{F}_{Ko}$ is read from N15; and $\varepsilon\sigma$ is read from N16.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ADD, ASYMB, MULT, ØUTPUT, READA, READM, SBTRCT, TMULT, ZERØ

Error Detection

None

SUBROUTINE JØRDAN

This routine inverts in situ a complex matrix by pivoting on the available row of largest magnitude within the next available column.

Algorithm

The method used is Gauss-Jordan. The array KØL is used to record the order of rows which have been selected for pivoting.

Input/Output

None

Argument List

A	A real matrix which is transformed to its inverse prior to exit from this routine
KØL	An integer work array of dimension 2*N. Prior to exit from this routine, the first N locations contain the order in which rows have been selected for pivoting. The second N locations contain the values of the pivoting elements.
N	An integer scalar defining the order and dimension of matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LINEAR

This routine computes vectors $\overline{P}$, $\delta\overline{\Delta}$, $\overline{v}$ and $\dot{\overline{v}}$.

Algorithm

Data set N8 is already positioned at the origin of the matrix of inverse eigenvectors. It is rewound as soon as this matrix has been read. The next read then brings in the matrix of eigenvectors proper.

$\delta\overline{\Delta}$ and $\overline{v}$ are computed in the complex mode and transferred to their final blank common destinations in the real mode. Through abstraction like CALL statements to utility routines, the matrix equations to compute $\overline{P}$, $\delta\overline{\Delta}$, $\overline{v}$, and $\dot{\overline{v}}$ are solved in the following manner:

$$\overline{P} = \overline{P} + \overline{P}_{(M)U} + \delta\overline{P}_{(\phi)U}$$

$$Z = -\overline{K}^{-1}\ \overline{P}$$

$$C_{aD} = H^{-1} \begin{bmatrix} Z \\ \hdashline v \end{bmatrix} \text{ diagonalized}$$

$$F(\tau) = e^{\lambda\tau}$$

$$Q = C_{ad}\ F(\tau)$$

$$\dot{\overline{v}} = H_v\ \lambda_D\ Q$$

$$\begin{bmatrix} \delta\overline{\Delta} \\ \hdashline \overline{v} \end{bmatrix} = HQ$$

$$\delta\overline{\Delta} \quad \delta\overline{\Delta} - Z$$

Input/Output

The matrix of eigenvectors and its inverse are read from data set N8. The incremental applied load matrix $\delta\overline{P}_{(\phi)U}$ is from data set N18.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ADD, CCMULT, CHGSGN, CØPY, CRMULT, EUTL9, PRINTR, READA, SYMSØL, VDMULT, XPØNNT, ZERØ

Error Detection

None

SUBROUTINE MATIN

This routine routes FØRMAT Program input matrices and arrays from tape 20 to the intended, or interim, data sets.

Algorithm

Reading over the data set header positions it at the origin of its first matrix.

The following takes place with each item sandwiched between a matrix header and trailer. The data set designation, onto which the next matrix in sequence is to be written, is taken from the list NT and rewound. The matrix header is read and ignored. Each record thereafter is transcribed. Additionally, the third matrix is also written on data set 14. This process continues as long as the first word of the records thus processed is a positive integer. Except in the case of matrix $\overline{P}_{UF}$, the output data set is then rewound.

Input/Output

The list of throughput matrices and the data set onto which they are output follows:

Matrix Name	Data Set Number	Description
$\overline{P}_{UF}$	14	Modal element force transform
MPT	3	Material property table
U_o	8, 14	Original joint coordinates
ECT	15	Element constant table
M_{EL}	13	Element mass matrix
$\overline{k}$	11	Element stiffness matrix
$\overline{c}$	17	Element damping matrix
$F_{\overline{F}}$	4	Element force transform

Matrix Name	Data Set Number	Description
$\sigma_{\bar{F}}$	15	Element stress transform
ϵ_{σ}	22	Element strain transform
$\delta_{\bar{e}_T}$	10	Initial thermal element deformations
EVT	9	Element variable table

Argument List

A	A real work array
NUM	An integer scalar used as a dimensioning variable

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MBAR

This routine fetches the upper half of the next uncoupled partition from a tape resident symmetric quasi-diagonal matrix.

Algorithm

The end product of this routine consists of a triangular matrix of order NF in expanded format wherein the rows begin from their diagonal elements. As each column (KØLX + 1) through (KØLX + NF) is read in the compressed format, its elements preceding the diagonal are discarded. The row designations of the elements retained are reduced by the quantity KØLX - NF*(I-1), where I represents the sequence of these columns from 1 through NF. One call to subroutine EUTL9 expands this triangular matrix in its desired form.

Input/Output

None

Argument List

NF An integer scalar defining the number of rows and columns in desired matrix partition

KØLX An integer scalar defining the designation of the last column read on previous entry into this routine

A A real storage array

NT An integer scalar defining the designation of the data set containing the matrix

Labeled Common

None

Subroutines Called

EUTL9, READA, ZERØ

Error Detection

None

SUBROUTINE MULT

This routine performs a matrix product.

Algorithm

The matrix product A*B is added to matrix C. Matrix C is initialized to zero only when the logical variable SUM is false.

Input/Output

None

Argument List

C — A real array accommodating the product matrix

A — A real array accommodating the pre-multiplier matrix

B — A real array accommodating the post-multiplier matrix

M — An integer scalar defining the row dimension of matrices C and A

N — An integer scalar defining the row dimension of matrix B and the column dimension of matrix A

P — An integer scalar defining the row dimension of matrices C and B

SUM — A logical scalar which if false, array C is initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

FUNCTION NDXSET

This routine defines the elements of a vector used to segment the working core.

Algorithm

The variable IØRGN contains the relative location from which the work array is available for segmentation. The following takes place for each of the N apportionments: 1) the value of JX, the indexing pointer used with vector NDEX, is incremented; 2) IØRGN is stored into NDEX(JX); 3) IØRGN is increased by INC, the extent of core being allocated each origin.

Input/Output

None

Argument List

INC	An integer scalar defining the extent of core to be allocated with each origin NDEX(I)
N	Number of origins to be defined

Labeled Common

LIMITS, NDICES

Subroutines Called

None

Error Detection

None

SUBROUTINE NITIAL

This is the intializing routine.

Algorithm

The variables N1 through N20, which are contained within labeled common TAPES, are sequencially set to represent tapes 1 through 4, and 8 through 23.

The input matrices are read in their FØRMAT generated mode and written onto their interim, or intended, data sets via subroutine MATIN. The following delimiting parameters are read from the first input card.

Item	Type	Description
BETA	INTEGER	Starting time
NELEMS	INTEGER	Number of structural elements
NTRVLS	INTEGER	Number of time intervals
NG	INTEGER	Number of modes
HEAT	LOGICAL	"TRUE" indicates presence of thermal conditions
NJTS	INTEGER	Number of structural joints
NWØRK	INTEGER	Number of words in blank common

Blank common array A is apportioned to the various matrices and arrays through the designation of their respective origins by the index from labeled common NDICES.

The NTRVLS values concerning the time array are read into the space beginning with A(ITIME) and checked for their being in ascending order. Execution is halted with an appropriate statement when this check fails.

The constant matrix $\sigma_{S\sigma}$ for cell elements is generated and stored at A(ISIGSS).

Overlay within array A is effected by assigning IKBL and IU the same value. The first of these subscripts represent the origin of space used by subroutine EIGSØL only; the second is the leading subscript of non-conversant space used by other modules.

Subroutine READK is invoked to merge selected matrices residing on interim data sets. The arrays which are to accommodate the following vectors are initialized with zeroes: $\overline{v}$, $\dot{\overline{v}}$, $\overline{v}_{\beta-1}$, $\delta\overline{P}_U$, $\delta\overline{P}_{(M)U}$, $\delta\overline{P}_{(C)U}$, $\delta\overline{P}_{(\phi)U}$, and $\delta\overline{E}_{CL}$.

Input/Output

All card input takes place in this routine. In addition, the following input matrices are written onto the post-processing data set, tape 2: the array of constants under its input FØRMAT program matrix name; the array of time intervals under the matrix name TIME; the array of joint coordinates under the matrix name UZERØ; vectors $\overline{P}_{(\phi)U}$ and $\overline{P}_{UPT}$ under their input names.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ASYMB, CHGSGN, MATIN, NOXSET, READA, READK, SIGFBR, SQRT, WRITE, ZERO

Error Detection

None

SUBROUTINE OUTPUT

This routine stores output onto the output data set in the mode of the FØRMAT program.

Algorithm

Vectors $\overline{\Delta}$, $\delta\overline{\Delta}$, $\overline{v}$, $\dot{\overline{v}}$, $\delta\overline{P}_U$, and $\overline{P}_{(\phi)U}$ are written onto data set 2 as a single column FØRMAT matrix with the name RESPNS.

Input/Output

Matrix RESPNS is output to data set 2.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

WRITE

Error Detection

None

SUBROUTINE PBARUF

This routine reads a partition of matrix $F_{\bar{F}}$ and the corresponding partition of matrix $\bar{P}_{UF}$.

Algorithm

Data set N13 contains matrix $F_{\bar{F}}$ in the form of one partition per record. Reading one such record provides the information necessary to determine the columns bounding the desired partition of $\bar{P}_{UF}$.

Inout/Outout

Matrix $F_{\bar{F}}$ is read from data set N13. Matrix $\bar{P}_{UF}$ is read from data set N11.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutine Called

READA, EUTL9

Error Detection

None

SUBROUTINE PREEIG

This routine generates matrices $\overline{M}$, $\overline{K}$, $\overline{C}$ and $\delta\overline{P}_{KO}$.

Algorithm

For each element, the matrix operations are performed through abstraction like CALL statements to utility routines. Having initialized matrices $\overline{K}$, $\overline{C}$, $\overline{M}$, $\overline{P}$, and $\overline{P}_{(M)U}$ as zero, the matrix operations performed are as follows:

$$\delta\overline{F}_{KO} = -\overline{k}\ \delta\overline{e}_T$$

$$D = \overline{P}_{UF}\ F_{\overline{F}}$$

$$\overline{P} = \overline{P} + D\ \delta\overline{F}_{KO}$$

$$DK = D\overline{k}$$

$$\overline{K} = \overline{K} + DK\ D^T$$

$$\overline{C} = C + D\overline{c}\ D^T$$

$$\overline{M} = \overline{M} + \overline{P}_{UF}\ m\ \overline{P}_{UF}{}^T$$

If point mass elements exist, their contribution to the modal mass matrix, $\overline{M}$, is computed and added by a single call to subroutine PTMASS.

Input/Output

For each physical element, a partition of the matrices $\overline{k}$, $\overline{c}$ and $\delta\overline{e}_T$ is read in that sequence from data set N1. Similar partitions of matrices $\overline{P}_{UF}$, $F_{\overline{F}}$, and m are read from data sets N11, N13 and N10, respectively. The contribution of each physical element to matrix DK is written onto data set N7 and, similarly, the contribution to δF_{KO} is written onto N15.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutine Called

ABSYM, ADD, ASYMB, CHGSGN, MBAR, MULT, PBARUF, PTMASS, READA, SYMABT, WRITEM, ZERØ

Error Detection

None

SUBROUTINE PRINTR

This routine prints a core resident matrix.

Algorithm

Groups of not more than 8 columns across each page are written for partitions of not more than 55 rows. Each page is headed with the title. The column designations appear under the title. The row designations precede each row. Only the most significant aspect of double precision matrices is written. Two calls to this routine are required to print a complex matrix.

Input/Output

None

Argument List

A A real array accommodating the matrix to be printed

K An integer scalar used as a dimensioning variable. K = 1 for real and single precision arrays. K = 2 for compressed or double precision arrays. K = 4 for double precision complex arrays.

M An integer scalar defining the row dimension

N An integer scalar defining the column dimension

TITLE An alphanumeric scalar for header data

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE PTMASS

This routine concludes the computations of $\overline{M}$ with the contributions of the point-mass elements.

Algorithm

The computations concern the triple product $\overline{P}_{UF}\ M_{EL}\ \overline{P}_{UF}^{\ T}$ involving point mass elements. M_{EL} consists of one element. The columns of these matrices are read and the computations are exercised upon matching the column designations.

Input/Output

Matrix M_{EL} is read from data set N10. Matrix $\overline{P}_{UF}$ is read from data set N11.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

READA, EUTL9, ABSYM, SYMABT

Error Detection

None

SUBROUTINE READK

This routine transcribes FØRMAT program generated matrices from interim data sets to their intended data sets.

Algorithm

The symmetric quasi-diagonal matrices $\overline{k}$, e_σ and $\overline{c}$ are read in their full forms. They are retained only in their symmetric halves prior to being written as single compressed records.

Matrix δe_T is simply copied from its input to its output tape. The quasi-diagonal matrix $F_{\overline{F}}$ is output in records of sub-matrix partitions.

Input/Output

Matrices $\overline{k}$, e_σ, $\overline{c}$, $\delta\overline{e}_T$, and $F_{\overline{F}}$ are respectively read from data sets N8, N19, N14, N10, and N4; matrices e_σ and $F_{\overline{F}}$ are written onto data sets N16 and N13. Matrices $\overline{k}$, $\overline{c}$ and $\delta\overline{e}_T$ are written onto data set N1 in the sequence $\overline{k}_1$, $\overline{c}_1$, $\delta\overline{e}_1$, $\overline{k}_2$, $\overline{c}_2$, $\delta\overline{e}_{T_2}$, ····, etc.

Argument List

None

Labeled Common

LIMITS, NOICES, TAPES

Subroutines Called

EUTL9, MBAR, READA, WRITE

Error Detection

None

SUBROUTINE READM

This routine reads a matrix which was written on a scratch data set via subroutine WRITEM.

Algorithm

The matrix is read via subroutine READA. Should it be compressed, subroutine EUTL9 expands it.

Input/Output

The matrix is read from data set NT.

Argument List

M	A dummy argument
N	A dummy argument
A	A real array into which the matrix is read
NT	An integer scalar defining the input data set

Labeled Common

None

Subroutine Called

None

Error Detection

None

SUBROUTINE SBTRCT

This routine computes the arithmetic difference between two vectors.

Algorithm

The algebraic difference between vectors A and B is stored in vector C.

Input/Output

None

Argument List

C	A real array for the resulting vector
A	A real array for the vector being subtracted
B	A real array for the subtracting vector
N	An integer scalar defining the order of vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SIGFBR

This routine partitions a quasi-diagonal matrix and outputs its partitions in single compressed records.

Algorithm

The columns of each partition are read and stored consecutively in their compressed format. In the process, the row designations are replaced by the location of the elements relative to the current submatrix. A single call to subroutine EUTL9 expands the partition which is then output by subroutine WRITEM.

Input/Output

The matrix is read from data set N14 and written onto data set N12.

Argument List

None

Labeled Common

TAPES, LIMITS, NDICES

Subroutines Called

EUTL9, READA, WRITEM

Error Detection

None

SUBROUTINE SMULT

This routine scales a matrix.

Algorithm

Each element of array A is multiplied by the scalar S, and the product is stored in array C.

Input/Output

None

Argument List

C	A real array accommodating the output matrix
S	A real scalar coefficient
A	A real array accommodating the matrix to be scaled
M	An integer scalar defining the row dimension
N	An integer scalar defining the column dimension

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMABT

This routine computes the triangular half of a symmetric matrix where the post-multiplier is transposed.

Algorithm

The mode of storage of the product matrix is $C_{11}\cdots C_{1n}$, $C_{22}\cdots C_{2n}$, etc. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero.

Input/Output

None

Argument List

C	A real array for the triangular product matrix
A	A real array for the pre-multiplier matrix
B	A real array for the transposed post-multiplier matrix
M	An integer scalar defining the row dimension of all matrices
N	An integer scalar defining the column dimension of A and B matrices
SUM	A logical scalar when false, causes the resulting matrix to be initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMFIL

This routine stores a symmetric half matrix into the partition of an array while reversing the sign.

Algorithm

The mode of storage of the symmetric half-matrix is $a_{11}, \cdots a_{1n}, a_{22} \cdots$. When K = 1, the receiving matrix is real. When K = 2, it is complex. Note that in this case, the imaginary components are left untouched.

Input/Output

None

Argument List

A	A real array for the symmetric half-matrix
B	A real or complex array for the output matrix
K	An integer scalar where K = 1 for real B array, K = 2 for complex B array
M	An integer scalar defining the row dimension of B array
N	An integer scalar defining the order of symmetric matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMSØL

Given matrices L and B in the expression $LL^TX = B$; this routine transforms B into X.

Algorithm

The mode of storage for matrix L is $l_{11}, \cdots, l_{n1}, l_{22} \cdots$. X is computed by way of Y where $LY = B$. Then, $L^TX = Y$. The array storing matrices B and X may be real or complex. When the integer K is 1, the array is real; when K is 2, the array is complex.

Input/Output

None

Argument List

A A real array accommodating matrix L in the mode of storage $l_{11} \cdots l_{n1}, l_{22} \cdots$

B A real or complex array accommodating matrices B, Y and X

K An integer scalar where array B is real when K = 1, and complex when K = 2

M An integer scalar defining the row dimension of array B

N An integer scalar defining the order of equation

P An integer scalar defining the number of columns in matrices B, Y and X

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE TMULT

This routine performs a matrix cross product wherein the pre-multiplier matrix is transposed.

Algorithm

The matrix product transpose A^TB is added to matrix C. Matrix C is initialized to zero only when the logical variable SUM is false.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the row dimension of matrix C and column dimension of matrix A

An integer row dimension of matrices A and B

N An integer scalar defining the row dimension of matrix C and column dimension of matrix A

An integer row dimension of matrices A and B

P An integer column dimension of matrices B and C

SUM A logical scalar when false causes array C to be initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE VDMULT

This routine performs the element by element product of two complex vectors.

Algorithm

The element by element product of complex vectors A and B is stored in vector C.

Input/Output

None

Argument List

C	A complex array for the output vector C
A	A complex array for the input vector A
B	A complex array for the input vector B
N	An integer scalar defining the order of the vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WRITEM

This routine writes a one-record matrix, compressed (as required).

Algorithm

Following an attempt to compress the matrix via subroutine SQUEEZ, it is written onto the output data set NT.

Input/Output

The matrix is written onto the output data set.

Argument List

M	An integer scalar defining the matrix row dimension
N	An integer scalar defining the matrix column dimension
A	A real array for the input matrix
NT	An integer scalar defining the output data set

Labeled Common

None

Subroutines Called

SQUEEZ, WRITE

Error Detection

None

SUBROUTINE XPØNNT

Given complex vector D and vector T, this routine finds vector $A = e^{D*T}$.

Algorithm

All computations take place in the real mode as vectors A and D are doubly dimensioned arrays.

Input/Output

None

Argument List

A A real array accommodating the output vector

D A real array for the complex input vector treated in the real mode

T A real array for the input vector

N An integer scalar defining the order of vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ZERØ

This routine zeros an array.

Algorithm

The specified extent of array A is set to zero.

Input/Output

None

Argument List

A A real array to be set to zero

N An integer scalar defining the extent of A to be set to zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

APPENDIX E

NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

APPENDIX E
NONLINEAR INCREMENTAL ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table E gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatim source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routinas called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usaoe	list of all routines which contain declarations for the labeled common block

TABLE E. INDEX TO NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

		Page
MAIN PROGRAM	RESPNS	380
LABELED COMMON	CRRAYS	382
LABELED COMMON	FICNDX	383
LABELED COMMON	GEØMS	385
LABELED COMMON	LIMITS	386
LABELED COMMON	LPEM	388
LABELED COMMON	NDICES	393
LABELED COMMON	TAPES	397
SUBROUTINE	ABSYM	325
SUBROUTINE	ADD	326
SUBROUTINE	ASYMB	327
SUBROUTINE	AVRAGE	399
SUBROUTINE	BALANC	Ref. 16
SUBROUTINE	BALBAK	Ref. 16
SUBROUTINE	CCMULT	329
SUBROUTINE	CHGSGN	331
SUBROUTINE	CHKØUT	400
SUBROUTINE	CHKPNT	401
SUBROUTINE	CHLSKY	332
SUBROUTINE	CØPY	333
SUBROUTINE	CRMULT	334
SUBROUTINE	EDGUNV	402
SUBROUTINE	EIGSØL	404
SUBROUTINE	ELMHES	Ref. 16
SUBROUTINE	ELTRAN	Ref. 16
SUBROUTINE	EUTL9	Ref. 6
SUBROUTINE	FIFDEF	406
SUBROUTINE	FØRMAT	409

TABLE E. INDEX TO NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

TABLE E. INDEX TO NONLINEAR INCRMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

MAIN PRØGRAM RESPNS

This is the executive routine for the Nonlinear Incremental Solution program.

Algorithm

This routine controls the incremental solution process including the iterations necessary to determine the modal response. Routine NITIAL is called first to establish partitions of array A in blank common as required by problem size. Routine ØUTPUT is then called to transcribe certain master input data to the master output file.

A loop is then entered the index of which is the number of increments defined in the card input data. Routines PREEIG, EIGSØL, FIFDEF, PREBAL, and IMBAL are the executive routines for the modules which make up the incremental solution. These modules are called in sequence for each increment within which routines EIGSØL and FIFDEF are iterated through 5 times. The variable controlling the number of iterations is NFFMAX which is initialized as 5.

