REPORT NO.
UCB/SEMM-92/28

STRUCTURAL ENGINEERING
MECHANICS AND MATERIALS

DYNAMIC RESPONSE ANALYSIS OF INELASTIC BUILDING STRUCTURES: The DRAIN Series of Computer Programs

BY

VIPUL PRAKASH

Dissertation Committee Chair: G. H. Powell

DECEMBER 1992

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

Dynamic Response Analysis of Inelastic Building Structures: The DRAIN Series of Computer Programs

by

Vipul Prakash

B.E. Honours (University of Roorkee, Roorkee) 1984
M.S. (University of California at Berkeley) 1985

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy
in
Engineering-Civil Engineering
in the
GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Graham H. Powell
Professor Filippos C. Filippou
Professor James W. Demmel

1992

The Dissertation of Vipul Prakash is Approved:

[illegible] 12/14/92

Chair Date

[illegible] 12/14/92

Date

[illegible] 12/15/92

Date

University of California at Berkeley

1992

Dynamic Response Analysis of Inelastic Building Structures: The DRAIN Series of Computer Programs

by

Vipul Prakash

Abstract

Dynamic Response Analysis of Inelastic Building Structures:

The DRAIN Series of Computer Programs

by

Vipul Prakash

Doctor of Philosophy in Civil Engineering

University of California at Berkeley

Professor Graham H. Powell, Chair

A family of three computer programs has been developed for the analysis of nonlinear building structures. These programs are DRAIN-2DX, DRAIN-3DX and DRAIN-BUILDING.

DRAIN-2DX is for two-dimensional structures, and is an extension of the well-established DRAIN-2D program. DRAIN-3DX is essentially a three-dimensional version of DRAIN-2DX. DRAIN-BUILDING is specifically for tall buildings. In this program a structure is modeled as an assemblage of floors connected by "interfloors" consisting of columns, walls and braces. The structure stiffness matrix is partitioned into floor and interfloor blocks, and a hypermatrix storage scheme and solver are used.

Each program consists of a base program plus a set of subroutines for each element type. Only a few simple elements are currently available, and additional elements are needed. The procedures for adding new elements are well defined. It is hoped that other researchers will find that the DRAIN programs are suitable platforms for developing new elements.

Static and dynamic analyses are based on an event-to-event strategy, with modification of the structure stiffness matrix at each event. For dynamic analysis a variable time step scheme can be used; and in-phase ground accelerations, out-of-phase ground displacements, or imposed dynamic loads can be considered. Energy balance computations are performed, identifying the static work, the damping work, the kinetic energy, and the input energy. Static and dynamic loads can be applied in any sequence. The structure state can be saved at the end of any analysis, and the analysis can be restarted from any saved state.

The programs are written in FORTRAN-77, and have been compiled and executed on PCs under DOS (using overlays), on PCs under WINDOWS, and on UNIX workstations.

This thesis documents the program design. It describes the data structures and data management schemes, and gives flow charts and lists of tasks done for each phase of the program. It also describes the special modeling, data management and equation solving procedure for DRAIN-BUILDING.

[illegible]

Graham H. Powell

ACKNOWLEDGMENTS

The research reported herein has been supported by the Kajima Corporation of Japan under CUREE-KAJIMA Grant No. 442408-57210. I wish to thank Kajima Corporation for providing the funds that made this research possible, and in particular Dr. Takashi Miyashita who supervised the project.

I am grateful to Professor Graham H. Powell for the guidance, insights and encouragement throughout the course of this research. I thank Professor Filippos Filippou for guiding me in the use of PC programs for word processing and graphics applications. I thank Professors James W. Demmel and Beresford N. Parlett for advice during the planning of the hypermatrix equation solver for the program DRAIN-BUILDING.

I thank the University of Roorkee, and in particular Dr. Bhagwat V. K. Lavania, for allowing me to take study leave to pursue the Ph.D. degree at the University of California at Berkeley.

Many friends have been generous in giving me encouragement. I particularly wish to thank Swami Aparanand, Dr. Anjur R. Chandrasekaran, Dr. Hari Kishan, Mr. Ashok D. Pandey, Dr. Alok Goyal, Dr. Rakesh Goel, Mr. Sajid Abbas, Mr. Allahnawaz Quaisrani, Mr. Graham Archer, Mr. Scott Campbell and Mr. Satinder Singh.

I am grateful to my father Dr. Anand Prakash for the love, inspiration, and prayers on my behalf for my Ph.D. from UC Berkeley. I am grateful to my mother, Mrs. Sushma Prakash for the love and strength she has provided me during my successes and failures alike. I am grateful to Dr. Vipin Garg, Mrs. Nandita Garg, Dr. Atul Prakash, and Mrs. Anvita Prakash for providing the love and support to me and family during our stay in the US. I thank my wife, Sandhya, for bearing my absence during the first year of my stay in Berkeley while she gave birth to our son, Divyam, in India.

Vipul Prakash

Dedicated to my parents

Mrs. Sushma Prakash

and

Dr. Anand Prakash

TABLE OF CONTENTS

1. INTRODUCTION

1.1 THE DRAIN PROGRAMS

The computer program DRAIN-2D (Dynamic Response Analysis of INelastic 2-Dimensional Structures) was first released in 1973 [10], and has been a useful analysis tool for many years. The main advantage of the program is that it is both simple and effective. Its capabilities, however, are very limited. Two extended versions of the program have been developed over the years, namely DRAIN-2D2 by Golafshani in 1983 [7] and DRAIN-2DX by Allahabadi in 1987 [1, 2]. These versions added more powerful capabilities to the program, while retaining its essential simplicity. They were not released because time and resources were not available to debug them and make them robust enough for general use.

Using DRAIN-2DX as a starting point, a family of three new computer programs has been developed in the present phase of the work. These programs are identified as DRAIN-2DX [11], DRAIN-3DX [12] and DRAIN-BUILDING [13]. DRAIN-2DX and DRAIN-3DX are for general two dimensional (2D) and three dimensional (3D) structures, respectively. DRAIN-BUILDING is specifically for 3D building structures. The programs are based on the same technology and have similar features, so that they are clearly members of a family. Although they are more complex than DRAIN-2D, they are simpler than other general purpose nonlinear analysis programs. They also provide a number of features that are important for nonlinear seismic analysis, yet are not provided by other programs.

As in DRAIN-2D, each of the programs consists of a "base" program which manages the data and controls the analysis, plus a set of subroutines for each element type which control the element details. Information is transferred between the base program and the elements through an interface that is the same for all element types. The base program knows nothing about the elements except what is transferred through the interface. This is an essential design requirement for a general purpose structural analysis program. Among other things, it allows new elements to be added to the element library without changing the base program. Because the new programs have more capabilities than DRAIN-2D, it is more difficult to develop new elements. However, the process is well defined, and is still relatively simple.

The programs are written in FORTRAN-77. They have been compiled and executed on PCs by using the Lahey compiler and PLINK overlay linker under DOS, the Lahey extended memory compiler under DOS, the Microsoft compiler under WINDOWS, and the Microsoft compiler under OS2. They have been compiled on UNIX workstations using the f77 compiler. There have been reliability problems with the Microsoft WINDOWS and OS2 versions. It is not known whether this is due to coding errors or to problems in the operating systems.

With DRAIN-2DX and DRAIN-BUILDING, structures of moderate size can be analyzed under DOS with a 640k limit, but for large structures it is necessary to use a workstation. Only small structures can currently be analyzed with DRAIN-3DX under DOS, mainly because the program uses an in-core equation solver.

1.2. GENERAL FEATURES OF THE DRAIN PROGRAMS

DRAIN-2D permitted only linear static analysis followed by nonlinear dynamic analysis, and considered only dynamic loads due to in-phase ground accelerations. The new programs perform nonlinear static and dynamic analyses, and for dynamic analysis consider ground accelerations (all supports moving in phase), ground displacements (supports may move out of phase), imposed dynamic loads (e.g., wind), and specified initial velocities (e.g., impulse loading). Static and dynamic loads can be applied in any sequence. For example, a dynamic analysis can be performed to damage the structure, and static loads can then be applied to investigate its behavior in the damaged state. If a static load follows a dynamic load, a special "restore to static state" (REST) analysis is performed to bring the structure to rest before the static load is applied.

The structure state can be saved at the end of any analysis, and the analysis can be restarted from any saved state. For example, the state at the end of a static analysis can be saved. Analyses for several different ground motions can then be restarted from this state, to study the effects of different earthquakes. The unstressed state is automatically saved. Hence, the input data defining the structure can be processed in one computer run, and the input data files for subsequent runs need only define the loading details.

The step-by-step integration scheme for dynamic analysis varies the time step during the analysis, on the basis of input error tolerances. This option is particularly useful if pounding or gap closing occurs, since a small time step is needed to obtain accurate results for a short period of time after a gap closes, but a longer step can usually be used for most of the analysis.

Energy balance computations are performed, identifying the static work (mainly hysteretic losses), the energy absorbed by viscous damping, the kinetic energy, and the input energy. The energy breakdown can be valuable for studying structural response. Also a substantial energy unbalance indicates that the analysis has not been performed correctly.

Mode shapes and periods can be calculated for any state. Linear response spectrum analyses can be performed for the unstressed state.

Static nonlinear analysis is performed by an event-to-event scheme, where each event corresponds to a significant change in stiffness. There is currently no provision for iteration (although this could be added), mainly because iteration schemes tend to be unreliable and difficult to specify. The event-to-event scheme is simpler and more reliable, and also permits a detailed energy balance to be calculated. However, it can require more computer time, and it may be difficult to define events for elements which have curvilinear action-deformation relationships. To reduce execution time, provision is made for event overshoot tolerances to be specified, so that the structure stiffness is not modified at each exact event but at a somewhat larger load. This allows more than one element to change its stiffness at each event, and reduces the number of times the stiffness matrix must be modified. However, overshoot also causes equilibrium unbalances, and the larger the overshoot tolerance the larger the unbalance. Provision is made for the overshoot tolerances to be changed without reprocessing the input data. Hence, rough static analyses can be performed by specifying large tolerances in the early stages of an investigation, followed by more accurate analyses later with small tolerances.

Static element loads, as well as nodal loads, can be specified, but the present versions of the programs have serious limitations. In particular, nonlinear behavior is not permitted if there are element loads. The reason for this is that element loads can, in principle, take any form. Element loads, unlike nodal loads, can not be generalized, and hence can not be processed at the base program level. Instead, they must be processed at the element level, with a generalized interface between the base program and the element. If an element load causes nonlinear behavior, this behavior must be recognized and accounted for at the element level, and information must be sent to the base program so that it can control the nonlinear solution strategy. In the present versions of the programs, the interface between the base program and the elements is not flexible enough to allow for nonlinear behavior under element loads.

Rigid link slaving can be specified. In DRAIN-3DX and DRAIN-BUILDING this allows rigid floor diaphragms to be defined.

Envelope results (maximum effects) and/or time history results can be output to a print file. Time history results are ordered node-by-node and element-by-element (i.e., a time history for one node followed by the history for the next node, etc.). Time histories can be bulky, however, and can lead to very large output files. Results can also be written to a (binary) post-processing file. A preliminary version of a post-processor, with a windows-type graphic interface, is being developed.

A structure geometry file is written, from which plots of structure geometry can be made. A preliminary geometry plotting program is also being developed. The lack of geometry plotting is a particular weakness, making it difficult to check the input data. It should be noted that geometry plotting is not a simple task, because of the many different element types that are possible. If an analysis model consists only of line and/or solid elements, as is the case in most programs, geometry plotting is relatively easy. A DRAIN model can consist of a wider variety of element types, particularly point (zero length) elements for modeling connections. Also, to model complex nonlinear behavior it may be necessary to place elements in parallel, so that there may be several elements connecting any given pair of nodes. It is necessary, therefore, to plot point elements as well as line and solid elements, and to indicate when elements are placed in parallel. Procedures for programmers to follow when adding new element types must also be devised and documented.

Cross sections can be specified through the structure, and the resultant normal, shear and overturning effects on these sections can be output. Relative displacements (e.g., story-to-story drifts) can also be output.

The programs all require formatted input, which is read from an input file. This is admittedly antiquated. The input format is, however, much more flexible than the DRAIN-2D format. Different sections of the data are now separated by key words; it is no longer necessary to specify the numbers of nodes, elements, etc.; comments can be added to the data; and the nodes need not be numbered sequentially. This last feature simplifies data preparation by allowing nodes to be laid out on a grid, and numbered according to the grid intersections.

1.3. SPECIAL FEATURES FOR DRAIN-BUILDING

In DRAIN-BUILDING the structure is modeled as a series of floors separated by "interfloors" consisting of columns, walls and braces. To define a structure, floor and interfloor types are first defined. Actual floors (floor instances) are then located in space, by specifying the floor type and location for each instance. Finally, interfloor instances are specified between the floors. The building geometry is not limited to simple towers. Mezzanines and atriums can be specified if desired, and multiple towers can be connected by floors which bridge between them.

This procedure simplifies the task of defining a building. In addition, the floor-interfloor description is reflected in the way the analysis data is organized within the program. For each floor and each interfloor a separate data structure is set up, preserving them as distinct entities. Also, the structure stiffness matrix is partitioned into floor and interfloor stiffnesses, in "hypermatrix" form, and a hypermatrix equation solver is used [5]. This provides an efficient blocking scheme for the structure stiffness matrix, and allows large structures to be analyzed with modest storage requirements.

1.4. NONLINEAR ELEMENTS

DRAIN-2D had a variety of elements for modeling beams, columns, panels and connections. All of these were based on very simple models. Some of these elements have been modified, and in some cases extended, for DRAIN-2DX. The present library of elements for this program includes (1) a truss bar, (2) a plastic hinge beam-column, (3) a zero-length connection with options for translational or rotational connection and a choice of three different hysteresis loops, (4) a bar which can act as an inelastic gap element or an inelastic cable which goes slack, and (5) an elastic rectangular panel. The beam-column element has a particular weakness, since it does not account correctly for P-M interaction. An improved beam-column element based on fiber concepts is being developed.

DRAIN-3DX and DRAIN-BUILDING share the same elements. However, 3D elements are inherently much more complex than 2D elements, and this complicates the task of providing a useful library of 3D elements. However, the procedures for adding new elements to the programs are logical and well defined. It is expected that as time progresses other researchers will also add new elements to the programs. It must be recognized, however, that programming an error-free element is vastly more complex for a nonlinear element than for a linear one.

1.5. OBJECTIVES AND SCOPE

The primary objective of the work described herein has been to develop reliable and robust versions of all three DRAIN programs.

The objective of the first phase of the work was to improve DRAIN-2DX, by allowing the input data to be separated by key words; providing extensive data checking and diagnostics; improving memory management; and implementing ground displacement, dynamic force, and initial velocity analyses. Although Allahabadi's 1987 version was the starting point, the program was almost completely rewritten.

The objective of the second phase was to develop DRAIN-3DX, based on the improved DRAIN-2DX. This objective was achieved, although the program is currently limited to small structures.

The objective of the final phase was to develop DRAIN-BUILDING, based on DRAIN-3DX, for practical analysis of 3D building structures. DRAIN-BUILDING uses data structures based on floors and interfloors, and an out-of-core hypermatrix equation solver. This objective was achieved, although several needed improvements in the program have been identified.

The author has had primary responsibility for developing the base programs for DRAIN-2DX, DRAIN-3DX and DRAIN-BUILDING. Full details of the programs are contained in several reports. The scope of the present report is limited to documenting the program design. Other reports include (a) user guides for the three programs, (b) examples to guide users in developing nonlinear models and performing analyses, (c) programming instructions for developers of new elements, (d) static and dynamic analysis theory, and (e) modeling guidelines for developers of new elements.