The basic functions of each routine called during the incremental solution are

1) PREEIG - compute $\bar{M}$ in the first increment and compute $\bar{K}$, $\bar{C}$, and $\bar{P}$ in every increment
2) EIGSØL - assemble the A matrix, solve for all eigenvalues and eigenvectors, and compute the modal response, $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$
3) FIFDEF - temporarily update geometry and compute the fictitious force and deformation effects, s_k and t_k
4) PREBAL - permanently update geometry and regenerate $F_{\bar{F}}$ and $\sigma_{\bar{F}}$
5) IMBAL - compute element forces, stresses, and strains from the modal response and new geometry

Array DBLL is used to store the cumulative modal displacements, $\bar{\Delta}$, and is of length NG, the number of modes. This array is currently declared in a DIMENSION statement and is of length 75. A partition of length NG for storing $\bar{\Delta}$ should be allocated in the blank common array A by routine NITIAL rather than using a dimensioned array.

Array A (IVBX) is used to store the modal loads, $\bar{P}$, which are acting at the beginning of the increment. Arrays A(ISK) and A(ITK) are used to store the equivalent modal loads due to fictitious force and deformation effects, respectively, which at computed by FIFDEF for each iteration.

Input/Output

None

Arguments

None

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

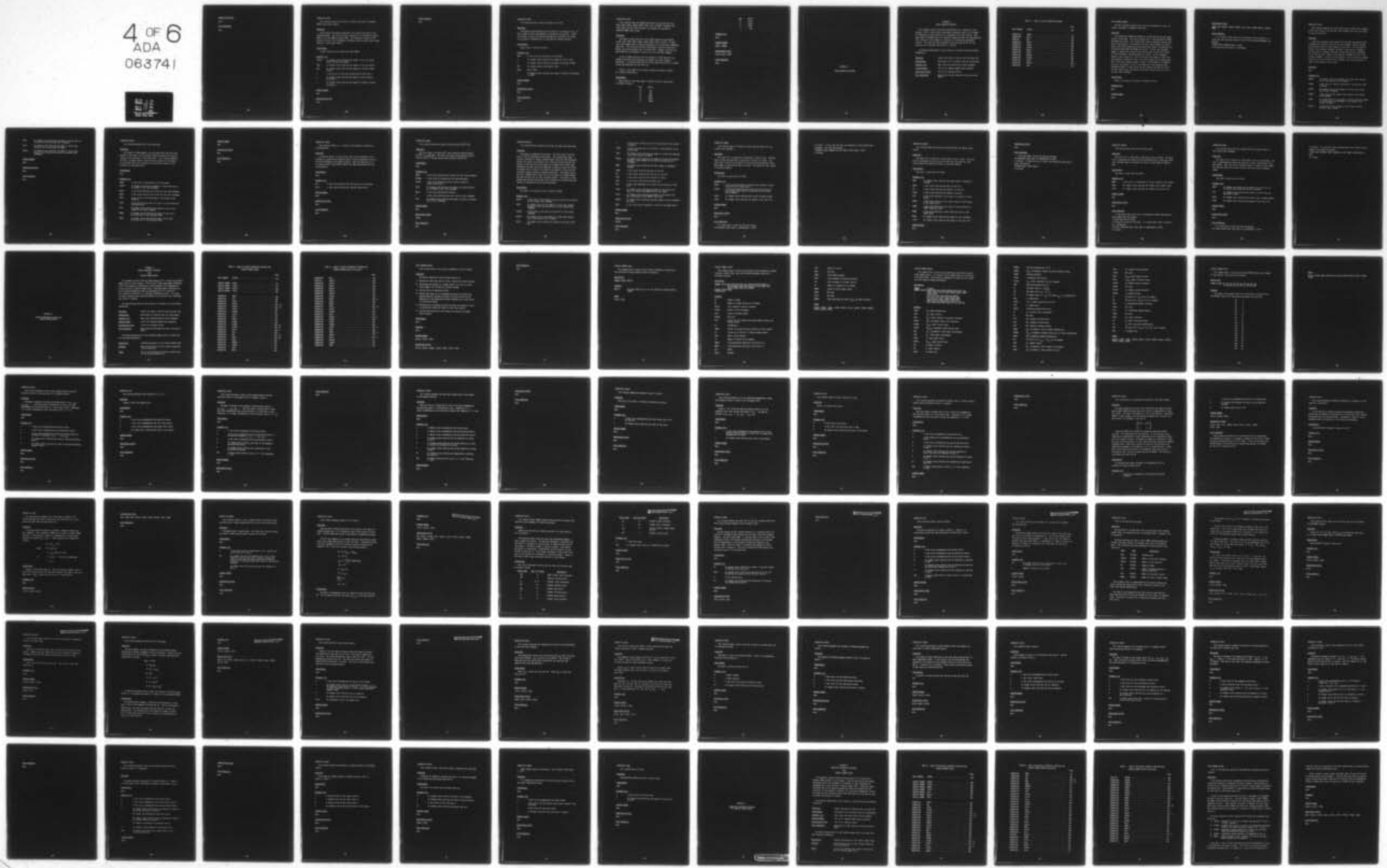
CØPY, EIGSOL, FIFDEF, IMBAL, NITIAL, ØUTPUT, PREBAL, PREEIG, ZERØ

Error Detection

None

AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3
AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)
DEC 77 R C MORRIS F33615-75-C-3105
UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

4 OF 6
ADA
063741



Subroutines Called

None

Error Detection

None

SUBROUTINE WPART

This routine writes a partition of a matrix onto tape in standard FØRMAT matrix data format.

Algorithm

Each column of the matrix partition A is written onto tape in the compressed mode. The origin of matrix partition A in the overall output matrix is at row IRØW and column ICØL. The row/column indices of the element of array A are incremented by IRØW and ICØL to obtain their proper location in the overall matrix.

Input/Output

A matrix partition is written onto tape TAPENØ.

Argument List

TAPENØ An integer scalar defining the number of the last column output logical tape number

IRØW An integer scalar defining the number of last row output

ICØL An integer scalar defining the number of the last column output

A A real array for the core resident matrix partition A

M An integer scalar defining the number of rows of matrix partition A

N An integer scalar defining the number of columns of matrix partition A

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WRMAT

This routine prints a matrix residing in the core.

Algorithm

The maximum column dimension of the matrix to be printed is 50. If MAP is equal to zero, the elements of the matrix are printed. If MAP is not zero, a map of the non-zero elements of the matrix is printed. This routine is executed only in runs made for program checkout.

Input/Output

Matrix data is printed on tape 6.

Argument List

AMAT	A real array for the matrix to be printed
M	An integer scalar defining the number of rows in AMAT
N	An integer scalar defining the number of columns of AMAT
MAP	An integer scalar print control flag
LABEL	Matrix label
L	An integer scalar defining the number of words (10 char/word) in LABEL

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WTAPE1

This routine reads the element partitions for matrices KEL, MEL, KBAR, CBAR, FFBAR, SIGFB, EPSIG, DEBT, EVT, and CØNST, assembles the partitions into the final matrices, and outputs the matrices as standard FØRMAT matrix data.

Algorithm

The order in which the first nine output matrices are processed is KEL, MEL, KBAR, CBAR, FFBAR, SIGFB, DPSIG, DEBT, and EVT. In each case, the element data is read and processed in the order bars, membranes, and cells. The only matrix having contributions from point mass elements, which are processed last, is matrix MEL. However, trailing null partitions are also present in matrix KEL for point mass elements.

In all of these matrices with the exception of DEBT and EVT, the element partitions are positioned as diagonal or psuedo diagonal partitions. In matrices DEBT and EVT, each element partition is a single column each beginning in the first row.

Finally, array CØNST which contains problem constants is output as a single column matrix.

Input/Output

Matrix data is read from tapes 7 through 15 and all output data is written to tape 1.

Tape	Matrix
14	KEL
10	MEL
13	KBAR
8	CBAR
12	FFBAR

Tape	Matrix
15	SIGFB
11	EPSIG
9	DEBT
7	EVT

Argument List

None

Labeled Common

CØNST, IERRØR

Subroutines Called

MATHD, MATTR, WPART

Error Detection

None

APPENDIX C

LOADS GENERATOR ROUTINES

APPENDIX C
LOADS GENERATOR ROUTINES

This appendix contains detailed descriptions of all routines in this program. Table C gives either page number references within this document or references to other documents for documentation of each routine. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

TABLE C. INDEX TO LOADS GENERATOR ROUTINES

MAIN PRØGRAM LØDGEN

The main program allocates core and calls subroutines to read, calculate, and output incremental loads data.

Algorithm

The subroutines READCØ and READCN are called and the average impact force is calculated. The load increment loop is then entered and READB and READJ are called. The current total load on the model and the location of the center of its footprint for this load increment are calculated, the largest component of the centroidal vector is determined, and the subroutines GENAB and MATMUL are called. If the load increment number is greater than one and less than or equal to the total number of load increments, subroutine DELTA is called. If the increment number is one, there is no need to calculate the change in loads from the preceding increment to this, since this change is the current load, and if the load increment number is equal to the number of increments plus one, the change in load is the negative of the preceding load. Subroutine ØUTPUT is then called and the current loads and the joints of their applications are stored at the end of the A array for use in the next load increment.

Input/Output

There is no input; all output is printed on file 6.

Argument List

None

Labeled Common

None

Subroutines Called

CRØSS, DØT, MATMUL, READB, READCØ, UVEC, DELTA, GENAB, ØUTPUT, READCN, READJ

Error Detection

If, in either of the checks on utilization of the A array, it is found that the available storage in the A array has been exceeded, the message
'INSUFFICIENT STORAGE nnnnn VS nnnnn'
is printed and the program is terminated.

SUBROUTINE DELTA

This routine compares the joint loads of the current load increment to the joint loads of the previous load increment and finds the change in load at each joint.

Algorithm

Since only non-zero joint loads for the current and previous load increments are stored in order of the joint numbers, there are three basic possibilities for each change in joint load. (1) The joint is loaded in the current increment, but not in the previous increment; (2) the joint is loaded in both the current and previous increments; or (3) the joint was loaded in the previous increment but not in the current increment. In any case the change in load on the joints loaded currently or in the previous load increments, are found and stored in order of the numbers of the joints.

Input/Out

None

Argument List

JØINTØ An integer array of the numbers of joints that received load in the previous load increment

PHIØLD A real array of loads on the joints in the previous load increment

JØINTN An integer array of the numbers of joints that receive load in this increment

PHINEW A real array of the loads on the joints in the current load increment

JTDEL An integer array of the numbers of joints that were loaded in the previous load increment or are loaded in the current increment

DELPHI A real array of the changes in joint loads from the previous load increment

NJTØ An integer scalar defining the number of joints that received load in the previous load increment

NJTN An integer scalar defining the number of joints that receive load in the current increment

NDELPH An integer scalar defining the number of joints that receive load in either the current or previous load increments

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE GENAB

This routine generates the A and B matrices.

Algorithm

The B matrix's three members are calculated first from the cross product of the vector to the current footprint location and the unit vector in the direction of load application. The A array (dimension 3X NJTN) is created second, using the cross products of the vectors to the joints that are to receive load and the unit vector in the direction of load application.

Input/Output

None

Argument List

CØØRJ A real array of coordinates of all the joints

JØINTN An integer array defining numbers of joints that are to be loaded in this load increment

ABETA A real array defining the A array for this load increment

BBETA A real array defining the B array for this load increment

CBETA A real array of the coordinates of the center of the footprint

UVAPP A real array defining the unit vector in the direction of load application

NJTN An integer scalar defining the number of joints to be loaded in this load increment

IRØW2 An integer scalar defining the number of one of the principal axes in the reference plane

IRØW3 An integer scalar defining the number of the other principal axis in the reference plane

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE INVERT

This routine inverts a 3 x 3 matrix by the method of successive transformations.

Algorithm

An identity matrix is set up first, then, while the matrix to be inverted is turned into an identity matrix, the row operations to do so are also performed on the identity matrix. When the input matrix is an identity matrix, the identity matrix has been transformed into the inverse matrix.

Input/Output

None

Argument List

AAT A real array defining the input matrix to be inverted

AATIN A real array defining the inverted output matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MATMUL

This routine performs the matrix multiplication $A^T(AA^T)^{-1}BF$.

Algorithm

The first step is to obtain AA^T, then the matrix multiplication $A^T(AA^T)^{-1}$ is performed. The final two steps are the matrix multiplication $A^T(AA^T)^{-1}B$ and the scalar multiplication $A^T(AA^T)^{-1}BF$.

Input/Output

None

Argument List

ABETA A real array defining the A matrix for this load increment

PREPRØ A real array of storage area for the pre-product

PHINEW A real array defining the final matrix product of loads on the joints

JØINT An integer array defining the numbers of joints that are to be loaded in this load increment

BBETA A real array defining the B matrix

F A real scalar defining the total load for this increment

NJTN An integer scalar defining the number of joints to receive load in this increment

Labeled Common

None

Subroutines Called

INVERT

Error Detection

None

SUBROUTINE ØUTPUT

This routine prints, punches and writes (on tape) the output data.

Algorithm

The output is produced in two forms. The first form is for the user to use for checking the input data. The constant data (mass velocity, duration of impact, average impact force, and the load application unit vector) are printed first, then the current load and the change in load that occurred between the preceding and current load increments for each joint that undergoes a change in load are printed. The second form of output is printed for the benefit of the user and written on tape for analysis programs downstream. The format used is that for a FØRMAT matrix whose rows are the degree of freedom numbers and whose columns are the load increment numbers. The change in load between the previous and current load increments are divided along the degrees of freedom and are written accordingly.

Input/Output

The output is written on files 6, JTAPE and KTAPE.

Argument List

DELPHI A real array of the change in joint loads from the previous to the current load increment

JØINT An integer array of the numbers of joints that undergo changes in load from the previous to the current load increment

PHINEW A real array of the loads on the joints in the current increment

JØINTN An integer array of the numbers of joints that receive load in the current load increment

IBETA An integer scalar defining the number of the load increment

F A real scalar defining the total load applied in the current increment

UVAPP A real array defining a unit vector in the direction of the applied load

NJTN An integer scalar defining the number of joints that received load in the current increment

NDELPH An integer scalar defining the number of joints that undergo changes in load between the current and preceding load increments

NDØF An integer scalar defining the total number of degrees of freedom

AMASS A real scalar defining the mass of the bird

VEL A real scalar defining the velocity of the bird

TIME A real scalar defining the duration of impact

FAVG A real scalar defining the average load

UVEL A real array defining a unit vector of the velocity of the bird

JTAPE An integer scalar defining the number of the output file that will contain the FØRMAT matrix card images

KTAPE An integer scalar defining the number of the output file that corresponds to the FØRMAT tape output

NBETAP An integer scalar defining the total number of load increments plus one

CØLM A real array used to assemble a column of the output matrix

Labeled Common

None

Subroutines Called

SQUEEZ

Error Detection

None

SUBROUTINE READB

This routine reads the footprint travel and load factor for the current load increment.

Algorithm

The input file is read until a Data Code 11 card is read. The Data Code 11 cards are read until a card with a flag of 6 is found. The cards with a flag of 6 are read until the card with the current increment number is read and the data is stored. The subroutine then returns to the main program.

Input/Output

The input is read from file ITAPE.

Argument List

DBETA A real scalar defining the distance the footprint travels from the initial impact point

FACTOR A real scalar defining the current position portion of the average impact load to be applied for the current load increment

IBETA An integer scalar defining the current increment number

ITAPE An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 11 cards are read the message 'NO CONSTANTS (DATA CODE 11) ENCOUNTERED IN INPUT'

is printed. If the card with data corresponding to the current load increment is not read the message
'INCREMENT (BETA) NUMBER nnn NOT FOUND IN DATA CODE 11 DATA'
is printed.

SUBROUTINE READCN

This routine reads the constants associated with the impact distribution.

Algorithm

The input file is read until a Data Code 11 card is read. The input file is backspaced one record and five cards are read and their data stored before the subroutine returns to the main program.

Input/Output

The input is read from file ITAPE.

Argument List

NDØF	An integer scalar defining the total number of degrees of freedom
AMASS	A real scalar defining the mass of the bird
VEL	A real scalar defining the velocity of the bird
ALENG	A real scalar defining the length of the bird
UVEL	A real array defining a unit vector in direction of bird motion
UNØRM	A real array defining a unit vector normal to the surface at the impact point
UFØØT	A real array defining a unit vector in the direction of impact footprint travel
CØØRI	A real array defining a vector from the origin to the impact point
NBETA	An integer scalar defining the number of load increments
ITAPE	An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 11 cards are read the message
'NO CONSTANTS (DATA CODE 11) ENCOUNTERED IN INPUT'
is printed. If there are not at least five consecutive Data Code 11 cards, the message
'MISSING DATA IN DATA CODE 11'
is printed.

SUBROUTINE READCØ

This routine reads the joint coordinate data.

Algorithm

The input file is read until a Data Code 2 card is found. The input file is backspaced one record and the coordinate data is read and stored until the data code is no longer 2 or until a joint number of 9999 is read.

Input/Output

The input is read from file ITAPE.

Argument List

CØØRJ	A real array of coordinates of joints stored by joint number
NJTØT	An integer scalar defining the largest joint number input
ITAPE	An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If the end of Data Code 2 data is encountered without encountering joint number 9999 the message
'A 29999 CARD WAS NOT FOUND'
is printed but execution continues. If no Data Code 2 data is found at all, the message
'NO JOINT COORDINATE DATA (DATE CODE 2) ENCOUNTERED IN INPUT'
is printed.

SUBROUTINE READJ

This routine reads the joint numbers that will receive load in the current load increment.

Algorithm

The input file is read until a Data Code 12 card is encountered. The file is backspaced one record and the data is read, checking the increment number. When the current increment number is read the joint numbers are read and stored for as many cards as have the current increment number.

Input/Output

The input is read from file ITAPE.

Argument List

JØINT	An integer array defining the numbers of joints that are to receive load during the current increment
NJTN	An integer scalar defining the number of joints that are contained in JØINT
IBETA	An integer scalar defining the current load increment number
ITAPE	An integer scalar defining the number of the input file

Labeled Common

None

Subroutines Called

None

Error Detection

If no Data Code 12 cards are read the message
'NO LOADED JOINTS DATA (DATA CODE 12) ENCOUNTERED IN INPUT'

is printed. If a card with data corresponding to the current load increment is not read the message

'JOINTS FOR INCREMENT (BETA) NUMBER nnnn NOT FOUND IN DATA CODE 12 DATA'

is printed.

APPENDIX D

LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

APPENDIX D
LINEAR INCREMENTAL ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table D gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatim source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description all external data set input/output
Argument List.	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usage	list of all routines which contain declarations for the labeled common block

TABLE D. INDEX TO LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

TABLE D. INDEX TO LINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

		Page
SUBROUTINE	MULT	348
FUNCTION	NDXSET	349
SUBROUTINE	NITIAL	350
SUBROUTINE	ØUTPUT	352
SUBROUTINE	PBARUF	353
SUBROUTINE	PREEIG	354
SUBROUTINE	PRINTR	356
SUBROUTINE	PTMASS	358
SUBROUTINE	READA	Ref. 5
SUBROUTINE	READK	359
SUBROUTINE	READM	360
SUBROUTINE	RGEIG	Ref. 16
SUBROUTINE	SBTRCT	361
SUBROUTINE	SIGFBR	362
SUBROUTINE	SMULT	363
SUBROUTINE	SQUEEZ	Ref. 5
SUBROUTINE	SYMABT	364
SUBROUTINE	SYMFIL	365
SUBROUTINE	SYMSØL	366
SUBROUTINE	TIMEQ	Ref. 6
SUBROUTINE	TMULT	368
SUBROUTINE	TSETQ	Ref. 6
SUBROUTINE	VDMULT	370
SUBROUTINE	WRITE	Ref. 5
SUBROUTINE	WRITEM	371
SUBROUTINE	XPØNNT	372
SUBROUTINE	ZERØ	373

MAIN PROGRAM RESPNS

This is the driver of the Linear Incremental Solution program.

Algorithm

The driver functions of this program consist of:

1) Defining 23 data sets used for input, output and scratch storage.

2) Delimiting the extents of 4 common blocks, the first of which, blank common, will be used for working storage.

3) Initializing via subroutine NITIAL.

4) Setting the stage for an incremental analysis by invoking subroutine PREEIG to generate matrices $\overline{M}$, $\overline{K}$ and $\overline{C}$ which are the components of the eigenvalue problem statement set up and solved via subroutine EIGSOL.

5) Performing the incremental analysis through the sequence of calls to subroutines LINEAR and IMBAL for each time interval.

6) Clocking the execution of each module and displaying elapsed time on output.

Input/Output.

None

Arguments

None

Labeled Common

NDICES, TAPES, LIMITS

Subroutines Called

NITIAL, PREEIG, EIGSØL, LINEAR, IMBAL, TSETQ, TIMEQ

Error Detection

None

LABELED COMMON CLØCK

This common block is used to store timing information to obtain CPU time required for major modules within the program.

Declaration

```
CØMMØN /CLØCK/ ØRGTIM
```

Contents

ØRGTIM	CPU time remaining for the run obtained by system routine TIMREM

Usage

TIMEQ, TSETQ

LABELED COMMON LIMITS

This common block is used to store problem size information, problem constants, option flags, and other program parameters used by all principle routines.