1.6. REPORT LAYOUT

Chapter 2 describes in detail the data structures and data management schemes used in DRAIN-2DX and 3DX. It also gives an overview of the disk files created by these programs.

Chapter 3 describes the base program phases, gives flow charts and lists the tasks done in each phase for DRAIN-2DX and 3DX.

Chapter 4 describes in detail the structures of the permanent disk files that are created by DRAIN-2DX and 3DX. Some of these files may be used by other programs, particularly for post-processing.

Chapter 5 describes how a DRAIN-BUILDING model is constructed by using floors and interfloors, the resulting hypermatrix structure of the stiffness matrix, the hypermatrix storage scheme used, and the hypermatrix algorithms that have been developed.

The data structures and data management schemes for DRAIN-BUILDING are similar to those for DRAIN-2DX and 3DX. Chapter 6 describes the changes for DRAIN-BUILDING.

The base program phases, flow charts and lists of tasks done for DRAIN-BUILDING are also similar to those for DRAIN-2DX and 3DX. Chapter 7 describes the base program phases and flow charts for each phase, with emphasis on the phases that are different for DRAIN-BUILDING.

Chapter 8 describes in detail the structures of the permanent disk files that are created by DRAIN-BUILDING.

Chapter 9 describes the interface between the base programs and the element subroutines. This chapter also lists the tasks performed in each element subroutine and describes the procedures to be followed for adding new elements.

Chapter 10 identifies several improvements and extensions that are possible in future versions of the programs, and concludes this report.

2. DATA MANAGEMENT -- DRAIN-2DX AND -3DX

2.1. INTRODUCTION

The DRAIN programs store data in memory and on a number of permanent and temporary (scratch) files. In memory, data that can be assigned fixed dimensions is stored in labeled common blocks, and data with dimensions which must be assigned based on the problem size is stored in blank common. This chapter describes the memory management in detail, and gives an overview of the disk files. Details of the disk files are presented in Chapter 4.

2.2. LABELED COMMONS

2.2.1. Introduction

The labeled common blocks are organized so that each block contains related information. In this section the contents and use of each labeled common block are described in alphabetical order.

Some labeled common blocks are used to transfer information between the base program and element subroutines. These blocks are part of the element interface, and are used by both the base program and the element subroutines. Other blocks are used by the base program only, and are not available in the element subroutines.

2.2.2. ASDFRC Common

For dynamic analysis, ground acceleration records, ground displacement records and dynamic force records must be set up in memory. These will be termed ***dynamic load records.*** These records can be very long, and it may be necessary to consider several records in a single analysis. In DRAIN, provision is made for specification of up to 36 records for ground displacement or dynamic force analysis. Thus, it is impractical to store the complete records in memory. The /ASDFRC/ block provides a fixed length buffer, TASDF, which is used to store those parts of each record which are currently being applied to the structure. This buffer is also used to store response spectra for response spectrum analysis.

In the input phase, the dynamic load records are read from the input file, divided into fixed length blocks, and written to permanent files with extensions .ACC, .DIS and .FRC. On these files, each block contains MPAIRS time-value pairs, where MPAIRS is in /LOADP/ and is currently set to 121. The last pair in any block is the same as the first pair in the next block, so that each block contains 120 new pairs. There is no limit on the number of blocks for any dynamic load record.

At the beginning of a dynamic analysis, blocks of those dynamic load records that are to be used in the analysis are read from the permanent file (using unit NFLOAD in /TAPES/) and re-blocked to a scratch file (using unit NFASDF in /TAPES/). A block from NFLOAD (with 121 time-value pairs) may be re-blocked into smaller size blocks before being written to NFASDF, as explained below. The number of NFASDF blocks can be different for each applied dynamic load, and the length of time covered by a block can vary from load to load and from block to block.

The number of time-value pairs in each NFASDF block is chosen so that one block for each applied dynamic load can be held in the buffer TASDF in /ASDFRC/. The length currently assigned to TASDF is 1512 words. The block size depends on the number of applied dynamic loads, as shown in following table.

No. of Dynamic Loads (NDFREC in /ADFREC/)	No. of Data-Pairs per Block (NPAIRS in /ADFREC)
1-6	121
7-12	61
13-18	41
19-24	31
25-30	25
31-36	21

At any time, only one block for each dynamic load is held in TASDF. Each dynamic load record is monitored as the analysis proceeds, and when the current time exceeds the maximum time for any block, the next block for that load is read from NFASDF, replacing the existing block.

/ASDFRC/ is declared as follows.

```
COMMON /ASDFRC/ tasdf(1512), inbl(36), lnbl(36), nbl(36), nt(36), icnbl
```

The variables are as follows.

Variable	Description
TASDF	Buffer for time-value pairs.
INBL	Block number on NFASDF of the first block for each dynamic load.
LNBL	Block number on NFASDF of the last block for each dynamic load.
NBL	Block number on NFASDF of the block currently in TASDF for each dynamic load.
NT	Interval containing current time for each block in TASDF. The current time lies between the times for pairs NT and NT+1.
ICNBL	Last block number read from NFASDF.

When the current time exceeds the maximum time for any block in TASDF, the next block (block NBL(i)+1 for dynamic load i) is read from NFASDF to replace the existing block.

ICNBL is use to determine whether NFASDF(a sequential file) must be rewound before skipping to the required block.

2.2.3. ADFREC Common

/ADFREC/ stores information on how each dynaimc load record is applied to the structure. For ground acceleration loading, accelerations are applied at all restrained and spring-supported DOFs. For a ground displacement analysis, displacements are applied at all spring-supported DOFs. For a dynamic force analysis, forces can be applied at any unrestrained DOF. The maximum number of DOFs at which any dynamic load record can be applied is the parameter NDSFRP, which can be changed if desired.

/ADFREC/ is declared as follows.

```
PARAMETER (NDSFRP=100)
COMMON /ADFREC/ tfac(36), recfac(36), dtime(36), axi(36), axe(36),
1                corot(2), dffac(NDSFRP), jdfrec(2,NDSFRP),
2                npadf(36), ndsfr, ndfrec, npairs
```

The variables are as follows.

Variable	Description
NDSFRP	Maximum number of DOFs at which a ground displacement or dynamic force record can be applied.
TFAC	Time scale factor for each dynamic load record.
RECFAC	Acceleration, displacement or force scale factor for each record.
DTIME	Time delay for each record (for ground displacement or dynamic force analysis only).
AXI	Acceleration, displacement or force for each record at beginning of current time step.
AXE	Acceleration, displacement or force for each record at end of current time step.
COROT	Coordinates for center of rotation for rotational ground acceleration. This array is COROT(3) in DRAIN-3DX.

DFFAC	Scale factor for displacement or force for each loaded DOF. This factor is applied after the displacement or force value has been scaled by RECFAC.
JDFREC	DOFs at which ground displacement or dynamic force records are applied. 1: Equation number for the DOF. 2: Dynamic load record number or direction (see note). **Note :** For ground displacement or dynamic force analysis, this is the order in which the load record was input. For ground acceleration analysis, 1 = global X direction, 2 = Y direction, etc.
NPADF	First-word-address in TASDF buffer for each load record block (see /ASDFRC/).
NDSFR	Number of DOFs subjected to ground displacement or dynamic force records. Must be less than NDSFRP.
NDFREC	Number of displacement or force records. Must be less than 36.
NPAIRS	Number of time-value pairs per NFASDF block (see /ASDFRC/).

2.2.4. AUTO Common

/AUTO/ stores time step data for dynamic analysis.

/AUTO/ is declared as follows.

```
COMMON /AUTO/ dtauto, dtcons, dtmax, dtmin, dtnew, dtold, dtred, dtinc,
1             erri, errs, tolhii, tolhis, tolloi, tollos, tolmx,
2             nsinc, insinc, modify, issav
```

The variables are as follows.

Variable	**Description**
DTAUTO	Default initial time step for variable time step scheme.
DTCONS	Default time step for constant time step scheme.
DTMAX	Maximum allowable time step size.
DTMIN	Minimum allowable time step size.
DTNEW	New time step size.
DTOLD	Old time step size.