Declaration

```
CØMMØN /LIMITS/NG,NK,NELEMS,NTRVLS,BETA,BETBAR,HEAT,NM,IØRGN,JX
,              KØLX,NF,MØREK,MØREC,NG2,NG2SQ,NJTS,AHAT,DEBCL,TBM1
,              TAU,NS,NØWRK,NSUPD,NFAIL,NØPBPU
LOGICAL BETBAR,NØBPU
INTEGER BETA
```

Contents

NG	Number of modes
NK	Number of lumped forces for an element
NELEMS	Total number of physical elements
NTRVLS	Number of time increments
BETA	Current increment number
BETBAR	Not used
HEAT	Option flag for element dissipated damping energy converted to heat
NM	(NG*NG+NG)/2
IØRGN	Pointer to unused trailing partition of blank common
JX	Pointer to a location in labelled common NDICES
KØLX	Matrix column counter
NF	Number of forces for an element
MØREK	Flag designating additional partitions of $\bar{k}$
MØREC	Flag designating additional partitions of $\bar{c}$
NG2	NG+NG
NG2SQ	NG2*NG2

NJTS	Number of joints
AHAT	Not used
DEBCL	Total damping energy
TBM1	Time increment of previous interval
TAU	Time increment of current interval
NS	Number of stresses for an element
NWØRK	Extent of blank common region
NSUPD	Not used
NFAIL	Not used
NØPBPU	Flag indicating the input $\delta\bar{P}_{(\phi)U}$ has been exhausted

Usage

RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR

LABELED COMMON NDICES

This common block is used to store pointers to partitions within blank common array A. All values in this common block are initialized in routine NITIAL. Partitions for elements are sized for cells which have the largest storage requirements.

Declaration

```
CØMMØN         A( 8000)
CØMMØN /NDICES/IVBDB,IVBB,IVBX,IDPBPU,IPBMUB,IDPPUB,IDBL,IVBL
,               IPBAR,IPBPHU,IKB,ICB,IMBAR,IMBARL,IDEBCL,ICØNST
,               ITIME,ISIGSS
,               IKBL,IZ,ICA,IQ,IVAL,IFTAU,IVEC,IEGSYS
,               IU,LU,IECT,IEVT,IMPT,IDEBO,IDFBKO,IDSEB
,               IFK,IFBK,IDEL,IDEDL,IFSFB,IFSFBB.IPBCU
,               IPBKU,ISIGFB,ISIGBH,IEPSIG,IPSLØN,ID,IDK
,               ISKB,ISKBB,ISCB,ISCBB,IPBUF,ISK,ITK,ICIB,IMEL
```

Contents

IVBDB	$\dot{\bar{v}}_{\beta}$, modal accelerations
IVBB	$\bar{v}_{\beta}$, modal velocity
IVBX	$\bar{v}_{\beta-1}$, modal velocity of previous increment
IDPBPU	$\delta\bar{P}_{U}$, incremental modal force imbalance
IPBMUB	$\bar{P}_{(M)U}$, modal inertia force
IDPPUB	$\delta\bar{P}_{(\phi)U}$, incremental modal applied load
IDBL	$\delta\bar{\Delta}_{L}$, incremental linear modal displacement
IVBL	$\bar{v}_{L}$, linear modal displacement
IPBAR	$\bar{P}$, total modal forces
IPBPHU	$\bar{P}_{(\phi)U}$, modal applied load
IKB	$\bar{K}$, modal stiffness
ICB	$\bar{C}$, modal damping
IMBAR	$\bar{M}$, modal mass

IMBARL	Cholesky decomposition of $\bar{M}$
IDEBCL	$\delta\bar{E}_{CL}$, incremental element dissipated damping energy
ICØNST	Problem constants
ITIME	Incremental time history
ISIGSS	$\sigma_{S\sigma}$, stress transform for cell elements
IKBL	Cholesky decomposition of $\bar{K}$
IZ	The modal matrix $Z = \bar{K}^{-1}_{\beta-1}\bar{P}_{\beta}$
ICA	The modal matrix $C_a = H^{-1}$ yo
IQ	The modal matrix $Q = C_{a_D} F(\tau)$ where C_{a_D} is C_a diagonalized
IVAL	λ, eigenvalues
IFTAU	$F(\tau)$, a modal column matrix of $e^{\lambda\tau}$
IVEC	H, eigenvectors
IEGSYS	Eigenvalue problem work array
IU	U_o, original joint coordinates
LU	Not used
IECT	ECT, element constant table
IEVT	EVT, element variable table
IMPT	MPT, material property tables
IDEBO	$\delta\bar{e}_o$, incremental initial element deformations
IDFBKO	$\sigma\bar{F}_{K_o}$, incremental element forces due to initial deformations
IDSEB	$\delta\bar{e}$, incremental element deformations
IFK	The matrix $FK = \bar{F}_{K_{\beta-1}} + \delta\bar{F}_{K_o}$ for an element
IFBK	$\bar{F}_K$, element forces
IDEL	δe_L, incremental linear element displacement
IDEDL	$\delta\dot{e}_L$, incremental linear element velocity

IFSFB	$F_{\bar{F}}$, element force transform
IFSFBB	Not used
IPBCU	$\bar{P}_{(C)U}$, modal damping forces
IPBKU	$\bar{P}_{(K)U}$, modal stiffness forces
ISIGFB	$\sigma_{\bar{F}}$, element stress transform
ISIGBH	Not used
IEPSIG	ε_{σ}, element strain transform
IPSLØN	ε, element strains
ID	The matrix $D = \bar{P}_{UF} F_{\bar{F}}$ for an element
IDK	The matrix $DK = \bar{P}_{UF} F_{\bar{F}} \bar{k}$ for an element
ISKB	$\bar{k}$, unassembled element stiffness
ISKBB	Not used
ISCB	$\bar{c}$, unassembled element damping
ISCBB	Not used
IPBUF	$\bar{P}_{UF}$, modal transform
ISK	s_k, modal fictitious forces
ITK	t_k, modal fictitious deformations
ICIB	The matrix $\overline{CI} = \bar{P}_{UF} F_{\bar{F}} \bar{c} F_F^T \bar{P}_{UF}^T$ for an element
IMEL	m, element mass

<u>Usage</u>

RESPNS, EIGSØL, IMBAL, LINEAR, NDXSET, NITIAL, ØUTPUT, PBARUF, PREEIG, PTMASS, READK, SIGFBR

LABELED COMMON TAPES

This common block is used to store the FORTRAN logical unit numbers of the external files used by the program.

Declaration

```
COMMON /TAPES /N1 ,N2 ,N3 ,N4 ,N5 ,N6 ,N7 ,N8 ,N9 ,N10
,              N11,N12,N13,N14,N15,N16,N17,N18,N19,N20
```

Contents

The values in the common block are initialized in routine NITIAL. The FORTRAN logical unit designations assigned are given here.

N1	1
N2	2
N3	3
N4	4
N5	8
N6	9
N7	10
N8	11
N9	12
N10	13
N11	14
N12	15
N13	16
N14	17
N15	18
N16	19
N17	20
N18	21
N19	22
N20	23

Usage

RESPNS, EIGSØL,IMBAL,LINEAR,NITIAL,ØUTPUT,PBARUF,PREEIG,PTMASS, READK, SIGFBR

SUBROUTINE ABSYM

This routine performs a matrix cross-product wherein the post-multiplier matrix is the upper half of a symmetric matrix.

Algorithm

The mode of storage of the post-multiplier matrix is b_{11}, b_{12} ··· b_{1n}, b_{22} ····. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix sum $C = C + A \cdot B$.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix in the mode of storage b_{11}, b_{12}, ···, b_{1n}, b_{22} ···

M An integer scalar defining the order of the pre-multiplier matrix

N An integer scalar defining the order of the post-multiplier symmetric matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ADD

This routine performs vector addition; C = A + B.

Algorithm

Vectors A and B are summed into C.

Input/Output

None

Argument List

C	A real array accommodating the resulting vector
A	A real array accommodating the first input vector
B	A real array accommodating the second input vector
N	An integer scalar defining the order of the vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ASYMB

This routine performs a matrix cross-product wherein the pre-multiplier matrix is the upper half of a symmetric matrix.

Algorithm

The mode of storage of the symmetric pre-multiplier matrix is $a_{11}, a_{12}, \cdots, a_{1n}, a_{22} \cdots$. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix sum $C = C + A \cdot B$.

Input/Output

None

Argument List

C — A real array accommodating the product matrix

A — A real array accommodating the pre-multiplier matrix in the mode of storage $a_{11}, a_{12}, \cdots, a_{1n}, a_{22} \cdots$

B — A real array accommodating the post-multiplier matrix

M — An integer scalar defining the order of the symmetric pre-multiplier matrix

P — An integer scalar defining the column order of the pre-multiplier matrix

SUM — A logical scalar which if true, $C = C + A \cdot B$; otherwise $C = A \cdot B$

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CCMULT

This routine computes the specified leading rows of the product of two complex matrices.

Algorithm

When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix product and summation C = C + A*B.

Input/Output

None

Argument List

C	A complex array accommodating the product matrix
A	A complex array accommodating the pre-multiplier matrix
B	A complex array accommodating the post-multiplier matrix
M	An integer scalar defining the row dimension of arrays C and A
N	An integer scalar defining the column dimension of array A, and the row dimension of array B
P	An integer scalar defining the column dimension of arrays B and C
ML	An integer scalar defining the computations concerning ML rows of C
SUM	A logical scalar which if true, C = C + A*B; otherwise C = A*B

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CHGSGN

This routine changes the algebraic sign of a vector.

Algorithm

The sign of the vector is reversed by being set to minus.

Input/Output

None

Argument List

A	A real array accommodating the vector whose sign is to be reversed
N	An integer scalar defining the order of the vector

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CHLSKY

This routine performs an in situ Cholesky decomposition, where the symmetric matrix is given in its triangular half.

Algorithm

Where $A = LL^T$, given the positive definite matrix A in its symmetric half, this routine transforms it to L. The mode of storage is $a_{11}, a_{12}, \cdots a_{1n}, a_{22} \cdots a_{2n}$, etc.

Input/Output

None

Argument List

A	A real array accommodating the symmetric half of the positive definite matrix on input, and its square root decomposition on output
N	An integer scalar defining the order of the problem

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE COPY

This routine copies an array from core to core.

Algorithm

Array A is copied into array B.

Input/Output

None

Argument List

A	A real array to be copied
B	A real array into which the copy is made
N	An integer scalar defining the order of the arrays

Labeled Common

None

Subroutined Called

None

Error Detection

None

SUBROUTINE CRMULT

This routine computes the specified leading rows of a matrix product wherein the pre-multiplier matrix is complex.

Algorithm

When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero. Otherwise, it is simply incremented to yield the matrix product and summation C = C + A*B.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real complex array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the row dimension of arrays C and A

N An integer scalar defining the column dimension of array A and the row dimension of array B

P An integer scalar defining the column dimension of arrays B and C

ML An integer scalar defining the computations concerning ML rows of C

SUM A logical scalar which if true, C = C + A*B; otherwise, C = A*B

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE EIGSOL

This routine sets up, executes and disposes of the eigen problem.

Algorithm

The upper quadrants of array A are filled with the negative of symmetric modal damping and mass matrices, $\bar{C}$ and $\bar{M}$, reconstructed in their full forms. The lower left quadrant is set to the identity. Subroutine SYMSOL is then invoked, with the Cholesky decomposition of modal stiffness matrix $\bar{K}$ as argument, to arrive at:

$$\begin{bmatrix} -\bar{K}^{-1}\bar{C} & -\bar{K}^{-1}\bar{M} \\ I & 0 \end{bmatrix}$$

Subroutine RGEIG now performs the complete eigenvalue problem solution. The eigenvalues are reciprocated. In this process, equal eigenvalues corresponding to equal eigenvectors are recognized as being part of a singular system. An attempt to remedy this condition is made by scalar multiplying matrix $\bar{C}$ with 1.0001 and going through the complete procedure anew. Should this instability persist, the printing of an appropriate statement is followed by a call to subroutine HALT. Otherwise, the writing of the matrix of eigenvectors onto data set N8 is followed by their inversion through subroutine JØRDAN. The inverse is also output onto data set N8.

Input/Output

To conserve core space, the matrix of eigenvectors and its inverse are stored on data set N8.

Argument List

B	A real array accommodating the eigenvalue problem statement

X — A real array accommodating the matrix of eigenvectors

M — An integer which defines the order of the eigenvalue problem

N — An integer whose value is M/2

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

CHLSKY, CØPY, HALT, JØRDAN, RGEIG, SMULT, SYMFIL, SYMSØL, WRITE, ZERØ

Error Detection

Equal complex eigenvectors with corresponding equal eigenvalues are detected as singular. An attempt to remedy this condition is made by scalar multiplying matrix $\overline{C}$ with 1.0001. Should the unstable condition persist, the printing of an appropriate message is followed by the call to subroutine HALT.

SUBROUTINE HALT

This routine performs termination processing as necessary in the event of a fatal error.

Algorithm

In anticipation of certain processing requirements before job termination initiated by the detection of a fatal error, this routine and appropriate calls were provided. Subsequently, however, it was determined that no processing of this nature was necessary.

Input/Output

A job termination message is output to unit 6.

Argument List

None

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE IMBAL

For each physical element within each type of element, this routine computes the force, stress and strain matrices, $\overline{F}_K$, σ and ε, and writes them onto the output data set.

Algorithm

The output matrices concern bar elements, membrane elements and cell elements. Their respective FØRMAT matrix names are BARS, MEMBRN and CELLS. Each physical element is represented by a column whose contents are vectors $\overline{F}_K$, σ and ε. These quantities are computed as follows:

$$\overline{F}_K = \delta\overline{F}_{K_0} - DK^T \overline{\Delta}$$

where $$DK = \overline{P}_{UF} \, F_{\overline{F}} \, \overline{k}$$

$$\sigma = \sigma_{S\sigma} \, \sigma_{\overline{F}} \, \overline{F}_K \text{ for cells}$$

or $$\sigma = \sigma_{\overline{F}} \, \overline{F}_K \quad \text{for bars and membranes}$$

$$\varepsilon = \varepsilon_\sigma \, \sigma$$

Inout/Outout

Tape 2 is the output data set. For each physical element, matrix $\sigma_{\overline{F}}$ is read from data set N14; the matrix quantity $\overline{P}_{UF} \, F_{\overline{F}}\overline{k}$ is read from data set N7; $\delta\overline{F}_{Ko}$ is read from N15; and $\varepsilon\sigma$ is read from N16.

Araument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ADD, ASYMB, MULT, ØUTPUT, READA, READM, SBTRCT, TMULT, ZERØ

Error Detection

None

SUBROUTINE JØRDAN

This routine inverts in situ a complex matrix by pivoting on the available row of largest magnitude within the next available column.

Algorithm

The method used is Gauss-Jordan. The array KØL is used to record the order of rows which have been selected for pivoting.

Input/Output

None

Argument List

A — A real matrix which is transformed to its inverse prior to exit from this routine

KØL — An integer work array of dimension 2*N. Prior to exit from this routine, the first N locations contain the order in which rows have been selected for pivoting. The second N locations contain the values of the pivoting elements.

N — An integer scalar defining the order and dimension of matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE LINEAR

This routine computes vectors $\bar{P}$, $\delta\bar{\Delta}$, $\bar{v}$ and $\dot{\bar{v}}$.

Algorithm

Data set N8 is already positioned at the origin of the matrix of inverse eigenvectors. It is rewound as soon as this matrix has been read. The next read then brings in the matrix of eigenvectors proper.

$\delta\bar{\Delta}$ and $\bar{v}$ are computed in the complex mode and transferred to their final blank common destinations in the real mode. Through abstraction like CALL statements to utility routines, the matrix equations to compute $\bar{P}$, $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$ are solved in the following manner:

$$\bar{P} = \bar{P} + \bar{P}_{(M)U} + \delta\bar{P}_{(\phi)U}$$

$$Z = -\bar{K}^{-1}\,\bar{P}$$

$$C_{aD} = H^{-1}\left[\begin{array}{c} Z \\ \hline v \end{array}\right] \text{ diagonalized}$$

$$F(\tau) = e^{\lambda\tau}$$

$$Q = C_{ad}\;F(\tau)$$

$$\dot{\bar{v}} = H_v\;\lambda_D\;Q$$

$$\left[\begin{array}{c} \delta\bar{\Delta} \\ \hline \bar{v} \end{array}\right] = HQ$$

$$\delta\bar{\Delta} \quad \delta\bar{\Delta} - Z$$

Input/Output

The matrix of eigenvectors and its inverse are read from data set N8. The incremental applied load matrix $\delta\bar{P}_{(\phi)U}$ is from data set N18.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ADD, CCMULT, CHGSGN, CØPY, CRMULT, EUTL9, PRINTR, READA, SYMSØL, VDMULT, XPØNNT, ZERØ

Error Detection

None

SUBROUTINE MATIN

This routine routes FØRMAT Program input matrices and arrays from tape 20 to the intended, or interim, data sets.

Algorithm

Reading over the data set header positions it at the origin of its first matrix.

The following takes place with each item sandwiched between a matrix header and trailer. The data set designation, onto which the next matrix in sequence is to be written, is taken from the list NT and rewound. The matrix header is read and ignored. Each record thereafter is transcribed. Additionally, the third matrix is also written on data set 14. This process continues as long as the first word of the records thus processed is a positive integer. Except in the case of matrix $\overline{P}_{UF}$, the output data set is then rewound.

Input/Output

The list of throughput matrices and the data set onto which they are output follows:

Matrix Name	Data Set Number	Description
$\overline{P}_{UF}$	14	Modal element force transform
MPT	3	Material property table
U_o	8, 14	Original joint coordinates
ECT	15	Element constant table
M_{EL}	13	Element mass matrix
$\overline{k}$	11	Element stiffness matrix
$\overline{c}$	17	Element damping matrix
$F_{\overline{F}}$	4	Element force transform

Matrix Name	Data Set Number	Description
$\sigma_{\bar{F}}$	15	Element stress transform
ϵ_{σ}	22	Element strain transform
$\delta_{\bar{e}_T}$	10	Initial thermal element deformations
EVT	9	Element variable table

Argument List

A A real work array

NUM An integer scalar used as a dimensioning variable

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MBAR

This routine fetches the upper half of the next uncoupled partition from a tape resident symmetric quasi-diagonal matrix.

Algorithm

The end product of this routine consists of a triangular matrix of order NF in expanded format wherein the rows begin from their diagonal elements. As each column (KØLX + 1) through (KØLX + NF) is read in the compressed format, its elements preceding the diagonal are discarded. The row designations of the elements retained are reduced by the quantity KØLX - NF*(I-1), where I represents the sequence of these columns from 1 through NF. One call to subroutine EUTL9 expands this triangular matrix in its desired form.

Input/Output

None

Argument List

NF	An integer scalar defining the number of rows and columns in desired matrix partition
KØLX	An integer scalar defining the designation of the last column read on previous entry into this routine
A	A real storage array
NT	An integer scalar defining the designation of the data set containing the matrix

Labeled Common

None

Subroutines Called

EUTL9, READA, ZERØ

Error Detection

None

SUBROUTINE MULT

This routine performs a matrix product.

Algorithm

The matrix product A*B is added to matrix C. Matrix C is initialized to zero only when the logical variable SUM is false.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the row dimension of matrices C and A

N An integer scalar defining the row dimension of matrix B and the column dimension of matrix A

P An integer scalar defining the row dimension of matrices C and B

SUM A logical scalar which if false, array C is initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

FUNCTION NDXSET

This routine defines the elements of a vector used to segment the working core.

Algorithm

The variable IØRGN contains the relative location from which the work array is available for segmentation. The following takes place for each of the N apportionments: 1) the value of JX, the indexing pointer used with vector NDEX, is incremented; 2) IØRGN is stored into NDEX(JX); 3) IØRGN is increased by INC, the extent of core being allocated each origin.

Input/Output

None

Argument List

INC An integer scalar defining the extent of core to be allocated with each origin NDEX(I)

N Number of origins to be defined

Labeled Common

LIMITS, NDICES

Subroutines Called

None

Error Detection

None

SUBROUTINE NITIAL

This is the intializing routine.

Algorithm

The variables N1 through N20, which are contained within labeled common TAPES, are sequencially set to represent tapes 1 through 4, and 8 through 23.

The input matrices are read in their FØRMAT generated mode and written onto their interim, or intended, data sets via subroutine MATIN. The following delimiting parameters are read from the first input card.

Item	Type	Description
BETA	INTEGER	Starting time
NELEMS	INTEGER	Number of structural elements
NTRVLS	INTEGER	Number of time intervals
NG	INTEGER	Number of modes
HEAT	LOGICAL	"TRUE" indicates presence of thermal conditions
NJTS	INTEGER	Number of structural joints
NWØRK	INTEGER	Number of words in blank common

Blank common array A is apportioned to the various matrices and arrays through the designation of their respective origins by the index from labeled common NDICES.

The NTRVLS values concerning the time array are read into the space beginning with A(ITIME) and checked for their being in ascending order. Execution is halted with an appropriate statement when this check fails.

The constant matrix $\sigma_{S\sigma}$ for cell elements is generated and stored at A(ISIGSS).

Overlay within array A is effected by assigning IKBL and IU the same value. The first of these subscripts represent the origin of space used by subroutine EIGSØL only; the second is the leading subscript of non-conversant space used by other modules.

Subroutine READK is invoked to merge selected matrices residing on interim data sets. The arrays which are to accommodate the following vectors are initialized with zeroes: $\overline{v}$, $\dot{\overline{v}}$, $\overline{v}_{\beta-1}$, $\delta\overline{P}_{U}$, $\delta\overline{P}_{(M)U}$, $\delta\overline{P}_{(C)U}$, $\delta\overline{P}_{(\phi)U}$, and $\delta\overline{E}_{CL}$.

Input/Output

All card input takes place in this routine. In addition, the following input matrices are written onto the post-processing data set, tape 2: the array of constants under its input FØRMAT program matrix name; the array of time intervals under the matrix name TIME; the array of joint coordinates under the matrix name UZERØ; vectors $\overline{P}_{(\phi)U}$ and $\overline{P}_{UPT}$ under their input names.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

ASYMB, CHGSGN, MATIN, NDXSET, READA, READK, SIGFBR, SQRT, WRITE, ZERO

Error Detection

None

SUBROUTINE OUTPUT

This routine stores output onto the output data set in the mode of the FØRMAT program.

Algorithm

Vectors $\overline{\Delta}$, $\delta\overline{\Delta}$, $\overline{v}$, $\dot{\overline{v}}$, $\delta\overline{P}_U$, and $\overline{P}_{(\phi)U}$ are written onto data set 2 as a single column FØRMAT matrix with the name RESPNS.

Input/Output

Matrix RESPNS is output to data set 2.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

WRITE

Error Detection

None

SUBROUTINE PBARUF

This routine reads a partition of matrix $F_{\overline{F}}$ and the corresponding partition of matrix $\overline{P}_{UF}$.

Algorithm

Data set N13 contains matrix $F_{\overline{F}}$ in the form of one partition per record. Reading one such record provides the information necessary to determine the columns bounding the desired partition of $\overline{P}_{UF}$.

Input/Output

Matrix $F_{\overline{F}}$ is read from data set N13. Matrix $\overline{P}_{UF}$ is read from data set N11.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutine Called

READA, EUTLS

Error Detection

None

SUBROUTINE PREEIG

This routine generates matrices $\bar{M}$, $\bar{K}$, $\bar{C}$ and $\delta\bar{P}_{KO}$.

Algorithm

For each element, the matrix operations are performed through abstraction like CALL statements to utility routines. Having initialized matrices $\bar{K}$, $\bar{C}$, $\bar{M}$, $\bar{P}$, and $\bar{P}_{(M)U}$ as zero, the matrix operations performed are as follows:

$$\delta\bar{F}_{KO} = -\bar{K}\,\delta\bar{e}_T$$

$$D = \bar{P}_{UF}\,F_{\bar{F}}$$

$$\bar{P} = \bar{P} + D\,\delta\bar{F}_{KO}$$

$$DK = D\bar{k}$$

$$\bar{K} = \bar{K} + DK\,D^T$$

$$\bar{C} = C + D\bar{c}\,D^T$$

$$\bar{M} = \bar{M} + \bar{P}_{UF}\,m\,\bar{P}_{UF}{}^T$$

If point mass elements exist, their contribution to the modal mass matrix, $\bar{M}$, is computed and added by a single call to subroutine PTMASS.

Input/Output

For each physical element, a partition of the matrices $\bar{k}$, $\bar{c}$ and $\delta\bar{e}_T$ is read in that sequence from data set N1. Similar partitions of matrices $\bar{P}_{UF}$, $F_{\bar{F}}$, and m are read from data sets N11, N13 and N10, respectively. The contribution of each physical element to matrix DK is written onto data set N7 and, similarly, the contribution to δF_{KO} is written onto N15.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutine Called

ABSYM, ADD, ASYMB, CHGSGN, MBAR, MULT, PBARUF, PTMASS, READA, SYMABT, WRITEM, ZERØ

Error Detection

None

SUBROUTINE PRINTR

This routine prints a core resident matrix.

Algorithm

Groups of not more than 8 columns across each page are written for partitions of not more than 55 rows. Each page is headed with the title. The column designations appear under the title. The row designations precede each row. Only the most significant aspect of double precision matrices is written. Two calls to this routine are required to print a complex matrix.

Input/Output

None

Argument List

A	A real array accommodating the matrix to be printed
K	An integer scalar used as a dimensioning variable. K = 1 for real and single precision arrays. K = 2 for compressed or double precision arrays. K = 4 for double precision complex arrays.
M	An integer scalar defining the row dimension
N	An integer scalar defining the column dimension
TITLE	An alphanumeric scalar for header data

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE PTMASS

This routine concludes the computations of $\overline{M}$ with the contributions of the point-mass elements.

Algorithm

The computations concern the triple product $\overline{P}_{UF} M_{EL} \overline{P}_{UF}^{T}$ involving point mass elements. M_{EL} consists of one element. The columns of these matrices are read and the computations are exercised upon matching the column designations.

Input/Output

Matrix M_{EL} is read from data set N10. Matrix $\overline{P}_{UF}$ is read from data set N11.

Argument List

None

Labeled Common

LIMITS, NDICES, TAPES

Subroutines Called

READA, EUTL9, ABSYM, SYMABT

Error Detection

None

SUBROUTINE READK

This routine transcribes FØRMAT program generated matrices from interim data sets to their intended data sets.

Algorithm

The symmetric quasi-diagonal matrices $\overline{k}$, e_σ and $\overline{c}$ are read in their full forms. They are retained only in their symmetric halves prior to being written as single compressed records.

Matrix δe_T is simply copied from its input to its output tape. The quasi-diagonal matrix $F_{\overline{F}}$ is output in records of sub-matrix partitions.

Input/Output

Matrices $\overline{k}$, e_σ, $\overline{c}$, $\delta\overline{e}_T$, and $F_{\overline{F}}$ are respectively read from data sets N8, N19, N14, N10, and N4; matrices e_σ and $F_{\overline{F}}$ are written onto data sets N16 and N13. Matrices $\overline{k}$, $\overline{c}$ and $\delta\overline{e}_T$ are written onto data set N1 in the sequence $\overline{k}_1$, $\overline{c}_1$, $\delta\overline{e}_1$, $\overline{k}_2$, $\overline{c}_2$, $\delta\overline{e}_{T_2}$, ····, etc.

Argument List

None

Labeled Common

LIMITS, NOICES, TAPES

Subroutines Called

EUTL9, MBAR, READA, WRITE

Error Detection

None

SUBROUTINE READM

This routine reads a matrix which was written on a scratch data set via subroutine WRITEM.

Algorithm

The matrix is read via subroutine READA. Should it be compressed, subroutine EUTL9 expands it.

Input/Output

The matrix is read from data set NT.

Argument List

M	A dummy argument
N	A dummy argument
A	A real array into which the matrix is read
NT	An integer scalar defining the input data set

Labeled Common

None

Subroutine Called

None

Error Detection

None

SUBROUTINE SBTRCT

This routine computes the arithmetic difference between two vectors.

Algorithm

The algebraic difference between vectors A and B is stored in vector C.

Input/Output

None

Argument List

C	A real array for the resulting vector
A	A real array for the vector being subtracted
B	A real array for the subtracting vector
N	An integer scalar defining the order of vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SIGFBR

This routine partitions a quasi-diagonal matrix and outputs its partitions in single compressed records.

Algorithm

The columns of each partition are read and stored consecutively in their compressed format. In the process, the row designations are replaced by the location of the elements relative to the current submatrix. A single call to subroutine EUTL9 expands the partition which is then output by subroutine WRITEM.

Input/Output

The matrix is read from data set N14 and written onto data set N12.

Argument List

None

Labeled Common

TAPES, LIMITS, NDICES

Subroutines Called

EUTL9, READA, WRITEM

Error Detection

None

SUBROUTINE SMULT

This routine scales a matrix.

Algorithm

Each element of array A is multiplied by the scalar S, and the product is stored in array C.

Input/Output

None

Argument List

C	A real array accommodating the output matrix
S	A real scalar coefficient
A	A real array accommodating the matrix to be scaled
M	An integer scalar defining the row dimension
N	An integer scalar defining the column dimension

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMABT

This routine computes the triangular half of a symmetric matrix where the post-multiplier is transposed.

Algorithm

The mode of storage of the product matrix is $C_{11}\cdots\cdot C_{1n}$, $C_{22}\cdots C_{2n}$, etc. When the logical variable SUM is false, the array accommodating the resulting matrix is initialized as zero.

Input/Output

None

Argument List

C A real array for the triangular product matrix

A A real array for the pre-multiplier matrix

B A real array for the transposed post-multiplier matrix

M An integer scalar defining the row dimension of all matrices

N An integer scalar defining the column dimension of A and B matrices

SUM A logical scalar when false, causes the resulting matrix to be initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMFIL

This routine stores a symmetric half matrix into the partition of an array while reversing the sign.

Algorithm

The mode of storage of the symmetric half-matrix is a_{11}, ··· a_{1n}, a_{22} ····. When K = 1, the receiving matrix is real. When K = 2, it is complex. Note that in this case, the imaginary components are left untouched.

Input/Output

None

Argument List

A A real array for the symmetric half-matrix

B A real or complex array for the output matrix

K An integer scalar where K = 1 for real B array, K = 2 for complex B array

M An integer scalar defining the row dimension of B array

N An integer scalar defining the order of symmetric matrix

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE SYMSØL

Given matrices L and B in the expression $LL^TX = B$; this routine transforms B into X.

Algorithm

The mode of storage for matrix L is $l_{11}, \cdots, l_{n1}, l_{22}\cdots$. X is computed by way of Y where $LY = B$. Then, $L^TX = Y$. The array storing matrices B and X may be real or complex. When the integer K is 1, the array is real; when K is 2, the array is complex.

Input/Output

None

Argument List

A — A real array accommodating matrix L in the mode of storage $l_{11}\cdots l_{n1}, l_{22}\cdots$

B — A real or complex array accommodating matrices B, Y and X

K — An integer scalar where array B is real when K = 1, and complex when K = 2

M — An integer scalar defining the row dimension of array B

N — An integer scalar defining the order of equation

P — An integer scalar defining the number of columns in matrices B, Y and X

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE TMULT

This routine performs a matrix cross product wherein the premultiplier matrix is transposed.

Algorithm

The matrix product transpose $A^T B$ is added to matrix C. Matrix C is initialized to zero only when the logical variable SUM is false.

Input/Output

None

Argument List

C A real array accommodating the product matrix

A A real array accommodating the pre-multiplier matrix

B A real array accommodating the post-multiplier matrix

M An integer scalar defining the row dimension of matrix C and column dimension of matrix A

An integer row dimension of matrices A and B

N An integer scalar defining the row dimension of matrix C and column dimension of matrix A

An integer row dimension of matrices A and B

P An integer column dimension of matrices B and C

SUM A logical scalar when false causes array C to be initialized as zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE VDMULT

This routine performs the element by element product of two complex vectors.

Algorithm

The element by element product of complex vectors A and B is stored in vector C.

Input/Output

None

Argument List

C	A complex array for the output vector C
A	A complex array for the input vector A
B	A complex array for the input vector B
N	An integer scalar defining the order of the vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE WRITEM

This routine writes a one-record matrix, compressed (as required).

Algorithm

Following an attempt to compress the matrix via subroutine SQUEEZ, it is written onto the output data set NT.

Input/Output

The matrix is written onto the output data set.

Argument List

M	An integer scalar defining the matrix row dimension
N	An integer scalar defining the matrix column dimension
A	A real array for the input matrix
NT	An integer scalar defining the output data set

Labeled Common

None

Subroutines Called

SQUEEZ, WRITE

Error Detection

None

SUBROUTINE XPØNNT

Given complex vector D and vector T, this routine finds vector $A = e^{D*T}$.

Algorithm

All computations take place in the real mode as vectors A and D are doubly dimensioned arrays.

Input/Output

None

Argument List

A	A real array accommodating the output vector
D	A real array for the complex input vector treated in the real mode
T	A real array for the input vector
N	An integer scalar defining the order of vectors

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ZERØ

This routine zeros an array.

Algorithm

The specified extent of array A is set to zero.

Input/Output

None

Argument List

A — A real array to be set to zero

N — An integer scalar defining the extent of A to be set to zero

Labeled Common

None

Subroutines Called

None

Error Detection

None

APPENDIX E

NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

APPENDIX E
NONLINEAR INCREMENTAL ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table E gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usage	list of all routines which contain declarations for the labeled common block

TABLE E. INDEX TO NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS

TABLE E. INDEX TO NONLINEAR INCREMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

TABLE E. INDEX TO NONLINEAR INCRMENTAL ROUTINES AND LABELED COMMON BLOCKS (Continued)

		Page
FUNCTION	PØLSØL	438
SUBROUTINE	PREBAL	440
SUBROUTINE	PREBGM	443
SUBROUTINE	PREBGU	444
SUBROUTINE	PREBLL	446
SUBROUTINE	PREBST	447
SUBROUTINE	PREEIG	448
SUBROUTINE	PTMASS	358
SUBROUTINE	READA	Ref. 5
SUBROUTINE	READK	359
SUBROUTINE	READM	360
SUBROUTINE	RGEIG	Ref. 16
SUBROUTINE	SBTRCT	361
SUBROUTINE	SMULT	363
SUBROUTINE	SQUEEZ	Ref. 5
SUBROUTINE	SYMABT	364
SUBROUTINE	SYMFIL	365
SUBROUTINE	SYMSØL	366
SUBROUTINE	TMULT	368
SUBROUTINE	TRNMLT	451
SUBROUTINE	VDMULT	370
SUBROUTINE	VECT	284
SUBROUTINE	WRITE	Ref. 5
SUBROUTINE	WRITEM	371
SUBROUTINE	XPØNNT	372
SUBROUTINE	ZERØ	373

MAIN PRØGRAM RESPNS

This is the executive routine for the Nonlinear Incremental Solution program.

Algorithm

This routine controls the incremental solution process including the iterations necessary to determine the modal response. Routine NITIAL is called first to establish partitions of array A in blank common as required by problem size. Routine ØUTPUT is then called to transcribe certain master input data to the master output file.

A loop is then entered the index of which is the number of increments defined in the card input data. Routines PREEIG, EIGSØL, FIFDEF, PREBAL, and IMBAL are the executive routines for the modules which make up the incremental solution. These modules are called in sequence for each increment within which routines EIGSØL and FIFDEF are iterated through 5 times. The variable controlling the number of iterations is NFFMAX which is initialized as 5.

The basic functions of each routine called during the incremental solution are

1) PREEIG - compute $\bar{M}$ in the first increment and compute $\bar{K}$, $\bar{C}$, and $\bar{P}$ in every increment
2) EIGSØL - assemble the A matrix, solve for all eigenvalues and eigenvectors, and compute the modal response, $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$
3) FIFDEF - temporarily update geometry and compute the fictitious force and deformation effects, s_k and t_k
4) PREBAL - permanently update geometry and regenerate $F_{\bar{\bar{F}}}$ and $\sigma_{\bar{\bar{F}}}$
5) IMBAL - compute element forces, stresses, and strains from the modal response and new geometry

Array DBLL is used to store the cumulative modal displacements, $\bar{\Delta}$, and is of length NG, the number of modes. This array is currently declared in a DIMENSION statement and is of length 75. A partition of length NG for storing $\bar{\Delta}$ should be allocated in the blank common array A by routine NITIAL rather than using a dimensioned array.

Array A (IVBX) is used to store the modal loads, $\bar{P}$, which are acting at the beginning of the increment. Arrays A(ISK) and A(ITK) are used to store the equivalent modal loads due to fictitious force and deformation effects, respectively, which at computed by FIFDEF for each iteration.

Input/Output

None

Arguments

None

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

CØPY, EIGSOL, FIFDEF, IMBAL, NITIAL, ØUTPUT, PREBAL, PREEIG, ZERØ

Error Detection

None

AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3
AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)
DEC 77 R C MORRIS F33615-75-C-3105
UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

5 OF 6
ADA
063741

LABELED COMMON CRRAYS

This common block is used to store intermediate data during the generation of fictitious forces and deformations.

Declaration

```
CØMMØN /CRRAYS/ C(225)
DIMENSIØN RNK(8,3), RRK(30,2)
EQUIVALENCE (C(136),RNK(1,1)), (C(160),RRK(1,1))
```

Contents

C An array used to store intermediate data including the element coefficients n_k and r_k

Usage

AVRAGE, CHKØUT, FIFDEF, NKRK

LABELED COMMON FICNDX

This common block is used to store the first locations of partitions in blank common array A for use during the generation of fictitious forces and deformations.

Declaration

```
COMMON /FICNDEX/ IRSR,IGFB1,IGFB,IDOB1,IFBF,IREC,IRK,INK,
                 IDEBBL,ICØL,IRCØL,IFBFS,ITHK,
                 JNTS,LAST2,LAST3,ICEDG,ICUNV
```

Contents

IRSR	r_r', reordering transform for membrane and cell elements
IGFB1	$G_{f_{\beta-1}}$, element geometry at beginning of increment
IGFB	G_{f_β}, element geometry at end of interation
IDOB1	$D_{o_{\beta-1}}$, warp offset at $t_{\beta-1}$
IFBF	$\bar{F}_f$, element forces
IREC	ECT, element constant table
IRK	r_k, fictitious element deformations
INK	n_k, fictitious element forces
IDEBBL	$\delta\tilde{e}$, fictitious element displacements
ICØL	Number of columns in $F_{\overset{\equiv}{F}}$
IRCØL	Number of columns in r_r'
IFBFS	Not used
ITHK	Cell element differential thickness vectors
JNTS	Number of joints for the element
LAST2	Last location of IGFB1
LAST3	Last location of IDOB1
ICEDG	Not used

ICUNV Not used

Usage

CHKØUT, EDGUNV, FIFDEF, GEØM, NKRK

LABELED COMMON GEØMS

This common block is used to store element geometry during the generation of fictitious forces and deformations.

Declaration

```
CØMMØN /GEØMS / EDGV(4,8),UNV(3,6),THICK(4,3),CØØRD(8,3)
```

Contents

EDGV	An array of edge vectors components and magnitudes
UNV	An array of unit edge vectors
THICK	An array of thickness vectors for cell elements
CØØRD	An array for temporary storage of joint coordinates

Usage

EDGUNV, FIFDEF, GEØM, LPBFIC, LPCFIC, LPMFIC

LABELED COMMON LIMITS

This common block is used to store problem size information, problem constants, option flags, and other program parameters used by all principle routines.

Declaration

```
CØMMØN /LIMITS/NG,NK,NELEMS,NTRVLS,BETA,BETBAR,HEAT,NM,IØRGN,JX
,               KØLX,NF,MØREK,MØREC,NG2,NG2SQ,NJTS,AHAT,DEBCL,TBM1
,               TAU,NS,NØWRK,NSUPD,NFAIL
,  X4(10)
LOGICAL BETBAR
INTEGER BETA
```

Contents

NG	Number of modes
NK	Number of lumped forces for an element
NELEMS	Total number of physical elements
NTRVLS	Number of time increments
BETA	Current increment number
BETBAR	Material nonlinearity iteration flag
HEAT	Option flag for element dissipated damping energy converted to heat
NM	(NG*NG+NG)/2
IØRGN	Pointer to unused trailing partition of blank common
JX	Pointer to a location in labelled common NDICES
KØLX	Matrix column counter
NF	Number of forces for an element
MØREK	Flag designating additional partitions of $\bar{k}$
MØREC	Flag designating additional partitions of $\bar{c}$
NG2	NG+NG

NG2SQ	NG2*NG2
NJTS	Number of joints
AHAT	Geometric nonlinearity correction factor
DEBCL	Total damping energy
TBM1	Time increment of previous interval
TAU	Time increment of current interval
NS	Number of stresses for an element
NWØRK	Extent of blank common region
NSUPD	Element stiffness update flag
NFAIL	Element failure flag
X4	Pad for expansion of LIMITS

<u>Usage</u>

RESPNS, AVRAGE, CHKØUT, CHKPNT, EIGSØL, FIFDEF, FØRMAT, IMBAL, NDXSET, NITIAL, NKRK, ØUTPUT, PBARUF, PØLSØL, PREBAL, PREBGM, PREBGU, PREBLL, PREBST, PREEIG, PTMASS, READK

LABELED COMMON LPEM

This common block is used to store element geometry and other data for use by the $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, and $\sigma_{\bar{F}}$ matrix regeneration routines in the PREBAL module. The configuration of the trailing portion of this common block is different for each of the element types; bars, membranes, and cells.