DTRED	Time step reduction factor.
DTINC	Time step increase factor.
ERRI	Inertia force (equivalent impulse) error for current time-step.
ERRS	Static force (midstep equilibrium) error for current time-step.
TOLHII	Upper inertia force (equivalent impulse) error tolerance.
TOLHIS	Upper static force (midstep equilibrium) error tolerance.
TOLLOI	Lower inertia force (equivalent impulse) error tolerance.
TOLLOS	Lower static force (midstep equilibrium) error tolerance.
TOLMX	Maximum static/inertia force error tolerance. Analysis quits if exceeded.
NSINC	Number of consecutive steps below TOLLOS and TOLLOI, after which time step size is increased.
INSINC	Counter for the number of consecutive steps below TOLLOS and TOLLOI.
MODIFY	Code for change in time-step size in the current analysis step (set during the dynamic analysis), as follows. -1 : The step must be repeated with a reduced time step size. 0 : No change in time step size. 1 : Time step size to be increased in next step. 2 : Use initial value for time step size. If MODIFY $\neq$ 0, the effective stiffness is reformed before the next step.
ISSAV	Code for backing up structure state, in case time step must be repeated. The element /INFGR/ and /INFEL/ blocks, the static tangent stiffness and the unbalanced load at the beginning of the current step are saved. To avoid unnecessary saving if there are no events, the state is saved at the end of the first substep, just before the state is updated. 0 : State not yet backed up. 1 : State has been backed up.

2.2.5. CLINE Common

/CLINE/ is used to hold the current line from the input data file. Each input line is read into the character variable XXLINE, using one of the utility subroutines GETLIN, GTLIN or GTLINE. XXLINE is then used as a FORTRAN internal file.

/CLINE/ is declared as follows.

```
COMMON /CLINE/ lecho, linpx, xxline
CHARACTER lecho*1, linpx*1, xxline*161
```

The variables are as follows.

Variable	Description
LECHO	Code for echoing input line on the monitor, as follows. "n" : No. "y" : Yes.
LINPX	Code for reading input line, as follows. "i" : Read input line from input data file, DRAIN.INP. "x" : Read input line from scratch file INPX. INPX is used when it is necessary to count the number of input items before allocating memory. The input is read from DRAIN.INP, written to INPX, then re-read from INPX after memory has been allocated.
XXLINE	For reading from DRAIN.INP: columns 1-80 : Input buffer. column 81 : Set to "/" to terminate record. For reading from a data file containing ground acceleration, ground displacement or dynamic force records: columns 1-160 : Input buffer. column 161 : Set to "/" to terminate record.

2.2.6. COLPSE Common

/COLPSE/ stores the nodal displacements for which collapse of the structure is assumed.

/COLPSE/ is declared as follows.

```
COMMON /COLPSE/ dismax(2), rtnmax(2)
```

The variables are as follows.

Variable	Description
DISMAX	Nodal displacement at which collapse is indicated (analysis quits if exceeded). 1 : use for static analysis. 2 : use for dynamic analysis.
RTNMAX	Nodal rotation at which collapse is indicated (analysis quits if exceeded). 1 : use for static analysis. 2 : use for dynamic analysis.

2.2.7. CONTR Common

/CONTR/ stores overall control information.

/CONTR/ is declared as follows.

```
 COMMON /CONTR/ incor, nchar, ndsp, ndtp, nelg, nelgr, neltot, nnods,
1                          nseg, nsnds, ntnds
```

The variables are as follows.

Variable	Description
INCOR	Code for storage of backup copies of /INFEL/ and /INFGR/ blocks, tangent stiffness and unbalanced load. When the variable time step scheme is used, a backup copy of the structure state is stored in case the current time step must be repeated. 0 : NBLOK > 1 (see /STOR/ block). State is always backed up on a scratch file in this case. 1 : Back up on file. 2 : Back up /INFEL/ and /INFGR/ blocks on file. Back up tangent stiffness and unbalanced load in memory. 3 : Back up entirely in memory.
NCHAR	Number of characters in FNAME (see /TITLE/). Used for opening permanent files.
NDSP	Number of support springs.

NDTP	Number of compound node types + 1.
NELG	Number of element groups with nonzero /INFGR/ blocks.
NELGR	Number of element groups.
NELTOT	Total number of elements.
NNODS	Total number of nodes.
NSEG	Current analysis segment number.
NSNDS	Total number of subnodes in all compound node types.
NTNDS	Total number of nodes and subnodes.

2.2.8. CURRNT Common

For any element task (e.g., during model definition, event factor calculation, stiffness formation, state determination, etc.), an element subroutine is called for each element. /CURRNT/ stores information on the current element.

/CURRNT/ is declared as follows.

```
COMMON /CURRNT/ igrc, ielc, idfc, inodc
```

The variables are as follows.

Variable	Description
IGRC	Current element group number.
IELC	Current element number in current group.
IDFC	Current element DOF number in current element. Used only in the model definition phase (see Section 3.3c).
INODC	Current element node number in current element. Used only in the model definition phase (see Section 3.3c).

2.2.9. DAMPG Common

/DAMPG/ stores factors for alpha-M and beta-K damping.

/DAMPG/ is declared as follows.

```
COMMON /DAMPG/ alpha, beta, kalpm, kbeta
```

The variables are as follows.

Variable	Description
ALPHA	Scale factor for alpha-M damping.
BETA	Scale factor for beta-K damping.
KALPM	Code for existence of alpha-M damping (0:No ; 1:Yes).
KBETA	Code for existence of beta-K damping (0:No ; 1:Yes).

2.2.10. DIMENS Common

Array dimensions may be passed through argument lists or common blocks. If a dimension is 0, the FORTRAN compiler may treat it as a fatal error even though the array is never used (e.g., the Lahey F77L compiler). /DIMENS/ stores dummy values for dimensions that may be zero.

/DIMENS/ is declared as follows.

```
COMMON /DIMENS/ mxcutd, mxtdfd, ndspd, nrdsd, nssecd, ntrnsd
```

The variables are as follows.

Variable	Description
MXCUTD	MAX(1,MAXCUT), where MAXCUT is maximum number of elements cut by any section (see /SECTON/).
MXTDFD	MAX(1,MAXTDF), where MAXTDF is maximum number of DOFs for any cut element (see /SECTON/).
NDSPD	MAX(1,NDSP), where NDSP is the number of support springs (see /CONTR/).
NRDSD	MAX(1,NRDS), where NRDS is the number of generalized displacements (see /SETREL/).
NSSECD	MAX(1,NSSEC), where NSSEC is the number of structure sections (see /SECTON/).
NTRNSD	MAX(1,NTRNS), where NTRNS is the number of force transformations for structure sections (see /SECTON/).

2.2.11. DISVEL Common

Element end displacements and velocities are sent to the element subroutines for certain element tasks (e.g., event factor calculation, state determination), and element end forces are returned. /DISVEL/ provides memory for these values. Currently these values are transferred through argument lists. In future programs, /DISVEL/ may be used to transfer these values, in which case /DISVEL/ will become a part of the interface between the base program and the element subroutines.

The maximum number of DOFs for any element is currently 30.

/DISVEL/ is declared as follows.

```
 COMMON /DISVEL/ ddise(30), vele(30), dise(30),
1                relas(30), rdamp(30), rinit(30)
```

The variables are as follows.

Variable	Description
DDISE	Increment in element end displacements.
VELE	Element end velocities.
DISE	Element end displacements.
RELAS	Element end static forces.
RDAMP	Element end damping forces.
RINIT	Element end initial forces, due to element loads applied in static gravity analysis.

2.2.12. ELMPAR Common

Up to 2 integer and 2 real analysis control parameters can be input for each element group, and can be changed before any analysis segment. These parameters might be used, for example, to set flags for printing debugging information. The use of these parameters depends on the element type. /ELMPAR/ stores the parameters for the element group that is currently being processed.