Declaration (Routine PREBAL)

```
CØMMØN /LPEM/ NUMECT,IUPDAT,IFAILD,IPARLL,ITMP1,ITMP2,IHØLD
,  NFK,NSK,NKSYM,NSSYM,JM(30),XXX(600)
LØGICAL IUPDAT,IFAILD,IPARLL
```

Declaration (Bar Elements)

```
CØMMØN /LPEM/ NUMECT,IUPDAT,IFAILD,IPARLL,ITMP1,ITMP2,IHØLD
,  NFK,NSK,NKSYM,NSSYM,JBN(30),PQ(4),AB(4)
EQUIVALENCE (JBN(1),JP),(JBN(2),JQ),(JBN(3),AREA)
LØGICAL IUPDAT,IFAILD,IPARLL
```

Declaration (Membrane Elements)

```
CØMMØN /LPEM/ NUMECT,IUPDAT,IFAILD,IPARLL,ITMP1,ITMP2,IHØLD
,  NFK,NSK,NKSYM,NSSYM,JMN(30),
                AØTP   , AØTQ   , AØTR   , AØTS   , AB(4),
                BB(4)  , CB(4)  , DB(4)  ,
                DP(3,3), DQ(3,3), DR(3,3), DS(3,3), EB(4),
                FK1    , FK2    , FK3    , FK4    , FK5  ,
                PQ(4)  , RQ(4)  , RS(4)  , PS(4)  ,
                SP(3)  , SQ(3)  , SR(3)  , SS(3)  ,
                THETAP , THETAQ , THETAR , THETAS ,
                ZP(3,3), ZQ(3,3), ZR(3,3), ZS(3,3)
EQUIVALENCE (JMN(1),JP),(JMN(2),JQ),(JMN(3),JR),(JMN(4),JS),
            (JMN(5),ZETAPQ),(JMN(6),T)
LØGICAL IUPDAT,IFAILD,IPARLL
```

Declaration (Cell Elements)

```
CØMMØN /LPEM/ NUMECT,IUPDAT,IFAILD,IPARLL,ITMP1,ITMP2,IHØLD
,  NFK,NSK,NKSYM,NSSYM,JCN(30),
                    AB(4,8)   , BB(4,8)   , CB(4,8)   ,
                    DB(4,8)   , EB(4,8)   , FB(4,8)   , FK(5,8)  ,
                    LA(8)     , LB(8)     , LT(8)     ,
                    PQ(4,8)   , RQ(4,8)   , RS(4,8)   , PS(4,8)  ,
                    THETA(8)  , TPB(4)    , TQB(4)    , TRB(4)   ,
                    TSB(4)    , TPQ(4)    , TRQ(4)    , TRS(4)   ,
                    TPS(4)    , UN(4,6)   , V(8)      , ZETA(8)
EQUIVALENCE (JCN(1),JPO),(JCN(2),JQO),(JCN(3),JRO),(JCN(4),JSO),
            (JCN(5),JP1),(JCN(6),JQ1),(JCN(7),JR1),(JCN(8),JS1),
            (JCN(9),ZETAPQ)
EQUIVALENCE (JCN(11),SIG11),(JCN(12),SIG21),(JCN(13),SIG41),
            (JCN(14),SIG12),(JCN(15),SIG22),(JCN(16),SIG42),
            (JCN(17),SIG13),(JCN(18),SIG23),(JCN(19),SIG43),
            (JCN(20),SIG54),(JCN(21),SIG64),(JCN(22),SIG55),
            (JCN(23),SIG65),(JCN(24),SIG36)
LØGICAL IUPDAT,IFAILD,IPARLL
REAL LA,LB,LT
```

Contents (General)

NUMECT	Length of ECT record
IUPDAT	Stiffness update flag
IFAILD	Element failure flag
IPARLL	Parallelogram membrane flag
ITMP1	Temporary storage array of length 30
ITMP2	Temporary storage array of length 30
IHØLD	Temporary storage array of length 240
NFK	NF*NK (see Labeled Common LIMITS)
NSK	NS*NK (see Labeled Common LIMITS)
NKSYM	(NK*NK+NK)/2
NSSYM	(NS*NS+NS)/2

Contents (Bar Elements)

JBN(1)	Number of joint p
JBN(2)	Number of joint q
JBN(3)	Bar area

Contents (Membrane Elements)

JMN(1)	Number of joint p
JMN(2)	Number of joint q
JMN(3)	Number of joint r
JMN(4)	Number of joint s
JMN(5)	Stress orientation angle
JMN(6)	Membrane thickness
AØTP	A*B*SIN(THETA)/T for corner p
AØTQ	A*B*SIN(THETA)/T for corner q
AØTR	A*B*SIN(THETA)/T for corner r
AØTS	A*B*SIN(THETA)/T for corner s
AB	Unit vector components, X/T, Y/T, Z/T, T
BB	Unit vector components, X/T, Y/T, Z/T, T
CB	Unit vector components, X/T, Y/T, Z/T, T
DB	Unit vector components, X/T, Y/T, Z/T, T
DP	$\tilde{D}$ matrix for corner p
DQ	$\tilde{D}$ matrix for corner q
DR	$\tilde{D}$ matrix for corner r
DS	$\tilde{D}$ matrix for corner s
ER	Unit vector components, X/T, Y/T, Z/T, T
FK1	K factor 1
FK2	K factor 2
FK3	K factor 3
FK4	K factor 4
FK5	K factor 5
PQ	Edge vector components and magnitude, X, Y, Z, T
RQ	Edge vector components and magnitude, X, Y, Z, T
RS	Edge vector components and magnitude, X, Y, Z, T
PS	Edge vector components and magnitude, X, Y, Z, T
SP	Skew matrix for corner p
SQ	Skew matrix for corner q
SR	Skew matrix for corner r
SS	Skew matrix for corner s

THETAP	Corner angle
THETAQ	Corner angle
THETAR	Corner angle
THETAS	Corner angle
ZP	Global translation matrix for corner p
ZQ	Global translation matrix for corner q
ZR	Global translation matrix for corner r
ZS	Global translation matrix for corner s

<u>Contents (Cell Elements)</u>

JCN(1) to JCN(8)	Number of joints p_o, q_o, r_o, s_o, p_1, q_1, r_1, and s_1
JCN(9)	Stress orientation angle
JCN(11) to JCN(24)	Non zero elements of $\sigma_{\bar{\bar{F}}}$ where the numbering of SIG corresponds to location in a singularly dimensional array storing the matrix columnwise
AB	Unit vector components, X/T, Y/T, Z/T, T
BB	Unit vector components, X/T, Y/T, Z/T, T
CB	Unit vector components, X/T, Y/T, Z/T, T
DB	Unit vector components, X/T, Y/T, Z/T, T
EB	Unit vector components, X/T, Y/T, Z/T, T
FB	Unit vector components, X/T, Y/T, Z/T, T
FK	K factor
LA	Length of cell sub-element
LB	Width of cell sub-element
LT	Thickness of cell sub-element
PQ	Edge vector components and magnitudes, X, Y, Z, T
RQ	Edge vector components and magnitudes, X, Y, Z, T
RS	Edge vector components and magnitudes, X, Y, Z, T
PS	Edge vector components and magnitudes, X, Y, Z, T
THETA	Corner angles (THETAPO, THETAQO, THETARO, THETASO, THETAP1, THETAQ1, THETAR1, THETAS1)

TPB	Thickness vector components and magnitude
TQB	Thickness vector components and magnitude
TRB	Thickness vector components and magnitude
TSB	Thickness vector components and magnitude
TPQ	Thickness vector components and magnitude
TRQ	Thickness vector components and magnitude
TRS	Thickness vector components and magnitude
TPS	Thickness vector components and magnitude
UN	Unit vectors approximately normal to surfaces 0 through 5
V	T*A*SIN(THETA) for cell sub-elements
ZETA	Orientation angle of cell sub-elements

<u>Usage</u>

CHKPNT, LPBG, LPCFFB, LPCG, LPCSFB, LPMFFB, LPMG, LPMSZD, LPMS1, LPMS2, ØUTPUT, PREBAL, PREBGM, PREBGU, PREBLL, PREBST

LABELED COMMON NDICES

This common block is used to store pointers to partitions within blank common array A. All values in this common block are initialized in routine NITIAL. Partitions for elements are sized for cells which have the largest storage requirements.

Declaration

```
CØMMØN          A( 8000)
CØMMØN /NDICES/IVBDB,IVBB,IVBX,IDPBPU,IPBMUB,IDPPUB,IDBL,IVBL
,               IPBAR,IPBPHU,IKB,ICB,IMBAR,IMBARL,IDEBCL,ICØNST
,               ITIME,ISIGSS
,               IKBL,IZ,ICA,IQ,IVAL,IFTAU,IVEC,IEGSYS
,               IU,LU,IECT,IEVT,IMPT,IDEBO,IDFBKO,IDSEB
,               IFK,IFBK,IDEL,IDEDL,IFSFB,IFSFBB,IPBCU
,               IPBKU,ISIGFB,ISIGBH,IEPSIG,IPSLØN,ID,IDK
,               ISKB,ISKBB,ISCB,ISCBB,IPBUF,ISK,ITK,ICIB,IMEL
,  X1(30)
```

Contents

IVBDB	$\dot{\bar{v}}_{\beta}$, modal accelerations
IVBB	$\bar{v}_{\beta}$, modal velocity
IVBX	$\bar{v}_{\beta-1}$, modal velocity of previous increment
IDPBPU	$\delta\bar{P}_{U}$, incremental modal force imbalance
IPBMUB	$\bar{P}_{(M)U}$, modal inertia force
IDPPUB	$\delta\bar{P}_{(\phi)U}$, incremental modal applied load
IDBL	$\delta\bar{\Delta}_{L}$, incremental linear modal displacement
IVBL	$\bar{v}_{L}$, linear modal displacement
IPBAR	$\bar{P}$, total modal forces
IPBPHU	$\bar{P}_{(\phi)U}$, modal applied load
IKB	$\bar{K}$, modal stiffness
ICB	$\bar{C}$, modal damping
IMBAR	$\bar{M}$, modal mass

IMBARL	Cholesky decomposition of $\bar{M}$
IDEBCL	$\delta\bar{E}_{CL}$, incremental element dissipated damping energy
ICØNST	Problem constants
ITIME	Incremental time history
ISIGSS	$\sigma_{s\sigma}$, stress transform for cell elements
IKBL	Cholesky decomposition of $\bar{K}$
IZ	The modal matrix $Z = \bar{K}^{-1}_{\beta-1}\bar{P}_{\beta}$
ICA	The modal matrix $C_a = H^{-1}$ yo
IQ	The modal matrix $Q = C_{a_D}\ F(\tau)$ where C_{a_D} is C_a diagonalized
IVAL	λ, eigenvalues
IFTAU	$F(\tau)$, a modal column matrix of $e^{\lambda\tau}$
IVEC	H, eigenvectors
IEGSYS	Eigenvalue problem work array
IU	U, joint coordinate table
LU	Joint coordinate update flags
IECT	ECT, element constant table
IEVT	EVT, element variable table
IMPT	MPT, material property tables
IDEBO	$\delta\bar{e}_o$, incremental initial element deformations
IDFBKO	$\sigma\bar{F}_{K_o}$, incremental element forces due to initial deformations
IDSEB	$\delta\bar{e}$, incremental element deformations
IFK	The matrix FK = $\bar{F}_{K_{\beta-1}} + \delta\bar{F}_{K_o}$ for an element
IFBK	$\bar{F}_K$, element forces
IDEL	δe_L, incremental linear element displacement
IDEDL	$\delta\dot{e}_L$, incremental linear element velocity

IFSFB	$F_{\bar{F}}$, element force transform
IFSFBB	$F_{\bar{F}_{\bar{\beta}}}$, average of $F_{\bar{F}_{\beta-1}}$ and $\bar{F}_{\beta}$
IPBCU	$\bar{P}_{(C)U}$, modal damping forces
IPBKU	$\bar{P}_{(K)U}$, modal stiffness forces
ISIGFB	$\sigma_{\bar{F}}$, element stress transform
ISIGBH	σ_{β}, preliminary element stresses
IEPSIG	ε_{σ}, element strain transform
IPSLØN	ε, element strains
ID	The matrix $D = \bar{P}_{UF} F_{\bar{F}}$ for an element
IDK	The matrix $DK = \bar{P}_{UF} F_{\bar{F}} \bar{k}$ for an element
ISKB	$\bar{k}$, unassembled element stiffness
ISKBB	$\bar{K}_{\bar{\beta}}$, average of $\bar{K}_{\beta-1}$ and $\bar{K}_{\beta}$
ISCB	$\bar{c}$, unassembled element damping
ISCBB	$\bar{C}_{\bar{\beta}}$, average of $\bar{C}_{\beta-1}$ and $\bar{C}_{\beta}$
IPBUF	$\bar{P}_{UF}$, modal transform
ISK	s_k, modal fictitious force equivalent loads
ITK	t_k, modal fictitious deformation equivalent loads
ICIB	The matrix $\bar{C}\bar{I} = \bar{P}_{UF} F_{\bar{F}} \bar{c} F_{\bar{F}}^{T} \bar{P}_{UF}^{T}$ for an element
IMEL	m, element mass
X1	Pad for expansion of NDICES

Usage

RESPNS, AVRAGE, CHKØUT, CHKPNT, EIGSØL, FIFDEF, FØRMAT, GEØM, IMBAL, NDXSET, NITIAL, ØUTPUT, PØLSØL, PBARUF, PREBAL, PREBGM, PREDBGU, PREBLL, PREBST, PREEIG, PTMASS, READK

LABELED COMMON TAPES

This common block is used to store the FORTRAN logical unit numbers of the external files used by the program.

Declaration

```
COMMON /TAPES /TAPES(20), X2(10)
```

Contents

The values in the common block are initialized in routine NITIAL. The FORTRAN logical unit designations and equivalences assigned are given here.

TAPES(1)=N1=	1
TAPES (2)=N2=	2
TAPES (3)=N3=	3
TAPES (4)=N4=	4
TAPES (5)=N5=	8
TAPES (6)=N6=	9
TAPES (7)=N7=	10
TAPES (8)=N8=	11
TAPES (9)=N9=	12
TAPES(10)=N10=	13
TAPES(11)=N11=	14
TAPES(12)=N12=	15
TAPES(13)=N13=	16
TAPES(14)=N14=	17
TAPES(15)=N15=	18
TAPES(16)=N16=	19
TAPES(17)=N17=	20
TAPES(18)=N18=	21
TAPES(19)=N19=	22
TAPES(20)=N20=	23

Array X2 is a pad for expansion of labeled common block TAPES.

Usage

RESPNS, CHKPNT, EIGSØL, FIFDEF, IMBAL, NITIAL, ØUTPUT, PBARUF, PREBAL, PREBGM, PREBGU, PREBLL, PREBST, PREEIG, PTMASS, READK

SUBROUTINE AVRAGE

This routine computes the fictitious force and deformation matrices n_k and r_k.

Algorithm

Matrices n_k and r_k are computed according to equations H.278 through H.329 given in Appendix H of Part 1.

Input/Output

None

Argument List

NUM	An integer scalar defining the length of the ECT record
FFDB	A real array for matrix $F_{\bar{\bar{F}}_{\beta}}$
FFDB1	A real array for matrix $F_{\bar{\bar{F}}_{\beta-1}}$
GFB	A real array for matrix $G_{f_{\beta}}$
GFB1	A real array for matrix $G_{f_{\beta-1}}$
RSR	A real array for matrix $r_{\acute{r}}$
FBF	A real array for matrix $\bar{F}_K$
DO	A real array for matrix D_O
RNK	A real array for matrix n_k
RRK	A real array for matrix r_k

Labeled Common

LIMITS, NDICES, CRRAYS

Subroutines Called

MLTMAT, NKRK, TRNMLT

Error Detection

None

SUBROUTINE CHKØUT

This routine prints intermediate data during the execution of the fictitious force and deformation module.

Algorithm

Depending on the value of the argument ICASE, control is transferred to a section of code which sets the first location of the array partition to be printed and the row/column dimensions of the partition. Appropriate title information is also set.

Control is then transferred to one of three calls to routine MATPRT to print the partition from one of three arrays.

Input/Output

Data to be printed is written to file 6.

Argument List

ICASE An integer scalar defining which data is to be printed

B A real array of intermediate data

Labeled Common

CRRAYS, FICNDX, NDICES, LIMITS

Subroutines Called

MATPRT

Error Detection

None

SUBROUTINE CHKPNT

This routine prints intermediate data at selected points during the execution of routine PREBAL.

Algorithm

Routine PREBAL calls this routine at selected check points during its execution. At each point, selected intermediate data is printed using routine PLØP. Logic is provided to control printing at each point using array KPRNT which is initialized in a DATA statement. Printing takes place at each point only if the corresponding value in array KPRNT is non-zero. Also, all printing can be suppressed by making KPRNT (15) equal to zero.

Input/Output

All printed output is written on file 6.

Argument List

I	An integer scalar defining the element number for which data is being printed
J	An integer scalar defining the checkpoint for which data is being printed

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

PLØP

Error Detection

None

SUBROUTINE EDGUNV

This routine calculates the edge vectors of an element from the joint coordinate data.

Algorithm

Repeated constants are calculated and the number of edges for the element type that is under consideration is set. The joint pairs that determine an edge vector are set and the corresponding edge vector is calculated from the coordinates. This is done for one edge for a bar, the 4 edges of a membrane and the edges associated with joints 1, 2, 3 and 4 of a cell. If the element is a cell, then the basic edge calculations just obtained are copied into the next 12 memory locations, rather than recalculated for the other surface. Using the cell thickness data developed in subroutine GEØM , the complete edge vector calculations are performed for all edges and stored in variable array EDGV.

For all element types the edge unit vectors are calculated and stored in variable array UNV.

Input/Output

Error messages, if any, are written to file 6.

Argument List

IECREC An integer array for the ECT table record for the element under consideration

NUM An integer scalar defining the number of words in the ECT record

Labeled Common

FICNDX, GEØMS

Subroutines Called

None

Error Detection

Testing is performed to detect zero edge lengths. If found, a diagnostic is written to that effect, the variable IERRØR is set to 1 and control is returned to the calling routine GEØM.

SUBROUTINE EIGSØL

This routine assembles the A matrix, solves for all eigenvalues and eigenvectors, and computes the modal response.

Algorithm

Input to EIGSØL are the core resident transformed modal stiffness, damping, decomposed mass, and loads matrices $\bar{K}$, $\bar{C}$, $\bar{M}^{-1}$, and $\bar{P}$, respectively. Also input for all iterations other than the first are the equivalent loads derived from fictitious force and deformation effects computed by routine FIFDEF which are also core resident. Output from EIGSØL are the modal displacements, velocities, and accelerations, $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$, respectively. The equations for solution of the modal response are given in Appendix H of Part 1 (H.161 through H.175).

First, the load variation constants, ω_p, c_p, and c_f are computed. The modal loads $\bar{P}$ and the incremental equivalent loads $\delta\bar{P}_{(f)}$ are scaled by c_p and c_f, respectively. Routine NEWKC is invoked to form matrix $\bar{K}_C$ and its inverse. The real and imaginary parts of $\delta\bar{\Delta}_C$ and $\delta\bar{\Delta}_f$ are then computed and stored in A(IKBL) and A(IKBLX), respectively.

The A matrix is then assembled and all eigenvalues, λ, and eigenvectors, H, are computed by routine RGEIG. The eigenvectors are written to N8 before being inverted by routine JØRDAN.

The modal response equations are then solved for $\delta\bar{\Delta}$, $\bar{v}$, and $\dot{\bar{v}}$. During these computations, the eigenvectors are read from N8 where they had been temporarily stored.

Input/Output

The eigenvectors H are written to file N8 and are subsequently read back into core.

Argument List

B	A real array for the A matrix
X	A real array for the eigenvectors
M	An integer scalar defining the order of the problem
N	An integer scalar equal to M/2

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

CCMULT, CRMULT, JØRDAN, NEWKC, PLØP, READA, RGEIG, SMULT, SYMFIL, SYMSØL, VDMULT, WRITE, XPØNNT

Error Detection

None

SUBROUTINE FIFDEF

This routine is the driver routine for the module that generates the elemental fictitious force and deformation matrices based on linear incremental displacements and average geometry (equations H.181 through H.331 in Appendix H of Part 1).

Algorithm

Pointers are identified for those matrices stored in blank common array A and initialization is performed.

For each element of the structure the following processing steps are performed. The ECT table record is read from file N12. If the element has failed, then the next element is read. Matrices $\bar{P}_{U\underline{F}}$, $F_{\bar{F}}$ and DK are read from files N11, N13, and N6 respectively. Matrix $\bar{k}$ is also read from N6. Processing parameters are set for the element type; bar, membrane or cell. δ_{e_L} and $\delta_{\dot{e}_L}$ are then calculated. The element deformations flag is set to zero, meaning that deformations will not be added to the core resident joint coordinates. The element geometry is calculated by routine GEØM relative to the joint coordinates at the beginning of the increment.

The variable BETBAR, which indicates whether this is the second iterative solution to account for material nonlinearity, is then tested. This test and the processing initiated when BETBAR is true are part of the original design and are no lonaer applicable. This is also true regarding the test and processing associated with the variable HEAT which is the option of accounting for damping energy converted to heat during impact.

Matrix $FK_{\beta-1}$ is read from file N6 and, if not the first iteration for geometric nonlinearity, the element forces due to fictitious deformations are read from file N30. The final element forces $\bar{F}_{K_\beta}$ for this iteration are then computed. If this is the last iteration, processing should end here. This would avoid the regeneration of fictitious forces and deformations for the last iteration. Appropriate testing and transfer instructions should be inserted in the code.

Matrices $F_{\bar{\bar{F}}}$, G_f, D_o, and D_f are then calculated based on the final element forces and the geometry at the beginning of the increment. This data is written to file N3. Code should be inserted here to avoid regeneration of this data for iterations other than the first. The data stored on file N3 should be read and re-used.