/ELMPAR/ is declared as follows.

```
COMMON /ELMPAR/ relpar(2), ielpar(2)
```

The variables are as follows.

Variable	Description
RELPAR	Integer element parameters for current element group.
IELPAR	Real element parameters for current element group.

2.2.13. ENRGY Common

/ENRGY/ stores the work quantities for checking energy balance. In addition to the current values, the values at the beginning of the current time step are also stored. These are backup values in case the time step must be repeated.

The maximum number of element groups is currently 20. This limit also affects /GENINF/.

/ENRGY/ is declared as follows.

```
COMMON /ENRGY/ tek, tei, ted, tes, tew, tep, teso, ten, tee, tead,
1              teki, teii, tedi, tesi, tewi, tepi, tesoi, teni, teei, teadi,
2              eneg(20), enrd(20), eext(20),
3              enegi(20), enrdi(20), eexti(20)
```

The variables are as follows.

Variable	Description
TEK	Total kinetic energy (from nodal masses and velocities).
TEI	Total inertia work (work done by inertia forces).
TED	Total damping work (TEAD+TEBD).
TES	Total element static work.

TEW	Total work done by dynamic nodal loads (consists of work done by effective loads in ground acceleration analysis and by support reactions in ground displacement analysis).
TEP	Total work done by static nodal loads.
TESO	Total second order work (i.e., work done by P-Δ shears).
TEN	Energy error (= TES+TEI+TED-TEE).
TEE	Total external work (=TEP+TEW+TESO).
TEAD	Total alpha-M damping work.
TEKI	TEK at start of time step.
TEII	TEI at start of time step.
TEDI	TED at start of time step.
TESI	TES at start of time step.
TEWI	TEW at start of time step.
TEPI	TEP at start of time step.
TESOI	TESO at start of time step.
TENI	TEN at start of time step.
TEEI	TEEI at start of time step.
TEADI	TEAD at start of time step.
ENEG	Static work done in each element group.
ENRD	Beta-K damping work done in each element group.
EEXT	Second-order work done in each element group.
ENEGI	ENEG at start of time step.
ENRDI	ENRD at start of time step.
EEXTI	EEXT at start of time step.

2.2.14. EQNS Common

/EQNS/ stores variables used in the solution of equations. The stiffness matrix is stored in compacted column form [9].

/EQNS/ is declared as follows.

```
COMMON /EQNS/ neq, neqq, lenk, jcol, maxdof
```

The variables are as follows.

Variable	Description
NEQ	Total number of equations.
NEQQ	NEQ+1. All vectors are of length NEQQ (see /STOR/). DOF number NEQQ is assigned to all restrained displacements (see KID in /STOR/).
LENK	Length of compacted stiffness matrix.
JCOL	First column in stiffness matrix that has changed due to element events in the most recent state determination. All columns from this point must be refactorized, but columns up to this point do not change.
MAXDOF	Maximum number of DOFs for any element in the current structure. Must be less than 30 (see /DISVEL/).

2.2.15. EVENT Common

/EVENT/ stores variables used in the event-to-event solution scheme.

/EVENT/ is declared as follows.

```
COMMON /EVENT/ afac, facc, unbl(3), unbf, unbm,
1              irdof, irelm, irend, irevnt, irgrp, irnod,
2              iquit, maxev, neven
```

The variables are as follows.

Variable	Description
AFAC	Accumulated event factor for current load or time step (proportion of step "used up").
FACC	Event factor for current substep.
UNBL	Maximum equilibrium unbalance in each displacement direction. This array is UNBL(6) in DRAIN-3DX.
UNBF	Maximum force unbalance.
UNBM	Maximum moment unbalance.

IRDOF	Governing displacement direction if collapse displacement is exceeded.
IRELM	Element number with the smallest event factor.
IREND	Event type code for element with the smallest event factor. The meaning of this code depends on the element type. See element User Guides.
IREVNT	Code for event type. 0 : No event. 1 : Element event. 2 : Load factor increment reached for load or time step. 3 : Controlled displacement increment reached for load step. 4 : Load removed to satisfy displacement control. 5 : Load factor increment or time increment reached for analysis segment. 6 : Controlled displacement increment reached for segment. 7 : Maximum number of steps reached for segment. 8 : Maximum number of events reached for load or time step. 9 : Maximum number of successive direction changes (flip-flops) exceeded for the step. 10 : Collapse translation exceeded. 11 : Collapse rotation exceeded. 12 : Structure unstable (detected by negative or zero term on the diagonal during factorization of the stiffness matrix). 13 : Displacement control failed to prevent flip-flop.
IRGRP	Group number of element with smallest event factor.
IRNOD	Node number at which collapse displacement exceeded.
IQUIT	Termination code at end of current load or time step. 0 : Proceed to the next step as current analysis segment has not been completed ($0 \le$ IREVNT ≤ 4). 1 : Proceed to next segment as current segment has been successfully completed ($5 \le$ IREVNT ≤ 6). -1 : Quit analysis without completing current segment as analysis cannot proceed further ($7 \le$ IREVNT ≤ 13).
MAXEV	Maximum number of events allowed in a load or time step.
NEVEN	Number of events in current step.

2.2.16. GENINF Common

/GENINF/ stores data for each element group.

The maximum number of element groups is currently 20.

/GENINF/ is declared as follows.

```
COMMON /GENINF/ betao(20), ovfac(20,2), relpr(2,20),
1                ielpr(2,20), kelem(20), kevnt(20),
2                kgeom(20), nedof(20), nelem(20), nenod(20),
3                ninfe(20), ninfel(20), ninfg(20), ninfl(20),
4                ninft(20), nlinf(20), maxgr
```

The variables are as follows.

Variable	Description
BETAO	Stiffness proportional damping factor for each group.
OVFAC	Event overshoot scale factor for each group. 1 : for static analyses. 2 : for dynamic analyses.
RELPR	Real element parameters for each group.
IELPR	Integer element parameters for each group.
KELEM	Element type number for each group.
KEVNT	Event calculation code for each group. 0 : Suppress element event factor calculation. 1 : Calculate element event factors.
KGEOM	$P-\Delta$ analysis code for each group. 0 : Ignore $P-\Delta$ effects. 1 : Consider $P-\Delta$ effects and allow geometric stiffness to change for static analyses only. 2 : Consider $P-\Delta$ effects and allow geometric stiffness to change for both static and dynamic analyses.
NEDOF	Number of element DOFs for each group.

NELEM	Number of elements in each group.
NENOD	Number of nodes per element for each group.
NINFE	Length (in 4-byte units) of /INFEL/ block for each group.
NINFEL	Location of integrity violation variable in /INFEL/ for each group. Certain element data is stored at the end of /INFEL/ by the base program. If an element subroutine writes more than the specified /INFEL/ length, this data is destroyed. To warn against this during element development, this variable is checked after each call to an element subroutine. If the variable has been over-written, the program writes an error message.
NINFG	Length (in 4-byte units) of /INFGR/ block for each group.
NINFL	Length (in 4-byte units) of one element load set for each group.
NINFT	Number of output items per element for static or dynamic analyses for each group (length of /THELM/).
NLINF	Number of output items per element for response spectrum analysis for each group.
MAXGR	Maximum number of element groups allowed. Set in MAIN. Equal to 20 in current version.

2.2.17. INDIC Common

/INDIC/ stores a number of indicators that are used to control the overall solution process.

/INDIC/ is declared as follows.

```
 COMMON /INDIC/ kdata, kexe, kresis, kenr, kpdel, kauto, keven,
1                     kenrc, keqbc, maxevd, ktit, kstat
```

The variables are as follows.

Variable	Description
KDATA	Data error counter.

KEXE	Execution code. 0 : Execute. 1 : Data checking only. 2 : Execute if all element /INFEL/ and /INFGR/ blocks can be held in memory, otherwise data checking only.
KRESIS	Code for calculating resisting forces. 1 : Static only. 2 : Static and dynamic.
KENR	Energy calculation code. 0 : Omit calculations. 1 : Static only. 2 : Static and dynamic.
KPDEL	$P-\Delta$ analysis code. 0 : Ignore $P-\Delta$ effects. 1 : Consider $P-\Delta$ effects.
KAUTO	Code for dynamic analysis scheme. 1 : Constant time step. 2 : Variable time step.
KEVEN	Code for event calculation. 0 : Ignore events. 1 : Consider events.
KENRC	Code for velocity correction to satisfy energy balance. 0 : Omit correction. 1 : Perform correction.
KEQBC	Code for acceleration modification to improve equilibrium 0 : Omit modification. 1 : Perform modification.
MAXEVD	Maximum number of events allowed in a time step.
KTIT	Code for printing title in results printout (0:No ; 1:Yes). KTIT is set to 1 at start of first load or time step, and reset to 0 after title has been printed.
KSTAT	Code for type of analysis. 1 : Static. 2 : Dynamic.

2.2.18. INFEL Common

Data is stored for each element to monitor its nonlinear behavior. Data unique to each element is transferred to and from the element subroutines by means of this block. The length of the block and the data it contains depend on the element type.

/INFEL/ is declared in base program subroutines as follows.

```
COMMON /INFEL/ ielm(*)
```

/INFEL/ is declared fully in the element subroutines.

2.2.19. INFGR Common

/INFGR/ stores data common to all elements of an element group (as distinct from /INFEL/, which stores data specific to each element). Element group data is transferred to and from the element subroutines by means of this block. The length of the block and the data it contains depend on the element type.

/INFGR/ is declared in base program subroutines as follows.

```
COMMON /INFGR/ igrin(*)
```

/INFGR/ is declared fully in the element subroutines.

2.2.20. INTCOF Common

/INTCOF/ stores coefficients for the constant average acceleration integration scheme. The integration scheme could be changed by changing these variables, but this is not recommended.

/INTCOF/ is declared as follows.

```
COMMON /INTCOF/ cof1, cof2, cof3, cof4, cof5, cof6,
1                    cof2a, cof2b, cof5a, cof5b
```

The variables are as follows.

Variable	Description
COF1	$4/\Delta t^2$
COF2	$2/\Delta t$
COF3	$4/\Delta t$
COF4	$\Delta t/12$
COF5	2
COF6	$\Delta t^2/12$
COF2A	COF2 × ALPHA (see /DAMPG/ for ALPHA).
COF2B	COF2 × BETA (see /DAMPG/ for BETA).
COF5A	COF5 × ALPHA (see /DAMPG/ for ALPHA).
COF5B	COF5 × BETA (see /DAMPG/ for BETA).

2.2.21. LOADP Common

/LOADP/ defines the storage block size for ground acceleration, ground displacement and dynamic force records on permanent files with extensions .ACC, .DIS and .FRC, respectively (see /ASDFRC/ and /ADFREC/).

/LOADP/ also stores maximum length information for response spectra. The maximum allowed size of a response spectrum block is MPSPEC (=141) time-value pairs, and each spectrum must fit in one block.

/LOADP/ is declared as follows.

```
COMMON /LOADP/ mpairs, mpspec
```

The variables are as follows.

Variable	Description
MPAIRS	Number of time-value pairs per block in .ACC, .DIS and .FRC files.
MPSPEC	Number of time-value pairs per response spectrum in .SPC file.

2.2.22. NUMS Common

/NUMS/ stores commonly used constants. The values of these constants are set in BLOCK DATA (file BLOCK.FOR).

/NUMS/ is declared as follows.

```
 COMMON /NUMS/ vlarg, vsmal, spstif(2)
```

The variables are as follows.

Variable	Description
VLARG	Large number (1.0e+6).
VSMAL	Small number (1.0e-6).
SPSTIF	Support-spring stiffnesses for translational and rotational DOFs, respectively.

2.2.23. OUTD Common

/OUTD/ stores the result output intervals for dynamic analysis.

/OUTD/ is declared as follows.

```
 COMMON /OUTD/ tsaved, tppsvd, tpoutd, tenvsd, tenvpd,
1                  isaved, ippsvd, ipoutd, ienvsd, ienvpd
```

The variables are as follows.

Variable	Description
TSAVED	Time interval for saving structure state. 0.0 : Ignored. Do not save if ISAVED also = 0. >0.0 : Save at this interval, unless ISAVED governs, and at end of analysis segment
TPPSVD	Time interval for saving results for post-processing. 0.0 : Ignored. Do not save if IPPSVD also = 0. >0.0 : Save at this interval, unless IPPSVD governs, and at end of analysis segment.
TPOUTD	Time interval for results printout. 0.0 : Ignored. Do not print if IPOUTD also = 0. >0.0 : Print at this interval, unless IPOUTD governs, and at end of analysis segment.
TENVSD	Time interval for saving envelopes for post-processing. 0.0 : Ignored. Do not save if IENVSD also = 0. >0.0 : Save at this interval, unless IENVSD governs, and at end of analysis segment.
TENVPD	Time interval for envelope printout. 0.0 : Ignored. Do not print if IENVPD also = 0. >0.0 : Print at this interval, unless IENVPD governs, and at end of analysis segment.
ISAVED	Step interval for saving structure state. 0 : Ignored. Do not save if TSAVED also = 0.0. n : Save state every 'n' steps, unless TSAVED governs, and at end of analysis segment
IPPSVD	Step interval for saving results for post-processing. 0 : Ignored. Do not save if TPPSVD also = 0.0. n : Save every 'n' steps, unless TPPSVD governs, and at end of analysis segment.

IPOUTD	Step interval for results printout. 0 : Ignored. Do not print if TPOUTD also = 0.0. n : Print every 'n' steps, unless TPOUTD governs, and at end of analysis segment.
IENVSD	Step interval for saving envelopes for post-processing. 0 : Ignored. Do not save if TENVSD also = 0.0. n : Save every 'n' steps, unless TENVSD governs, and at end of analysis segment.
IENVPD	Step interval for envelope printout. 0 : Ignored. Do not print if TENVPD also = 0.0. n : Print every 'n' steps, unless TENVPD governs, and at end of analysis segment.

2.2.24. OUTS Common

/OUTS/ stores the result output intervals for static analysis.

/OUTS/ is declared as follows.

```
COMMON /OUTS/ isaves, ippsvs, ipouts, ienvss, ienvps
```

The variables are as follows.

Variable	Description
ISAVES	Step interval for saving structure state. 0 : Do not save. n : Save every 'n' steps and at end of analysis segment
IPPSVS	Step interval for saving results for post-processing. -1 : Save every event. 0 : Do not save. n : Save every 'n' steps and at end of analysis segment.
IPOUTS	Step interval for results printout. -1 : Print every event. 0 : Do not print. n : Print every 'n' steps and at end of analysis segment.

IENVSS	Step interval for saving envelopes for post-processing. 0 : Do not save. n : Save every 'n' steps and at end of analysis segment.
IENVPS	Step interval for envelope printout. 0 : Do not print. n : Print every 'n' steps and at end of analysis segment.

2.2.25. OUTP Common

/OUTP/ stores the step and time intervals since the last results and/or envelope output.

/OUTP/ is declared as follows.

```
 COMMON /OUTP/ ttsave, ttppsv, ttpout, ttenvs, ttenvp,
1                    iisave, iippsv, iipout, iienvs, iienvp,
2                    ksave, kppsv, kpout, kenvs, kenvp
```

The variables are as follows.

Variable	Description
TTSAVE	Time interval since last structure state save.
TTPPSV	Time interval since last results post-processing save.
TTPOUT	Time interval since last results printout.
TTENVS	Time interval since last envelope post-processing save.
TTENVP	Time interval since last envelope printout.
IISAVE	Step interval since last structure state save.
IIPPSV	Step interval since last results post-processing save.
IIPOUT	Step interval since last results printout.
IIENVS	Step interval since last envelope post-processing save.
IIENVP	Step interval since last envelope printout.
KSAVE	Code for saving structure state in the current step (0:No ; 1:Yes).
KPPSV	Code for saving results for post-processing in the current step (0:No ; 1:Yes).
KPOUT	Code for results printout in the current step (0:No ; 1:Yes).