The geometry is then temporarily updated according to δ_e, the element displacements, and matrices $F_{\bar{\bar{F}}}$, G_f, and $\bar{F}_f$ generated for the new geometry. The average of these matrices for new and old geometry is computed. From the average matrices, the fictitious element deformations $\delta_{\bar{\bar{e}}}$ are computed as are the n_k and r_k coefficients. Finally, the equivalent applied loads for fictitious force and deformation effects s_k and t_k, respectively, are computed.

If this is not the last iteration for geometric nonlinearity, the element forces due to fictitious deformations are computed and written to file N31. If it is the last iteration, the final element forces and deformations and velocities computed earlier are written to files N5 and N7, respectively.

This ends the processing for an element. Control is transferred to the beginning of this loop to process the next element.

Input/Output

The matrices are read or written as indicated below

$\bar{P}_{UF}$	read from N11
$F_{\bar{F}_{\beta-1}}$	read from N13
ECT	read from N12
$\bar{F}_K$	written on N5
DK,$\bar{k}$,FK	read from N6
$\delta\bar{F}_{K_f}$(old)	read from N30
$\delta\bar{F}_{K_f}$(new)	written on N31
δe,$\delta\dot{e}$	written on N7

Intermediate data is printed on file 6.

Argument List

NFFINC	An integer scalar defining the iteration number
NFFMAX	An integer scalar defining the maximum number of iterations
N30	An integer scalar defining the input file for $\delta\bar{F}_{K_f}$
N31	An integer scalar defining the output file for $\delta\bar{F}_{K_f}$

Labeled Common

LIMITS, NDICES, TAPES, FICNDX, GEØMS, CRRAYS

Subroutines Called

ABSYM, ASYMB, AVRAGE, CHGSGN, CØPY, GEØM, LPBFIC, LPCFIC, LPMFIC, MLTMAT, MULT, PBARUF, READA, READM, TMULT, WRITE, WRITEM

Error Detection

None

SUBROUTINE FØRMAT

This routine outputs matrices $\bar{K}$, $\bar{C}$, and $\bar{M}$ in a format consistent with FØRMAT master input/output tapes.

Algorithm

The core resident matrices $\bar{K}$, $\bar{C}$, and $\bar{M}$, which are stored in upper triangular row-wise form, are assembled in full form using routine SYMFIL. Each is then output to file 19 as FØRMAT matrix data.

This routine was used in the early stages of program development and is no longer required nor called by the final version of the program.

Input/Output

A FØRMAT matrix tape is generated on file 19 containing matrices $\bar{K}$, $\bar{C}$, and $\bar{M}$.

Argument List

None

Labeled Common

NDICES, LIMITS

Subroutines Called

CHGSGN, SYMFIL, WRITE

Error Dectection

None

SUBROUTINE GEØM

This routine stores the joint coordinates for an element and, conditionally, adds element deformations. Thickness vectors are calculated for cell elements and edge and unit vectors are calculated for all elements.

Algorithm

The element joint coordinates are accessed in the coordinate table and stored in array CØØRD. The deformation flag IDEFRM is checked and, if equal to one, the element displacements are added to the joint coordinates. If the element is a cell, then the cell thicknesses are calculated, and subsequently, the cell thickness differentials.

For any element type, routine EDGUNV is called to determine the edge and unit vectors.

Input/Output

None

Argument List

NUM	An integer scalar defining the number of words in the ECT record, used to determine element type
IDEFRM	An integer scalar flag for deformations
REC	A real array for storage of the element ECT table

Labeled Common

GEØMS, FICNDX, NDICES

Subroutines Called

EDGUNV

Error Detection

None

SUBROUTINE IMBAL

This routine computes the stresses and strains of each element and writes the element forces, stresses, and strains to the master output file.

Algorithm

First, the logical variable BETABAR is tested to determine if this is the second iterative solution due to material nonlinearity. This test and the processing it initiates, as well as other tests and processing related to BETABAR equal true, are part of the original design and are no longer required.

The cumulative transformed applied loads $\bar{P}_{(\phi)U}$ are then augmented with their incremental component. The equilibrium imbalance $\delta\bar{P}_U$ is then computed. The subsequent test of the imbalance to check the solution is part of the original design and is no longer required.

The modal response consisting of $\bar{\Delta}$, $\delta\bar{\Delta}$, $\bar{v}$, $\dot{\bar{v}}$, $\bar{P}_{(\phi)U}$, and $\delta\bar{P}_U$ is then assembled and is output as a column matrix with the name RESPNS and a subscript equal to the increment number.

A loop is then entered whose index is the number of element types. Control flags are then set according to element type. The elements types are processed in the order bars, membranes, and cells. If elements of a given type exist in the model, a second inner loop is entered whose index is the number of elements of that type.

A matrix is written to the master output file for each element type that exists in the model. The following operations are performed for each element in the inner loop.

Read $\sigma_{\bar{F}}$, ϵ_σ, $\bar{F}_K$, $\epsilon_{\beta-1}$ and $\delta\bar{F}_K$ from scratch files

$\sigma = \sigma_{\bar{F}} \bar{F}_K$ for bars and membranes or

$$\sigma = \sigma_{s\sigma} \sigma_{\bar{F}} \bar{F}_K \text{ for cells}$$

$$\varepsilon = \varepsilon_{\beta-1} + \varepsilon_{\sigma} \sigma_{\bar{F}} \delta\bar{F}_K$$

Write $\bar{F}_K$, σ, and ε to master output file

The element forces, stresses, and strains are arranged in that order into a single column of the output matrix for each type of element. The matrix names used for the three element types are BARS, MEMBRN, and CELLS. Each is subscripted with the increment number.

After processing all elements, external files are positioned and appropriate files are flip flopped according to the original design.

Input/Output

The following matrix data is read and written during IMBAL:

$\sigma_{\bar{F}}$, $\delta\bar{F}_K$	read from	N6
$\bar{F}_K$	read from	N5
ε_{σ}	read from	N16
$\varepsilon_{\beta-1}$	read from	N9
ε_{β}	written to	N10

The modal response and element forces, stresses, and strains are written to the master output file N20.

Argument List

DBLL A real array for the cumulative modal displacements $\bar{\Delta}$

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

ABSYM, ADD, ASYMB, HALT, MULT, PLØP, READA, READM, SYMABT, WRITEM, ZERØ

Error Detection

Error conditions are tested for when BETABAR equal true but these tests are no longer applicable.

SUBROUTINE LPBFIC

This routine calculates the geometric and force matrices for fictitious force and deformation effects from bar elements.

Algorithm

Matrices G_f, $\bar{F}_{f_1}$ and $\bar{\bar{F}}_F$ are computed according to equations H.181 through H.192 given in Appendix H of Part 1. Input consists of geometric data in labeled common block GEØMS and the element forces $\bar{F}_K$.

Input/Output

None

Argument List

FFBB	A real array for matrix $\bar{\bar{F}}_F$
GF	A real array for matrix G_f
FBF	A real array for matrix $\bar{F}_f$
FB	A real array for matrix $\bar{F}_K$

Labeled Common

GEØMS

Subroutines Called

None

Error Detection

None

SUBROUTINE LPBG

This routine regenerates matrices $F_{\bar{\bar{F}}}$ and $\sigma_{\bar{\bar{F}}}$ for bar elements.

Algorithm

Using the joint coordinate table stored in array U and the element definition stored in labeled common LPEM, this routine forms matrices $F_{\bar{\bar{F}}}$ and $\sigma_{\bar{\bar{F}}}$ as follows (array dimensions appear on the left):

$$(3 \times 1) \qquad \bar{a} = \frac{\overline{pq}}{|\overline{pq}|}$$

$$(7 \times 2) \qquad F_{\bar{\bar{F}}} = \left[\begin{array}{c|c} -a_x & \\ -a_y & \\ -a_z & \\ \hline & a_x \\ & a_y \\ & a_z \\ \hline 1.0 & -1.0 \end{array}\right]$$

$$(1 \times 2) \qquad \sigma_{\bar{\bar{F}}} = \left[\begin{array}{c|c} \frac{1}{2A} & \frac{1}{2A} \end{array}\right]$$

Input/Output

None

Argument List

U	A real array of joint coordinates
FFBAR	A real array for matrix $F_{\bar{\bar{F}}}$
SIGFB	A real array for matrix $\sigma_{\bar{\bar{F}}}$

Labeled Common

LPEM

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPCFFB

This routine regenerates matrix $F_{\bar{F}}$ for a cell element.

Algorithm

This routine regenerates matrix $F_{\bar{F}}$ in a manner identical to routine LPCFFB in the Initial Generator (see Appendix B). The coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

FFBAR	A real array for matrix $F_{\bar{F}}$
HØLD	A real array for intermediate storage

Labeled Common

LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE LPCFIC

This routine calculates the geometric and force matrices for fictitious force and deformation effects from cell elements.

Algorithm

Matrices D_o, G_f, $\bar{F}_f$, r_r', and $F_{\bar{\bar{F}}}$ are computed according to equations H.225 through H.277 given in Appendix H of Part 1. Input consists of geometric data in labeled common block GEØMS and the element forces $\bar{F}_K$.

Input/Output

None

Argument List

FFBB	A real array for matrix $F_{\bar{\bar{F}}}$
RR	A real array for matrix r_r'
GF	A real array for matrix G_f
FBF	A real array for matrix $\bar{F}_f$
DO	A real array for matrix D_o
FB	A real array for matrix $\bar{F}_K$

Labeled Common

GEØMS

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPCG

This routine computes geometric data and panel K factors for a cell element.

Algorithm

This routine computes all geometric data and panel K factors in a manner identical to routine LPCG in the Initial Generator (see Appendix B). It does not, however, compute element volume and mass. Otherwise, the coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

U	A real array of joint coordinates

Labeled Common

LPEM

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPCSFB

This routine regenerates matrix $\sigma_{\bar{F}}$ for a cell element.

Algorithm

This routine regenerates matrix $\sigma_{\bar{F}}$ in a manner identical to routine LPCKC in the Initial Generator (see Appendix B). It does not, however, compute the stiffness matrix. Otherwise, the coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

SFBAR	A real array for matrix $\sigma_{\bar{F}}$
JHØLD	An integer array for intermediate storage

Labeled Common

LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMFFB

This routine regenerates matrix $F_{\bar{F}}$ for a membrane element.

Algorithm

This routine regenerates matrix $F_{\bar{F}}$ in a manner identical to routine LPMFFB in the Initial Generator (see Appendix B). The coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

FFBAR	A real array for matrix $F_{\bar{F}}$
HØLD	A real array used for temporary storage

Labeled Common

LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE LPMFIC

This routine calculates the geometric and force matrices for fictitious force and deformation effects from membrane elements.

Algorithm

Matrices D_o, G_f, $\bar{F}_f$, r_r', and $F_{\bar{\bar{F}}}$ are computed according to equations H.193 through H.224 given in Appendix H of Part 1. Input consists of geometric data in labeled common block GEØMS and the element forces $\bar{F}_K$.

Input/Output

None

Argument List

FFBB	A real array for matrix $F_{\bar{\bar{F}}}$
RR	A real array for matrix r_r'
GF	A real array for matrix G_f
FBF	A real array for matrix $\bar{F}_f$
DO	A real array for matrix D_o
FB	A real array for matrix $\bar{F}_K$

Labeled Common

GEØMS

Subroutines Called

VECT

Error Detection

None

SUBROUTINE LPMG

This routine computes geometric data and panel K factors for a membrane element.

Algorithm

This routine computes all geometric data and panel K factors in a manner identical to routine LPMG in the Initial Generator (see Appendix B). It does not, however, compute element volume or mass. Otherwise, the coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

U	A real array of joint coordinates

Labeled Common

LPEM

Subroutines Called

VECT

Error Detection

None

SUBRØUTINE LPMSƵD

This routine computes the area matrices and the global transform matrices used to compute matrix $\sigma_{\bar{F}}$ for a membrane element.

Algorithm

This routine regenerates matrix $\sigma_{\bar{F}}$ in a manner identical to routine LPMSƵD in the Initial Generator (see Appendix B). It does not, however, compute the $\tilde{\mho}$ matrices. Otherwise the coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

SFBAR	A real array for matrix $\sigma_{\bar{F}}$

Labeled Common

LPEM

Subroutines Called

LPMS1, LPMS2, LPMƵ

Error Detection

None

SUBRØUTINE LPMS1

This routine forms matrix $\sigma_{\bar{F}}$ for a parallelogram membrane element.

Algorithm

This routine regenerates matrix $\sigma_{\bar{F}}$ in a manner identical to routine LPMS1 in the Initial Generator (see Appendix B). The coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

SFBAR	A real array for matrix $\sigma_{\bar{F}}$

Labeled Common

LPEM

Subroutines Called

None

Error Detection

None

SUBRØUTINE LPMS2

This routine forms matrix $\sigma_{\bar{\bar{F}}}$ for an approximate parallelogram membrane element.

Algorithm

This routine regenerates matrix $\sigma_{\bar{\bar{F}}}$ in a manner identical to routine LPMS2 in the Initial Generator (see Appendix B). The coding is identical except for common blocks and arguments.

Input/Output

None

Argument List

SFBAR   A real array for matrix $\sigma_{\bar{\bar{F}}}$

Labeled Common

LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE MATPRT

This routine is a matrix print utility routine used by the fictitious force and deformation module.

Algorithm

The matrix to be printed is assumed to be full and real mode. After printing a title line, the matrix is printed row by row.

Input/Output

Matrix data is printed on file 6.

Argument List

TARG	A real array for the matrix to be printed
IRØW	An integer scalar defining the row dimension
ICØL	An integer scalar defining the column dimension
CØMENT	An alphanumeric array of descriptive information

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE MLTMAT

This routine performs an incore matrix multiplication.

Algorithm

If a cumulative summation of matrix products is not requested, the product matrix is initialized to zero. The cumulative summation of matrix products is then performed according to

$$C = C + AB$$

Input/Output

None

Argument List

C	A real array for the product matrix
A	A real array for the premultiplier matrix
B	A real array for the postmultiplier matrix
L	An integer scalar defining the usable row dimension of A and C
M	An integer scalar defining the usable column dimension of A and the usable row dimension of B
N	An integer scalar defining the usable column dimension of B and C
MRØW	An integer scalar defining the row dimension of A and C
MC1	An integer scalar defining the row dimension of B and the column dimension of A
MC2	An integer scalar defining the column dimension of B and C
SUM	A logical scalar flag for cumulative summation

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE NEWKC

This routine computes the inverse of the augmented modal stiffness matrix $\bar{K}_C$.

Algorithm

The augmented modal stiffness matrix is formed according to

$$\bar{K}_C = \bar{K} - \omega_p^2 \bar{M} + j \omega_p \bar{C}$$

and routine JØRDAN is called to invert $\bar{K}_C$.

Input/Output

None

Argument List

X	A real array for storing the real and imaginary parts of $\bar{K}_C$
A	A real array for matrix $\bar{K}$
B	A real array for matrix $\bar{M}$
C	A real array for matrix $\bar{C}$
D	A real array used by routine JØRDAN to compute the inverse of $\bar{K}_C$
NG	An integer scalar defining the order of the problem
WW	A real scalar for the load variation constant ω_p
WWSQ	A real scalar for the square of the load variation constant ω_p

Labeled Common

None

Subroutines Called

JØRDAN

Error Detection

None

SUBRØUTINE NITIAL

This routine performs the initialization steps necessary before entering the incremental solution.

Algorithm

Routine NITIAL performs the first stage of initialization process. First, labeled common block TAPES is initialized with the FORTRAN logical unit numbers to be used for the scratch files. Routine MATIN is then called to read the first master input tape and copy each input matrix to individual scratch files. The input file contains matrices $\bar{P}_{UF}$, MPT, U_o, ECT, m, $\bar{k}$, $\bar{c}$, $\bar{F}_{\bar{F}}$, $\sigma_{\bar{F}}$, ε_σ, $\bar{e}_T$, EVT, and CØNST assembled in a previously executed FORMAT step. Only matrices CØNST and U_o remain core resident.

All card input is then read by routine NITIAL consisting of run parameters and incremental time history. The first data card is read with a format (4I6, 1L1, 2I6) and contains the following run parameters:

BETA	beginning time interval
NELEMS	total number of physical elements (excludes point mass elements)
NTRVLS	number of time intervals
NG	number of transformation modes
HEAT	"F", dummy logical control flag
NJTS	number of joints
NWØRK	optional extent of blank common

The array NTRVLS, ascending values of elapsed time, is then read with format (6E12.0) and tested for ascending order.

Using the problem sizing information obtained from the matrix and card input, the process of allocating partitions of the blank common region for each array required in subsequent processing steps is performed. The first location of each of these partitions is stored in labeled common block NDICES.

Next, the cell stress transform, $\sigma_{S\sigma}$, is initialized since it is constant for all cell elements. Routine READK is then called to rewrite matrices $\bar{k}$, $\bar{c}$, $F_{\bar{F}}$, ε_{σ}, and $\delta\bar{e}_T$ in optimum format.

The original joint coordinate data is then read into core from the scratch file where it had been written by routine MATIN. Before exiting NITIAL, various blank common partitions are initialized to zero including those for $\bar{v}$, $\dot{\bar{v}}$, $\delta\bar{P}_U$, $\bar{P}_{(M)U}$, $\delta\bar{P}_{(\phi)U}$, and $\delta\bar{E}_{CL}$.

Interspersed in the above initialization steps is the setting of problem parameter variables in labeled common block LIMITS.

Some code exists in this routine which was intended for restart capability which is not operational and could be deleted. This code can be identified by a test for BETA not equal to one which implies a restart run.

Input/Ouput

All card input is read from file 5 and all printed output is written to file 6. Card input consists of a single card of run parameters and the incremental time history. Printed output consists of incremental time history and error messages, if any.

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

ASYMB, CHGSGN, MATIN, NDXSET, PLØP, READA, READK, ZERØ

Error Detection

The incremental times are tested for ascending order and, if an error is found, the run is terminated. The available workspace is also tested against that required for the problem and, if insufficient, the run is terminated. Appropriate error messages are printed on file 6 in each case.

SUBROUTINE NKRK

This routine computes the fictitious force and deformation matrices n_B and r_B for membrane and cell elements.

Algorithm

Matrices n_B and r_B are computed for membrane and cell elements according to equations H.293 through H.324 given in Appendix H of Part 1.

Input/Output

Error messages, if any, are written to file 6.

Argument List

AA	A real scalar equal to the length a
B	A real scalar equal to the length b
SKE	A real scalar equal to the skew angle ξ
AEO	A real scalar equal to the element deformation $\bar{\bar{e}}_o$.
Q	A real scalar equal to the element force f
EBBL	A real array for matrix $\delta\tilde{e}$
RN	A real array for matrix n_B
RB	A real array for matrix r_B

Labeled Common

LIMITS, FICNDX, CRRAYS

Subroutines Called

None

Error Detection

The sine and cosine of the angle SKE are tested for zero. If true, appropriate error messages are printed, the error flag is set to one, and control returned to the calling routine.

SUBROUTINE ØUTPUT

This routine transcribes certain input data to the master output file.

Algorithm

Input matrices U_o, $\delta\bar{P}_{(\phi)U}$, and $\bar{P}_{UPTJ}$ are written to the master output file as are the input array of problem constants and the card input array of incremental times. Before outputing the array of problem constants, it is augmented with the number of time intervals and number of modes from the card input data.

The FØRMAT tape name and modifier used is TAPE, 1 and is imbedded in the code. The FØRMAT matrix names and modifiers of the output matrices and their sequence of appearance on the master output file is as follows:

CØNST, 1	Problem constants
TIME, 1	Incremental time history
UZERØ, 1	Original joint coordinates, U_o
(DPBPHI, 1)	Incremental applied loads, $\delta\bar{P}_{(\phi)U}$
(PBUPTJ, 1)	Modal to joint T degree of freedom transform, $\bar{P}_{UPTJ}$

where the names in parenthesis are user defined in a previous FØRMAT step and all others are imbedded in the code.

Input/Output

A FØRMAT matrix data tape is constructed on file N20 and consists of five matrices. File N18 is read to copy two of these matrices.

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

READA, WRITE

Error Detection

None

SUBROUTINE PLØP

This routine prints any real or integer array with specified headings.

Algorithm

First, a title line is printed using a single alphanumeric word of descriptive information and row/column dimensions of the array to be printed. The variable NT is then tested to determine if the array mode is real or integer. Real arrays are then printed row-wise at 5 values per line while integer arrays are printed row-wise at 10 values per line.

Input/Output

Printed output is written to file 6.

Argument List

NAME	An alphanumeric scalar for descriptive information
IR	An integer scalar defining the row dimension of the array
IC	An integer scalar defining the column dimension of the array
A	A real or integer array of data to be printed
NR	An integer scalar defining the row dimension for printing
NC	An integer scalar defining the column dimension for printing
NT	An integer scalar defining whether the array is real or integer mode

Labeled Common

None

Subroutines Called

None

Error Detection

None

FUNCTION PØLSØL

This routine solves a third order polynomial for the modal response correction factor $\hat{a}$.

Algorithm

In the original theoretical approach, geometric nonlinearity was to be accounted for by scaling the modal response by the correction factor $\hat{a}$. This is no longer applicable and, in the final version of the code, routine PØLSØL is never referenced.