KENVS	Code for saving envelopes for post-processing in the current step (0:No ; 1:Yes).
KENVP	Code for envelope printout in the current step (0:No ; 1:Yes).

2.2.26. PREC Common

/PREC/ stores a precision code for real variables. The default for real variables is double precision. If single precision is desired, change IPREC to 1 in BLOCK DATA, and change DOUBLE.H to set the default precision to single.

/PREC/ is declared as follows.

```
COMMON /PREC/ iprec
```

The variable is as follows.

Variable	Description
IPREC	1 : Single precision; 2 : Double precision.

2.2.27. PTOP Common

/PTOP/ is for page layout during printing.This feature has not yet been implemented.

/PTOP/ is declared as follows.

```
COMMON /PTOP/ npage, nlin, maxlin, madum
```

The variables are as follows.

Variable	Description
NPAGE	Current page number.
NLIN	Current line number.
MAXLIN	Number of writable lines per page.
MADUM	Number of skip lines per page. Used to skip the perforations for continuous paper.

2.2.28. RHIST Common

/RHIST/ stores counts of numbers of items for results printout and post-processing.

/RHIST/ is declared as follows.

```
 COMMON /RHIST/ lrec, ntime, ndpout, nelth, npsec, nnrds,
1                       nptime, ndpost, nelthp, npsecp, nnrdsp
```

The variables are as follows.

Variable	Description
LREC	Length of buffer REC (fwa = KREC in /STOR/) for output of time-history and/or envelope results.
NTIME	Number of printout sets in current analysis segment.
NDPOUT	Number of nodes and subnodes in a printout set.
NELTH	Number of elements in a printout set.
NPSEC	Number of structure sections in a printout set.
NNRDS	Number of generalized displacements in a printout set.
NPTIME	Number of post-processing sets in current analysis segment.
NDPOST	Number of nodes and subnodes in a post-processing set.
NELTHP	Number of elements in a post-processing set.
NPSECP	Number of structure sections in a post-processing set.
NNRDSP	Number of generalized displacements in a post-processing set.

2.2.29. SECTON Common

/SECTON/ stores structure section information.

/SECTON/ is declared as follows.

```
 COMMON /SECTON/ nssec, maxcut, ntrns, maxtdf
```

The variables are as follows.

Variable	Description
NSSEC	Number of structure sections.
MAXCUT	Largest number of elements cut by any section.
NTRNS	Number of force transformations.
MAXTDF	Largest number of DOFs for any transformation.

2.2.30. SETREL Common

/SETREL/ stores generalized displacement information.

/SETREL/ is declared as follows.

```
 COMMON /SETREL/ nrds
```

The variable is as follows.

Variable	Description
NRDS	Number of generalized displacements.

2.2.31. STAT Common

/STAT/ stores control information for static analysis.

/STAT/ is declared as follows.

```
 COMMON /STAT/ disa, disma, fdis, flod, slfac, tload,
1              iflip, ipsign, isign, jdof, ldof, kdc, klc, maxfp
```

The variables are as follows.

Variable	Description
DISA	Controlled displacement increment per step.
DISMA	Controlled displacement increment for analysis segment.

FDIS	Current value of controlled displacement increment for analysis segment.
FLOD	Current value of load factor for analysis segment.
SLFAC	Load factor increment per load step.
TLOAD	Load factor increment for analysis segment.
IFLIP	Current number of successive load direction changes (i.e., flip-flops).
IPSIGN	ISIGN in preceding substep. Used to detect flip-flops.
ISIGN	Sign of load factor increment for satisfying controlled displacement condition.
JDOF	Equation number for the 1st node displacement defining the controlled displacement.
LDOF	Equation number for the 2nd node displacement defining the controlled displacement.
KDC	Displacement control code (0:No ; 1:Yes).
KLC	Load control code (0:No ; 1:Yes).
MAXFP	Maximum number of successive flip-flops allowed.

2.2.32. STOR Common

/STOR/ stores the first word addresses of the arrays in blank common, plus some variables used frequently with the storage scheme (see Section 2.3. for details). The variables defining the array dimensions are in /CONTR/, /EQNS/, /RHIST/, /SECTON/, /SETREL/ and /STOR/.

/STOR/ is declared as follows.

```
 COMMON /STOR/ ntst, knsb, kndfsb, kcosb, kndid, kcoord, kid,
1              kidsp, kspdsp, klstif, kfmnod, kfmdof, kalpha,
2              knecut, ksang, kdist, kidcut, kstrns, knodir,
3              krdfac, kjnod, kjelm, kjsec, kjrds, kwkspc, krds,
4              ksefor, krec, kenp, kenn, kistp, kistn, ksecen,
5              kisece, krdsen, kirdse, kxlod, kexts, kext,
6              kdext, krints, krint, kru, kdisi, kveli, kacci,
7              kdds, kcvel, kcacc, kdis, kvel, kacc, kenri,
8              kdru, kdtan, kbetak, kdinfb, kiad, kinfb, keffk,
9              ktank, nblok, niad, ninfb, ksofar
```

The variables are as follows.

Variable	**Description**
NTST	Length of blank common in 4-byte units.
NBLOK	Number of disk blocks used to store /INFGR/ and /INFEL/ for all elements.
NIAD	Length of IAD array, = NELTOT + NELG (see /CONTR/).
NINFB	Length of INFB array = size of each disk block used to store /INFGR/ and /INFEL/.
KSOFAR	Next unallocated address in blank common.

The arrays stored in blank common are as follows.

FWA	**Array**	**Description**
KNSB	NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type, NT, is equal to NSB(nt+1) - NSB(nt).

KNDFSB	NDFSB(3,nsnds) in 2DX NDFSB(6,nsnds) in 3DX	DOF codes for all subnodes of each compound node type, as follows. 0 : Absolute displacement. 1 : Restrained (not a DOF). 2 : Relative displacement w.r.t. main node. 3 : Special degree of freedom (i.e., not a conventional translation or rotation).
KCOSB	COSB(2,nsnds) in 2DX COSB(3,nsnds) in 3DX	Coordinate offsets from main node for subnodes of each compound node type.
KNDID	NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type number (0 = not a compound node). The compound node types are numbered in the order of input. 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.
KCOORD	COORD(2,nnods) in 2DX COORD(3,nnods) in 3DX	Nodal coordinates, in ascending node number order.
KID	ID(3,ntnds) in 2DX ID(6,ntnds) in 3DX	Equation numbers for displacements at each node and subnode, coded as follows. NEQQ : Restrained displacement. (See /EQNS/ for NEQQ). +n : Displacement is unrestrained and unslaved. 'n' = equation number. -n : Displacement is slaved. 'n' = sequence number of master node.
KIDSP	IDSP(ndsp)	Equation number for each spring supported displacement.

KSPDSP	SPDSP(ndsp,2)	Imposed displacements at spring supports (1: total ; 2: increment for current step).
KLSTIF	LSTIF(neqq)	Location of diagonal elements in compacted stiffness matrix.
KFMNOD	FMNOD(3,nnods) in 2DX FMNOD(6,nnods) in 3DX	Nodal masses.
KFMDOF	FMDOF(neqq)	Mass matrix (diagonal).
KALPHA	ALPHAM(neqq)	Alpha-M damping matrix (diagonal).
KNECUT	NECUT(nssec)	Number of elements cut by each structure section.
KSANG	SANG(2,nssec) in 2DX SANG(3,3,nssec) in 3DX	Cosine and sine of section inclination angles in 2DX. Direction cosines of section axes in 3DX.
KDIST	DIST(2,maxcut,nssec) in 2DX DIST(3,maxcut,nssec) in 3DX	Coordinate offsets from structure section centers to element cuts (X and Y offsets in 2DX; X, Y and Z offsets in 3DX).
KIDCUT	IDCUT(3,maxcut,nssec)	Identification array for cut elements in structure sections, as follows. 1 : Group number. 2 : Element number. 3 : Force transformation number.
KSTRNS	STRNS(3,maxtdf,ntrns) in 2DX STRNS(6,maxtdf,ntrns) in 3DX	Force transformation matrices for structure sections.