The polynomial expression was formed in the following manner.

$$W_{\bar{\bar{F}}_{k\beta}} = -\frac{1}{2}\,\delta\bar{\Delta}^{T}_{L_\beta}\; s_{k\beta} \qquad k = 1, 2, 3$$

$$W_{f_{k\beta}} = -\frac{1}{2}\,\delta\bar{\Delta}^{T}_{L_\beta}\; t_{k\beta} \qquad k = 1, 2$$

$$\delta\bar{E}_{ML\beta} = \frac{1}{2}\,(\bar{v}^{T}_{L_\beta}\;\bar{M}\;\bar{v}_{L_\beta} - \bar{v}^{T}{}_{\beta-1}\bar{M}\;\bar{v}_{\beta-1})$$

$$\delta\bar{E}_{CL\beta} = \frac{\tau\beta}{2}\,(\bar{v}^{T}_{\beta-1}\bar{C}_{\beta-1}\bar{v}_{\beta-1} - \bar{v}^{T}_{L\beta}\bar{C}_{\beta-1}\bar{v}_{L\beta})$$

$$\delta\bar{E}_{KL\beta} = \frac{1}{2}\,\delta\bar{\Delta}^{T}_{L\beta}\;\bar{K}_{\beta-1}\;\delta\bar{\Delta}_{L\beta}$$

$$\delta\bar{E}_{L\beta} = \delta\bar{E}_{ML\beta} + \delta\bar{E}_{CL\beta} + \delta\bar{E}_{KL\beta}$$

$$\Omega_{0_\beta} = -\,\delta\bar{E}_{L\beta}$$

$$\Omega_{1_\beta} = \delta\bar{E}_{L\beta} + W_{\bar{\bar{F}}1\beta} + W_{f1\beta}$$

$$\Omega_{2_\beta} = W_{\bar{\bar{F}}2\beta} + W_{f2\beta}$$

$$\Omega_{3_\beta} = W_{\bar{\bar{F}}3\beta}$$

$$\Omega_{0_\beta} + \Omega_{1_\beta}\hat{a}_\beta + \Omega_{2_\beta}\hat{a}_\beta^2 + \Omega_{3_\beta}\hat{a}_\beta^3 = 0$$

This expression was then solved for the positive value closest to unity.

Input/Output

Intermediate data and $\hat{a}$ are printed on file 6.

Argument List

EBCL   A real scalar equal to $\delta\bar{E}_{CL}$

Labeled Common

NDICES, LIMITS

Subroutines Called

ASYMB, MULT, RGEIG, SMULT, ZERØ

Error Detection

None

SUBROUTINE PREBAL

This routine permanently updates joint coordinates and regenerates matrices $F_{\bar{F}}$ and $\sigma_{\bar{F}}$.

Algorithm

Initialization steps include file positioning, calling routine PREBLL to allocate space for intermediate data, and setting control flags. The modal response is then scaled by the correction factor $\hat{a}$. This step was part of the original design and is no longer required. The transformed modal inertia forces $\bar{\bar{P}}_{(M)U}$ are then computed.

A loop is then entered whose index is the number of physical elements in the model. The first processing step within the loop is the reading of the ECT record for the element from file N12. The record length is tested to determine element type and appropriate control flags are set accordingly.

A test is then made to determine if the element has failed in a previous increment by examining the value of A(KDEBCL). This is part of the original design for material nonlinearity and is not required.

A test is then made on logical variable BETABAR to detect a second iterative solution due to material nonlinearity. This test as well as others appearing within this loop is no longer applicable. The same is true of the processing initiated when BETABAR is true.

The incremental element displacements and velocities, δe and $\delta \dot{e}$, are read from file N7. The scaling of the matrices by $\hat{a}$ is part of the original design and is no longer necessary.

Routine PBARUF is then called to read matrices $\bar{\bar{P}}_{UF}$ and $F_{\bar{F}_{\beta-1}}$ for the element from files N11 and N13, respectively. Routine PREBGM is called to permanently update geometry using the ECT array and matrix δe and to regenerate matrices $F_{\bar{F}_\beta}$ and $\sigma_{\bar{F}}$. Matrix D is then computed for the new geometry.

The element stiffness, damping, and thermal deformation matrices are read from N1 and the element forces previously computed by routine FIFDEF, are read from file N5. The contribution of the element to $\bar{P}_{(C)U}$ is then computed. The incremental element forces are then computed from $\bar{F}_{K_\beta}$ and $\bar{F}_{K_{\beta-1}}$ read from file N4. After this, the element contribution to matrix $\bar{P}_{(K)U}$ is computed.

Finally, matrix $F_{\bar{F}_\beta}$ is output to file N14, matrices $\sigma_{\bar{F}}$ and $\delta\bar{F}_K$ are output to file N6, and matrix D is output to file N19.

This ends the element processing loop. The final processing step consists of positioning files and equating files N1 and N2 and files N16 and N17. This is done because matrices $\bar{k}$, $\bar{c}$, $\delta\bar{e}_T$, and ϵ_T are now assumed constant. In the original design, these files were rewritten and "flip flopped" at each increment since these matrices may be regenerated due to material nonlinearity.

Interspersed in the code of this routine are calls to routine CHKPNT. This routine prints preselected intermediate data at the specified points for program debugging.

Input/Output

The following data is read and written during the execution of PREBAL:

$\bar{P}_{UF}$	read from	N11
ECT	read from	N12
$\bar{k}$, $\bar{c}$, $\delta\bar{e}_T$	read from	N1
$F_{\bar{F}_{\beta-1}}$	read from	N13

$\bar{F}_{K_{\beta-1}}$	read from	N4
$\bar{F}_{K_{\beta}}$	read from	N5
δe, $\delta \dot{e}$	read from	N7
$\bar{F}_{F_{\beta}}$	written to	N14
D	written to	N19
$\sigma_{\bar{F}}$, $\delta \bar{F}_{K}$	written to	N6

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

ADD, ASYMB, CHGSGN, CHKPNT, CØPY, MULT, PBARUF, PREBGM, PREBLL, READA, READM, SMULT, TMULT, WRITE, WRITEM, ZERØ

Error Detection

None

SUBROUTINE PREBGM

This routine controls the updating of joint coordinates and regeneration of matrices $F_{\overset{=}{F}}$ and $\sigma_{\overset{=}{F}}$ for an element.

Algorithm

Routine PREBGU is called first to permanently update joint coordinates. The length of the ECT record is then tested to determine element type and appropriate routines are called to regenerate $F_{\overset{=}{F}}$ and $\sigma_{\overset{=}{F}}$ based on the new geometry.

Input/Output

None

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

LPBG, LPCFFB, LPCG, LPCSFB, LPMFFB, LPMG, LPMSZD, PREBGU, ZERØ

Error Detection

None

SUBROUTINE PREBGU

This routine permanently updates joint coordinates and assembles element parameters from array A(IECT).

Algorithm

The length of the ECT record is tested to determine element type and a transfer made to the appropriate code. Specific data for each type element is extracted from array A(IECT) and stored in array JM in labeled common block LPEM. The data extracted for a bar is the area; for a membrane, the thickness and stress orientation angle; and for a cell, the stress orientation angle. The number of joints for the element type is also set in this section of code.

A loop is then entered whose index is the number of joints defining the element. Within this loop, the joint numbers of the joints defining the element are also stored in array JM and are used together with the element displacements to update the joint coordinates. Array A(LU) is used to flag each joint as it is updated since more than one element may be connected to a joint and each joint need be updated only once.

Input/Output

None

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE PREBLL

This routine allocates partitions in array A in blank common for temporary use by routine PREBAL.

Algorithm

Three indices for partitions of length 30, 30, and 240 are determined and stored in labeled common block LPEM. The partitions are contiguous and begin at A(ISK).

All additional code in this routine is no longer applicable since it is based on the original theoretical approach using $\hat{a}$.

Input/Output

In the obsolete code, matrices δe and $\delta \dot{e}$ were computed and written to file N7.

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

CHGSGN, PBARUF, TMULT, WRITEM

Error Detection

None

SUBROUTINE PREBST

This routine controls the updating of element stiffness and damping.

Algorithm

This routine was intended to be the executive routine for the element stiffness and damping updating module. The design would provide for calls to other routines to test the change in temperature and stress/strain state of the element since last update. If necessary, other routines would be called to regenerate material properties, matrices $\bar{k}$ and $\bar{c}$, and update the EVT table.

This routine currently performs no operations other than setting two logical flags false which were intended to indicate element failure and element updating. In the final version of the code, this routine is never referenced.

Input/Output

None

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES, LPEM

Subroutines Called

None

Error Detection

None

SUBROUTINE PREEIG

This routine generates the transformed modal matrices, $\bar{K}$, $\bar{C}$, $\bar{M}$, and $\bar{P}$.

Algorithm

Incremental solution processing begins with the execution of routine PREEIG. First, the transformed incremental applied loads for the increment, if any, are read from file N18. The matrix of transformed loads $\bar{P}$ is then partially assembled according to

$$\bar{P} = \delta\bar{P}_U - \bar{P}_{(M)U} + \delta\bar{P}_{(\phi)U}$$

where $\delta\bar{P}_U$ are the unbalanced forces from the previous increment, $\bar{P}_{(M)U}$ are the inertia forces from the previous increment, and $\delta\bar{P}_{(\phi)U}$ are the applied loads.

A loop is then entered whose index is the number of elements in the structural model. The following operations are performed for each element.

```
Read $\bar{k}$, $\bar{c}$, and $\delta\bar{e}_T$ from file N1
$\delta\bar{F}_{Ko} = -\bar{k}\,\delta\bar{e}_T$
Read D from file N3 ($D = \bar{P}_{UF}\,F_{\bar{F}}$)
$\bar{P} = \bar{P} + D\,\delta\bar{F}_{Ko}$
$DK = D\,\bar{k}$
$\bar{K} = \bar{K} + DK\,D^T$
$\bar{C} = \bar{C} + D\,\bar{c}\,D^T$
Read $\bar{F}_K$ from file N4
$FK = \bar{F}_K + \delta\bar{F}_{Ko}$
Write DK, $\bar{k}$ and FK to file N6
Write $\delta\bar{F}_{Ko}$ to file N15
```

The last step in this sequence, writing $\delta\bar{F}_{Ko}$ to file N15 is part of the original design code and is not presently required.

After all bars, membranes, and cell elements have been processed, the assembly of the transformed loads is completed by subtracting the modal damping forces of the previous increment $\bar{P}_{(C)U}$ from the partially assembled loads $\bar{P}$.

The sequence of operations above is slightly different on the initial pass for the first increment. Matrix D is computed from matrices $\bar{P}_{UF}$ and $\bar{F}_{\bar{F}}$ read from files N11 and N13, respectively, and then matrix D is written to file N3. Matrix $\bar{F}_K$, the element forces from the previous increment, is initialized to zero rather than read from file N4. Matrix $\bar{M}$ is computed according to

$$\bar{M} = \bar{M} + \bar{P}_{UF} \, m \, \bar{P}_{UF}^{\,T}$$

where $\bar{M}$ is implicitly initialized to zero as are $\bar{K}$ and $\bar{C}$ in the sequence above. After all bar, membrane, and cell elements have been processed, routine PTMASS is called by PREEIG to add contributions from point mass elements, if any, to matrix $\bar{M}$. Finally, any null rows/columns of $\bar{M}$ are augmented on the diagonal with a value equal to 1×10^{-4} of the root mean square of all non zero diagonal elements of the matrix. Routine CHLSKY is then called to decompose $\bar{M}$. The decomposition of this matrix then remains core resident for the remainder of the run.

Input/Output

Intermediate printed output is written to file 6. Matrix data is read and written as follows:

$\bar{k}$, $\bar{c}$, $\delta\bar{e}_T$	read from N1
D	read from N3
$\bar{F}_{K_{\beta-1}}$	read from N4

DK, $\bar{k}$, FK	written to N6
$\delta\bar{F}_{Ko}$	written to N15

Argument List

None

Labeled Common

NDICES, LIMITS, TAPES

Subroutines Called

ABSYM, ADD, ASYMB, CHGSGN, CHLSKY, CØPY, EUTL9, MBAR, MULT, PBARUF, PLØP, PTMASS, READA, READM, SYMABT, WRITE, WRITEM, ZERØ

Error Detection

None

SUBROUTINE TRNMLT

This routine performs an incore transpose matrix multiplication.

Algorithm

If a cumulative summation of matrix products is not requested, the product matrix is initialized to zero. The cumulative summation of matrix products is then performed according to

$$C = C + A^T B$$

Input/Output

None

Argument List

C	A real array for the product matrix.
A	A real array for the premultiplier matrix
B	A real array for the postmultiplier matrix
L	An integer scalar defining the usable row dimension of A and B
M	An integer scalar defining the usable column dimension of B and C
N	An integer scalar defining the usable row dimension of C and the usable column dimension of A
MRØW	An integer scalar defining the row dimension of A and B
MC1	An integer scalar defining the row dimension of C and the column dimension of A
MC2	An integer scalar defining the column dimension of B and C
SUM	A logical scalar flag for cumulative summation

Labeled Common

None

Subroutines Called

None

Error Detection

None

APPENDIX F

POSTPROCESSOR ROUTINES AND LABELED COMMON BLOCKS

APPENDIX F
POSTPROCESSOR ROUTINES
AND
LABELED COMMON BLOCKS

This appendix contains detailed descriptions of all routines and labeled common blocks in this program. Table F gives either page number references within this document or references to other documents for documentation of each routine or labeled common block. Some page number references may be to preceding appendices where the documentation for a routine in this program is identical to a previously documented routine. This does not imply verbatum source code duplication for the routine, only functional duplication is implied.

The detailed description of each routine is divided into the following subheadings:

Algorithm	verbal flow chart of routine logic and data flow
Input/Output	description of all external data set input/output
Argument List	name, type, and description of each argument
Labeled Common	list of all labeled common blocks declared
Subroutines Called	list of all routines called
Error Detection	description of tests made for errors and action taken

The detailed description of each labeled common block is divided into the following subheadings:

Declaration	verbatum declaration of the labeled common block
Contents	name and description of each variable appearing in the declaration
Usage	list of all routines which contain declarations for the labeled common block

TABLE F. INDEX TO POSTPROCESSOR ROUTINES AND LABELED COMMON BLOCKS

MAIN PROGRAM PØST

This routine calls the initialization routine and the increment processing routine.

Algorithm

The FØRTRAN logical unit numbers of the three devices used and the maximum number of lines and columns per page are initialized in DATA statements.

The initialization routine INITL is called first followed by the call to the increment processing routine PØUT.

Input/Output

Error messages are output to unit 6.

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

INITL, PØUT

Error Detection

If an error occurs in either routine called, an error message is printed with the error number; i.e.,

***** ERROR nnnnnn DETECTED *****

LABELED COMMON INDEX

This common block is used to store pointers into blank common array A which are the beginning locations of partitions of array A and to store other program parameters.

Declaration

```
CØMMØN / INDEX / ITIT  ,ITIM  ,ICØR  ,IPRN  ,IRED  ,IINC  ,ISEL
                ,IELEM ,IBLHD ,IMLHD ,ICLHD ,IBLPR ,IMLPR ,ICLPR
                ,IPBUPT,IEND  ,ICLMAX,ILNMAX,ILINE ,IRØW  ,ICØL
                ,ITP1  ,ITP2  ,ITP3  ,ITVLS ,IERROR,IFLGPR,IINDEX
```

Contents

ITIT	Index of array A for title information
ITIM	Index of array A for incremental times
ICØR	Index of array A for joint coordinates
IPRN	Index of array A for print buffer
IRED	Index of array A for input record buffer
IINC	Index of array A for increment selection table
ISEL	Index of array A for joint/element selection tables
IELEM	Index of array A for element numbers for printing
IBLHD	Index of array A for bar element line headers
IMLHD	Index of array A for membrane element line headers
ICLHD	Index of array A for cell element line headers
IBLPR	Index of array A for bar element print line flags

IMLPR	Index of array A for membrane element print line flags
ICLPR	Index of array A for cell element print line flags
IPBUPT	Index of array A for matrix PBUPT partition
IEND	Index of array A of last word used
IFLGPR	Intermediate print flag for check out
ICLMAX	Maximum number of printed columns per page
ILNMAX	Maximum number printed lines per page
ILINE	Number of last line printed on a page
IRØW	Number of lines printed for type of element currently printing
ICØL	Number of columns on page currently being printed
ITP1	Number of unit which is used to store joint/element selection tables
ITP2	Number of unit where incremental solution output resides
ITP3	Number of unit used to store selected partitions of matrix PBUPT
ITVLS	Number of current BETA being processed
IERRØR	Main error flag
IINDEX	Number of words in common block INDEX

Usage

Labeled common block INDEX together with labeled common block PARM and blank common are used as a group by all principle routines.

LABELED COMMON PARM

This common block is used to store model definition data and program generated print control parameters.

Declaration

```
COMMON / PARM / NJTS   ,NCNTS  , NMATS ,NBARS ,NMEMS ,NCELS ,NPTMS
                NFDØF  ,NMØDS  , NLDJTS,NØBRC ,ABTMP ,AMCHT ,NMCØF
                NMWRD  ,NMLNG  , NMLNGM,ASTCØF,ACMCØF,NTVLS ,ADSPCF
                NSELI  ,NSELJ  , NSELB ,NSELM ,NSELC ,NPRJC ,NPRJD
                NPRJV  ,NPRJA  , NPRMD ,NPRMV ,NPRMA ,NPRBF ,NPRBS
                NPRBSS,NPRBE   , NPRMF ,NPRMS ,NPRMSS,NPRME ,NPRCF
                NPRCS  ,NPRCSS, NPRCE  ,NPARM
```

Contents

NJTS	Number of joints
NCNTS	Number of constraints
NMATS	Number of materials
NBARS	Number of bars
NMEMS	Number of membranes
NCELS	Number of cells
NPTMS	Number of point mass elements
NEDØF	Number of edge degrees of freedom
NMØDS	Number of modes
NLDJTS	Number of loaded joints
NØBRC	Number of oblique constraints
ABTMP	Base temperature
AMCHT	Mechanical equivalent of heat
NMCØF	Number of coefficients in the material property table
NMWRD	Number of descriptive words in material property tables
NMLNG	Number of words in material property record

NMLNGM	Number of words in all material property records
ASTCØF	Cell stiffness cutoff coefficient
ACMCØF	Bar compliance suppression coefficient
NTVLS	Number of time intervals
ADSPCF	Joint displacement scalar
NSELI	Selectivity flag for increments
NSELJ	Selectivity flag for joints
NSELB	Selectivity flag for bars
NSELM	Selectivity flag for membranes
NSELC	Selectivity flag for cells
NPRJC	Print option flag for joint coordinates
NPRJD	Print option flag for joint displacements
NPRJV	Print option flag for joint velocities
NPRJA	Print option flag for joint accelerations
NPRMD	Print option flag for modal displacements
NPRMV	Print option flag for modal velocities
NPRMA	Print option flag for modal accelerations
NPRBF	Print option flag for bar forces
NPRBS	Print option flag for bar stresses
NPRBSS	Print option flag for bar strains
NPRBE	Print option flag for bar equivalent stresses

NPRMF	Print option flag for membrane forces
NPRMS	Print option flag for membrane stresses
NPRMSS	Print option flag for membrane strains
NPRME	Print option flag for membrane equivalent stresses
NPRCF	Print option flag for cell forces
NPRCS	Print option flag for cell stresses
NPRCSS	Print option flag for cell strains
NPRCE	Print option flag for cell equivalent stresses
NPARM	Number of words in common blcck PARM

Usage

Labeled common block PARM together with labeled common block INDEX and blank common are used as a group by all principle routines.

SUBROUTINE BARS

This routine prints selected bar element data.

Algorithm

If no bar data exists, control is returned to the calling routine. If bar data exists but none has been requested for printing, the bar data on input tape ITP2 is read over and control returned to the calling routine.

If bar data exists and is to be printed, the bar selection table is read from scratch tape ITP1 into array A at location ISEL. The bar data is then read from input tape ITP2 one element at a time and stored in array KPRN. The element selection table is checked for the bar being processed and, if not selected, data for the next bar is read from the input tape.

The element number is stored for use as a heading on the column of KPRN when it is printed. The counter for number of columns of KPRN used is incremented and tested against the maximum number of columns specified for array KPRN which is the maximum number of columns to be printed per page. When array KPRN is full or element data is exhausted on the input tape, the data in array KPRN is printed.

If equivalent stresses are to be printed, routine EBAR is called which augments array KPRN with equivalent stresses. Line counters are checked and appropriate headers are printed using title information from array A at location ITIT, the time increment stored in array A, and the element numbers stored in array A. Using the line selection array and line header array assembled by routine BLKIN, appropriate rows of array KPRN which correspond to print lines are identified and printed with appropriate headers.

Input/Output

The bar element selection table is read from scratch tape ITP1 if appropriate. The bar element data is read from input tape ITP2. Selected bar element data is output to unit 6.

Argument List

KPRN An integer array used to store bar element data where each column contains the forces, stresses, strains, and equivalent stresses for an element

JRØW An integer scalar specifying the row dimension for array KPRN

Labeled Common

PARM, INDEX

Subroutines Called

EBAR, READC

Error Detection

None

SUBROUTINE BLKIN

This routine initializes the element line header and print flag arrays.

Algorithm

The arrays of names, numbers, and lengths associated with bar, membrane, and cell element printed data are initialized in DATA statements. Using these arrays and the four subset print option flags for each of the three elements, arrays of line headers, KHD, and arrays of line print flags, KPR, are assembled.

Input/Output

None

Argument List

KØPT An integer array of four print flags for each of three element types

KHD An alphameric array of headers for each line of output for each of three element types

KPR An integer array of print flags for each line of output for each of three element types

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE CELLS

This routine prints selected cell element data.

Algorithm

If no cell data exists, control is returned to the calling routine. If cell data exists but none has been requested for printing, the cell data on input tape ITP2 is read over and control returned to the calling routine.

If cell data exists and is to be printed, the cell selection table is read from scratch tape ITP1 into array A at location ISEL. The cell data is then read from input tape ITP2 one element at a time and stored in array KPRN. The element selection table is checked for the cell being processed and, if not selected, data for the next cell is read from the input tape.

The element number is stored for use as a heading on the column of KPRN when it is printed. The counter for number of columns of KPRN used is incremented and tested against the maximum number of columns specified for array KPRN which is the maximum number of columns to be printed per page. When array KPRN is full or element data is exhausted on the input tape, the data in array KPRN is printed.

If equivalent stresses are to be printed, routine ECEL is called which augments array KPRN with equivalent stresses. Line counters are checked and appropriate headers are printed using title information from array A at location ITIT, the time increment stored in array A, and the element numbers stored in array A. Using the line selection array and line header array assembled by routine BLKIN, appropriate rows of array KPRN which correspond to print lines are identified and printed with appropriate headers.

Input/Output

The cell element selection table is read from scratch tape ITP1 if appropriate. The cell element data is read from input tape ITP2. Selected cell element data is output to unit 6.

Argument List

KPRN An integer array used to store cell element data where each column contains the forces, stresses, strains, and equivalent stresses for an element

JRØW An integer scalar specifying the row dimension for array KPRN

Labeled Common

PARM, INDEX

Subroutines Called

ECEL, READC

Error Detection

None

SUBROUTINE CHKIN

This routine checks the logical consistency of user specified options.

Algorithm

First, a test is made to determine if items of this type exist and, if not, has a selection table been specified.

If no items of this type exist and no selection table is specified, appropriate print flags in labeled common PARM are set to zero, KSEL is set to -10 and IB(K3) is set to -1.

If items of this type exist, a test is made to determine if any subsets of this type were requested for printing. If so, IB(K3) is set to KNUM. If not, KSEL is set to -1 and IB(K3) is set to zero. Then a test is made to determine if a selection table was specified.

Input/Output

None

Argument List

KNUM   An integer scalar specifying the number of items of this type

KSEL   An integer scalar input as the user option flag indicating sets of data of this type

K1   An integer scalar indicating the first location in array IB of print flags for this type

K2   An integer scalar indicating the last location in array IB of print flags for this type

K3   An integer scalar indicating the location in array IB of the flag indicating selectivity for this type

IERRØR	An integer scalar main error flag
KØDE	An integer scalar specifying the type to be processed

Labeled Common

None

Subroutines Called

None

Error Detection

Any inconsistency between the existance or non-existance of items of the type being processed, the subset print flags specified, and the selectivity flag specified results in the main error being set to KØDE.

SUBROUTINE CRDIN

This routine reads and processes all card input.

Algorithm

The first six input cards are read and the data stored in labeled common PARM. User over-rides of default parameters are implemented as required.

The next four cards of title information are read and stored in blank common at location ITIT.

Routine SELCT is then called once for each of the following possible input selection tables; increments, joints, bars, membranes, and cells.

Input/Output

All input cards are read from unit 5 and output to unit 6.

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

SELCT, ZERØ

Error Detection

Checks are made on the sequence of all input cards and on the size of available storage verses required storage. If any error is detected, the main error flag is set and control is returned to the calling routine.

SUBROUTINE EBAR

This routine computes equivalent stresses for a bar element.

Algorithm

In the case of a bar element, the equivalent stress is equal to the absolute value of the computed stress.

Input/Output

None

Argument List

S	A real array of bar element forces, stresses, strains, and equivalent stresses
I	An integer scalar specifying the row dimension of array S
J	An integer scalar specifying the column dimension of array S or number of elements

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ECEL

This routine computes equivalent stresses for a cell element.

Algorithm

Two equivalent stresses are computed for each cell element corresponding to the upper and lower surfaces of the element. For each surface, the equivalent stress, σ_e, is computed from the average stresses, σ_x, σ_y, σ_z, τ_{xy}, τ_{yz}, and τ_{zx}, as follows:

$$S_1 = \frac{1}{3}\left(2\sigma_x - \sigma_y - \sigma_z\right)$$

$$S_2 = \frac{1}{3}\left(-\sigma_x + 2\sigma_y - \sigma_z\right)$$

$$S_3 = \frac{1}{3}\left(-\sigma_x - \sigma_y + 2\sigma_z\right)$$

$$S_4 = \tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2$$

$$\sigma_e = \left[\frac{3}{2}\left(S_1^2 + S_2^2 + S_3^2 + 2S_4\right)\right]^{\frac{1}{2}}$$

Input/Output

None

Argument List

S	A real array of cell element forces, stresses, strains, and equivalent stresses
I	An integer scalar specifying the row dimension of array S
J	An integer scalar specifying the column dimension of array S or number of elements

Labeled Common

None

Subroutine Called

None

Error Detection

None

SUBROUTINE EMEM

This routine computes equivalent stresses for a membrane element.

Algorithm

The equivalent stress, σ_e, for a membrane element is computed from the average stresses, σ_x, σ_y, and τ_{xy}, as follows:

$$S_1 = \frac{1}{3}\left(2\sigma_x - \sigma_y\right)$$

$$S_2 = \frac{1}{3}\left(-\sigma_x + 2\sigma_y\right)$$

$$S_3 = \frac{1}{3}\left(-\sigma_x - \sigma_y\right)$$

$$\sigma_e = \left[\frac{3}{2}\left(S_1^2 + S_2^2 + S_3^2 + 2\tau_{xy}^{\,2}\right)\right]^{\frac{1}{2}}$$

Input/Output

None

Argument List

S A real array of membrane element forces, stresses, strains, and equivalent stresses

I An integer scalar specifying the row dimension of array S

J An integer scalar specifying the column dimension of array S or number of elements

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE INITL

This routine is the driver routine for the initialization module.

Algorithm

After rewinding all tapes, the CØNST array is read from input tape ITP2 and stored in the beginning of labeled common block PARM. Using the problems constants so obtained, core is allocated for each of the required arrays and the first location of each is stored in labeled common block INDEX.

Routine CRDIN is then called to read all card input containing option and selectivity data. The arrays of time increments and original joint coordinates are then read from input tape ITP2 and stored in core.

Tests are then made to determine whether or not data from matrix PBUPT is required. If not, both matrix DBPØ and PBUPT are skipped over on input tape ITP2. If so, only matrix DBPØ is skipped over and then routine PBUPT is called to assemble the required data.

Finally, routine BLKIN is called to initialize element line headers and print flags.

Input/Output

Matrices CØNST, TIME, CØØRDS, DBPØ, and, if appropriate, PBUPT are read from input tape ITP2.

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

BLKIN, CRDIN, EUTL9, PBUPT, READC

Error Detection

If the main error flag is returned with a non-zero value from routine CRDIN, control is returned to the main program.

SUBROUTINE JNTPR

This routine prints selected subsets of joint data.

Algorithm

Processing is controlled by a loop indexed from one to the number of selected joints. First, the partition of matrix PBUPT for the joint being processed is read from scratch tape ITP3 and stored in array PB. An inner loop indexed from one to the number of subsets to be processed is then entered. A maximum of two subsets can be accommodated by routine JNTPR.

Argument KTYPE(J) is tested to determine which of the four possible subsets of data is to be assembled for printing.

Coordinate data is assembled into array PRN by multiplying the transpose of matrix PBUPT times the modal displacements in array A to obtain joint displacements. These displacements are then scaled by the user input displacement coefficient ADSPCF before being added to the original coordinates to form the final joint coordinates for the increment.

Displacement data is assembled in array PRN in a similar manner as coordinate data except the addition of original coordinates is omitted.

Velocity data is assembled in array PRN by multiplying the transpose of matrix PBUPT times the modal velocities in array A to obtain joint velocities.

Acceleration data is assembled in a similar manner as velocity data using the modal accelerations in array A.

Array PRN is then printed with appropriate headings as a single line of output.

Input/Output

Matrix PBUPT is read from scratch tape ITP3. Joint coordinate, displacement, velocity, and acceleration data are output to unit 6.

Argument List

KTYPE	An integer array of subset identification numbers from 1 to 4 corresponding to joint coordinates, displacements, velocities, and accelerations, respectively
KNUM	An integer scalar specifying the number of subsets to be processed
NAME	An alphameric array of subset headings
PB	A real array used to store partitions of matrix PBUPT
KMØDE	An integer scalar specifying the number of modes
CØØRD	A real array of original joint coordinates
KJTS	An integer scalar specifying the number of joints in the model

Labeled Common

PARM, INDEX

Subroutines Called

None

Error Detection

None

AD-A063 741 DOUGLAS AIRCRAFT CO LONG BEACH CALIF F/G 1/3
AIRCRAFT WINDSHIELD BIRD IMPACT MATH MODEL. PART 3. PROGRAMMING--ETC(U)
DEC 77 R C MORRIS F33615-75-C-3105
UNCLASSIFIED MDC-J-7174-PT-3 AFFDL-TR-77-99-PT-3 NL

6 OF 6
ADA
063741

END
DATE
FILMED
3 -79
DDC

SUBROUTINE JØINTS

This routine processes all joint data for an increment.

Algorithm

Column headers of two words each for each of the four subsets of data are initialized in array KEAD in DATA statements. The header for the first matrix is read from input tape ITP2 and the increment is checked for sequence. The modal response data is then read into core beginning at A(IRED).

The main joint option flag is checked to determine if any joint data is to be printed. If not, control is returned to the calling program. If so, the joint selection table is read from scratch tape ITP1 and stored in core at A(ISEL).

Each of the subset print flags is then checked. The header array NAME and other parameters are set according to the subset print flags. Routine JNTPR is then called to print the first two subsets requested. If more than two subsets were requested, routine JNTPR is called again to print the third and/or fourth subsets.

Input/Output

The modal response data is read from input tape ITP2. The joint selection table is read from scratch tape ITP1.

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

JNTPR, READC

Error Detection

The increment number from input tape ITP2 is checked for correct sequence.

SUBROUTINE MEMS

This routine prints selected membrane element data.

Algorithm

If no membrane data exists, control is returned to the calling routine. If membrane data exists but none has been requested for printing, the membrane data on input tape ITP2 is read over and control returned to the calling routine.

If membrane data exists and is to be printed, the membrane selection table is read from scratch tape ITP1 into array A at location ISEL. The membrane data is then read from input tape ITP2 one element at a time and stored in array KPRN. The element selection table is checked for the membrane being processed and, if not selected, data for the next membrane is read from the input tape.

The element number is stored for use as a heading on the column of KPRN when it is printed. The counter for number of columns of KPRN used is incremented and tested against the maximum number of columns specified for array KPRN which is the maximum number of columns to be printed per page. When array KPRN is full or element data is exhausted on the input tape, the data in array KPRN is printed.

If equivalent stresses are to be printed, routine EMEM is called which augments array KPRN with equivalent stresses. Line counters are checked and appropriate headers are printed using title information from array A at location ITIT, the time increment stored in array A, and the element numbers stored in array A. Using the line selection array and line header array assembled by routine BLKIN, appropriate rows of array KPRN which correspond to print lines are identified and printed with appropriate headers.

Input/Output

The membrane element selection table is read from scratch tape ITP1 if appropriate. The membrane element data is read from input tape ITP2. Selected membrane element data is output to unit 6.

Argument List

KPRN	An integer array used to store membrane element data where each column contains the forces, stresses, strains, and equivalent stresses for an element
JRØW	An integer scalar specifying the row dimension for array KPRN

Labeled Common

PARM, INDEX

Subroutines Called

EMEM, READC

Error Detection

None

SUBROUTINE MØDES

This routine prints selected modal data.

Algorithm

Subset print flags are tested to determine if any modal data is to be printed. If not, control is returned to the calling routine.

Each subset print flag is tested individually and the header and print arrays are initialized accordingly. Headers are stored in array NAME and modal data is stored in array KPRN.

The modal displacements, velocities, and/or accelerations are then printed from arrays NAME and KPRN with appropriate headings according to the subsets specified.

Input/Output

None

Argument List

KPRN	An integer array used to store modal displacements, velocities, and accelerations
JRØW	An integer scalar specifying the number of modes or rows of array KPRN

Labeled Common

PARM, INDEX

Subroutines Called

None

Error Detection

None

SUBROUTINE PBUPT

This routine assembles selected partitions of matrix PBUPT and writes the resulting data to scratch tape ITP3.

Algorithm

The previously assembled joint selection table is read from scratch tape ITP1 and stored in core. Input tape ITP2 is then positioned immediately following the header for matrix PBUPT.

For each of the selected joints, input tape ITP2 is read and appropriate columns of matrix PBUPT are extracted. When all three columns for a joint have been determined, the partition of PBUPT is written to scratch tape ITP3 as one record.

Since those columns of matrix PBUPT corresponding to constrained degrees of freedom are null, the reading of the input tape includes testing of column number read verses the column numbers associated with the joint being processed. It should be noted also that assembled partitions of matrix PBUPT which are written to scratch tape may contain null columns.

Input/Output

The joint selection table is read from scratch tape ITP1. Matrix PBUPT is read from input tape ITP2. Assembled partitions of matrix PBUPT are written onto scratch tape ITP3.

Argument List

JSEL An integer array used to store the joint selection table

TEMP A real array used to temporarily store input columns of matrix PBUPT

PB A real array used to assemble the three columns of matrix PBUPT corresponding to a selected joint

KG An integer scalar specifying the number of modes or rows of matrix PBUPT

Labeled Common

PARM, INDEX

Subroutines Called

EUTL9

Error Detection

None

SUBROUTINE PØUT

This routine is the driver routine for the print module.

Algorithm

For each time increment, the following takes place. The incremental selectivity flag for the current increment is tested. If equal to zero, this increment is to be bypassed and no data printed. Routine SKPALL is called to read over all data for this increment on input tape ITP2.

If not equal to zero, each of the following routines is called to process the corresponding type data from input tape ITP2; JØINTS, MØDES, BARS, MEMS, and CELLS.

Input/Output

None

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

BARS, MEMS, CELLS, JØINTS, MØDES, SKPALL

Error Detection

If the main error flag is returned with a non-zero value from any routine, control is returned to the main program.

SUBROUTINE SELCT

This routine reads selection table input cards, forms the selection table, and writes the table onto scratch tape ITP1.

Algorithm

The input argument KØDE is tested to determine which selection table is to be processed. Processing parameters are then initialized according to which table is to be processed.

Routine CHKIN is then called to check the logical consistency of input options and to return a flag indicating which one of three possible paths to follow.

If the user has requested all items of this type, then no input selection table is expected. The code then generates a selection table requesting all items and writes this table to scratch tape ITP1.

If the user has specified the selectivity option for this type, the program reads the selection table cards and assembles the selection table as so indicated. The table is then written to scratch tape ITP1.

If either no items of this type exist or no items of this are requested for printing, then no input selection table is expected and nothing is written on the scratch tape.

An exception to the above is that in the case of processing the incremental selection table, the resulting selection table is left in core and not written to the scratch tape.

Input/Output

The selection table cards present are read from unit 5. The assembled selection tables for those types so indicated by user specified options are written to scratch tape ITP1.

Argument List

JSEL	An integer array in which the selection table is formed
KØDE	An integer scalar identifying which selection table is to be processed

Labeled Common

PARM, INDEX

Subroutines Called

CHKIN

Error Detection

Checks are made for correct table number on the input cards and that the values specified are within range for this type. A test is also made to insure that a null selection table has not been specified. If any of these errors are detected, the main error flag is set to the appropriate value and control is returned to the calling routine.

SUBROUTINE SKPALL

This routine reads over all data for an increment on input tape ITP2.

Algorithm

The header of the first matrix for the increment is read from input tape ITP2. The seventh word of the header record is the increment number which is tested against the anticipated increment number to insure proper sequencing.

The input tape contains matrices only for each element type that exists in the model. Therefore, appropriate problem parameters are tested to determine which element types exist. Input tape KTP2 is then read past the number of matrix trailers corresponding to the number of matrices present.

Input/Output

Input tape ITP2 is read past all data for the increment.

Argument List

None

Labeled Common

PARM, INDEX

Subroutines Called

READC

Error Detection

The increment number of the data on input tape ITP2 is tested against the anticipated increment number. If not equal, the main error flag is set to 200 and control is returned to the calling routine.

SUBROUTINE READC

This routine reads a FØRMAT tape record.

Algorithm

The specified tape is read according to FØRMAT convention and the data stored in array A.

Input/Output

One record is read from tape ITAPE.

Argument List

ICØL	An integer scalar defining the column number
KØDE	An integer scalar indicating expanded or compressed format
NUM	An integer scalar defining the number of words remaining in the record
A	A real array used to store the data read
ITAPE	An integer scalar defining the file to be read

Labeled Common

None

Subroutines Called

None

Error Detection

None

SUBROUTINE ZERØ

This routine changes negative zero values to positive values for printing.

Algorithm

Each value in the input array is tested for a numerical zero and, if so, is set equal to a positive zero integer value.

Input/Output

None

Argument List

K	An integer array of values to be processed
N	An integer scalar equal to the length of array K

Labeled Common

None

Subroutines Called

None

Error Detection

None

REFERENCES

1. J. P. Cogan, Jr., FORMAT II - Second Version of Fortran Matrix Abstraction Technique; Volume II, Description of Digital Computer Program, AFFDL-TR-66-207, Volume II, Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base, Ohio, March 1967.

2. W. J. Lackey, R. E. Wild, FORMAT II - Second Version of Fortran Matrix Abstraction Technique; Volume II, Supplement I. Description of Digital Computer Program, AFFDL-TR-66-207, Volume II, Supplement I, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, November 1968.

3. C. G. Hooks, FORMAT II - Second Version of Fortran Matrix Abstraction Technique; Volume II, Supplement II. Description of Digital Computer Program System/360, AFFDL-TR-66-207, Volume II, Supplement II, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, February 1969.

4. W. J. Lackey, S. H. Miyawaki, FORMAT - Fortran Matrix Abstraction Technique; Volume II, Supplement III. Description of Digital Computer Program - Extended, AFFDL-TR-66-207, Volume II, Supplement III, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, June 1970.

5. L. Chahinian, S. H. Miyawaki, FORMAT - Fortran Matrix Abstraction Technique; Volume II, Supplement IV. Description of Digital Computer Program - Extended, AFFDL-TR-66-207, Volume II, Supplement IV, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, April 1973.

6. L. Chahinian,FORMAT - Fortran Matrix Abstraction Techniques; Volume II, Supplement V. Description of Digital Computer Program - Extended, AFFDL-TR-66-207, Volume II, Supplement V, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, December 1977.

7. J. Pickard, FORMAT - Fortran Matrix Abstraction Technique; Volume V, Engineering User and Technical Report, AFFDL-TR-66-207, Volume V, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio October 1968.

8. J. Pickard, FORMAT - Fortran Matrix Abstraction Technique; Volume V, Supplement I. Engineering User and Technical Report - Extended, AFFDL-TR-66-207, Volume V, Supplement I, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, June 1970.

9. J. Pickard,FORMAT - Fortran Matrix Abstraction Technique; Volume V, Supplement II. Engineering User and Technical Report - Extended, AFFDL-TR-66-207, Volume V, Supplement II, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, April 1973.

REFERENCES (Continued)

10. J. Pickard, FORMAT - Fortran Matrix Abstraction Technique; Volume V, Supplement III. Engineering User and Technical Report - Extended, AFFDL-TR-66-207, Volume V, Supplement III, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, December 1977.

11. J. P. Cogan, Jr., R. C. Morris, and J. R. Wells, FORMAT - Fortran Matrix Abstraction Technique; Volume VI. Description of Digital Computer Program - Phase I, AFFDL-TR-66-207, Volume VI, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, September 1968.

12. R. C. Morris, FORMAT - Fortran Matrix Abstraction Technique; Volume VI, Supplement I. Description of Digital Computer Program - Phase I - Extended, AFFDL-TR-66-207, Volume VI, Supplement I, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, June 1970.

13. R. C. Morris, FORMAT - Fortran Matrix Abstraction Technique; Volume VI, Supplement VI, Supplment II. Description of Digital Computer Program - Phase I - Extended, AFFDL-TR-66-207, Volume VI, Supplement II, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, April 1973.

14. R. C. Morris, J. R. Wells, and P. S. Yoon, FORMAT - Fortran Matrix Abstraction Technique; Volume VII. Description of Digital Computer Program - Phase III, AFFDL-TR-66-207, Volume VII, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, September 1968.

15. J. A. Frank, FORMAT - Fortran Matrix Abstraction Technique; Volume VII, Supplement I. Description of Digital Computer Program - Phase III - Extended, AFFDL-TR-66-207, Volume VII, Supplement I, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio, June 1970.

16. B. T. Smith, et al., Matrix Eigensystem Routines: EISPACK Guide, Springer-Verlag, New York, New York, 1974.

17. D. E. Knuth, The Art of Computer Programming; Volume III, Sorting and Searching, Addison and Wesley, Reading, Massachusetts, 1973.