KNODIR	NODIR(8,nrds)	Displacements forming each generalized displacement, coded as follows. +n : Equation number. -n : Slaved displacement. 'n' = NDISP + NLOC×4 for 2DX. 'n' = NDISP + NLOC×7 for 3DX. Where, NLOC = sequence number of the slaved node = n/4 for 2DX and n/7 for 3DX. NDISP = direction of DOF = MOD(n,4) for 2DX and MOD(n,7) for 3DX.
KRDFAC	RDFAC(8,nrds)	Participation factors for displacements forming each generalized displacement.
KJNOD	JNOD(nnods)	Output codes for nodal displacements, as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KJELM	JELM(neltot)	Output codes for element results, as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KJSEC	JSEC(nssec)	Output codes for structure section forces, as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.

KJRDS	JRDS(nrds)	Output codes for generalized displacements, as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KWKSPC	WKSPC(nwksp)	Work space for use by base program subroutines.
KRDS	RDS(nrds)	Current generalized displacement magnitudes.
KSEFOR	SEFOR(6,nssec) in 2DX SEFOR(12,nssec) in 3DX	Current structure section forces (3 or 6 static values followed by 3 or 6 damping values for each section).
KREC	REC(lrec)	Buffer for output of time-history and/or envelope results for post-processing and/or printout.
KENP	DENP(3,ntnds) in 2DX DENP(6,ntnds) in 3DX	Positive nodal displacement envelopes.
KENN	DENN(3,ntnds) in 2DX DENN(6,ntnds) in 3DX	Negative nodal displacement envelopes.
KISTP	ISTP(3,ntnds) in 2DX ISTP(6,ntnds) in 3DX	Step numbers for DENP.
KISTN	ISTN(3,ntnds) in 2DX ISTN(6,ntnds) in 3DX	Step numbers for DENN.
KSECEN	SECENV(3,6,nssec) in 2DX SECENV(6,6,nssec) in 3DX	Section force envelopes. 2nd index indicates: 1 : Total positive. 2 : Total negative. 3 : Static positive. 4 : Static negative. 5 : Damping positive. 6 : Damping negative.
KISECE	ISECEN(3,6,nssec) in 2DX ISECEN(6,6,nssec) in 3DX	Step numbers for SECENV.
KRDSEN	RDSENV(2,nrds)	Positive and negative generalized displacement envelopes.

KIRDSE	IRDSEN(2,nrds)	Step numbers for RDSENV.
KXLOD	XLOD(neqq)	Effective nodal load increment for the current load or time step.
KEXTS	EXTS(neqq)	Total static load.
KEXT	EXT(neqq)	Total static + dynamic load.
KDEXT	DEXT(neqq)	Dynamic load increment for the current time step. For *ACCN or *ACCR analysis DEXT is the load increment due to ground acceleration increments. For *DISN or *DISR analysis DEXT is the load increment required to impose the specified ground displacement increments.
KRINTS	RINTS(neqq)	Static resisting force.
KRINT	RINT(neqq)	Total (static + damping + inertia) resisting force.
KRU	RU(neqq)	Unbalanced load (EXT - RINT).
KDISI	DISI(neqq)	Displacements at start of current time step.
KVELI	VELI(neqq)	Velocities at start of current time step.
KACCI	ACCI(neqq)	Accelerations at start of time step.
KDDS	DDIS(neqq)	Displacement increment in current substep.
KCVEL	CVEL(neqq)	Velocity increment in current substep.
KCACC	CACC(neqq)	Acceleration increment in current substep.
KDIS	DIS(neqq)	Total displacements.
KVEL	VEL(neqq)	Total velocities.
KACC	ACC(neqq)	Total accelerations.
KENRI	ENRI(neqq)	Work done by inertia forces in current time step.
KDRU	DRU(neqq)	Backed up RU (if KDRU $\neq$ 1). Used if time step is repeated. See INCOR in /CONTR/.

KDTAN	DTAN(lenk)	Backed up TANK (if KDTAN ≠ 1). Used if time step is repeated. See INCOR in /CONTR/.
KBETAK	BETAK(lenk)	Beta-K damping matrix, compacted column (if KBETAK ≠ 1). If KBETA in /DAMPG/ = 0, then this matrix is not required and KBETAK=1.
KDINFB	IINFB(ninfb)	Backed up INFB (if KDINFB ≠ 1). Used if time step is repeated. See INCOR in /CONTR/.
KIAD	IAD(niad)	First word addresses in INFB of /INFGR/ and /INFEL/ blocks. If IAD=1 for any /INFGR/ or /INFEL/ block, that block does not lie in current INFB, and a new INFB block must be read from the file storing the element data blocks.
KINFB	INFB(ninfb)	Buffer for /INFGR/ and /INFEL/ blocks.
KEFFK	EFFK(lenk)	Current effective tangent stiffness matrix (compacted column, factorized).
KTANK	TANK(lenk)	Current static tangent stiffness matrix (compacted column, unfactorized).

2.2.33. TAPES Common

/TAPES/ stores the unit numbers for disk files. The unit numbers are assigned in BLOCK DATA (file BLOCK.FOR).

/TAPES/ is declared as follows.

```
 COMMON /TAPES/ inp, iou, inpx, nfbeg, nfcur, nfupd, nfscrt, nfres,
1               nfperm, nfload, nflis, nfoutp, nflog, nfgeo, nfprnt,
2               nfmode, nfmrsl, nfasdf, nfenvp
```

The variables are as follows.

Unit No.	Associated File(s)
INP	Input file, DRAIN.INP.
IOU	Echo file, .ECH.
INPX	Input scratch file. Used for counting input items before data is actually read.
NFBEG	Scratch file storing /INFGR/ and /INFEL/ blocks at start of current step. Each disk block may contain one or more /INFGR/ and /INFEL/ blocks. The size of each disk block is NINFB (in /STOR/).
NFCUR	Scratch file storing /INFGR/ and /INFEL/ blocks at start of current substep.
NFUPD	Scratch file storing updated /INFGR/ and /INFEL/ blocks at end of current substep.
NFSCRT	Scratch file storing individual /INFGR/ and /INFEL/ blocks during data input.
NFRES	Results post-processing file, .RXX, where XX is analysis segment number.
NFPERM	Structure state file, .SXX, where XX is analysis segment number.
NFLOAD	Files containeng load patterns and dynamic load records (i.e., .ELD, .STA, .SPC, .ACC, .VEL, .DIS, .FRC).
NFLIS	Analysis list file, .LST.
NFOUTP	Printout file, .OUT.
NFLOG	Solution log file, .SLO.
NFGEO	Geometry file, .GEO.
NFPRNT	Scratch file for saving results for printout. Results are saved step-by-step, then reorganized and written to .OUT file item-by-item.
NFMODE	Mode shape file, .MXX, where XX is analysis segment number.
NFMRSL	Modal response file, .UXX, where XX is analysis segment number.
NFASDF	Scratch file for dynamic load records. Used during dynamic analysis.
NFENVP	Envelope file, .EXX, where XX is analysis segment number, for envelope post-processing.

2.2.34. THELM Common

Element results are transferred to results files (.RXX) by means of /THELM/. The results are stored in single precision. The data and length of /THELM/ depends on the element type.

/THELM/ is declared in base program subroutines as follows.

```
COMMON /THELM/ thout(*)
REAL thout
```

/THELM/ is declared fully in the element subroutines.

2.2.35. TIME Common

/TIME/ stores time data for dynamic analysis.

/TIME/ is declared as follows.

```
COMMON /TIME/ dt, timax, tim, kstep, nsteps
```

The variables are as follows.

Variable	Description
DT	Current time step, Δt.
TIMAX	Time increment for current analysis segment.
TIM	Total time in the current dynamic load record.
KSTEP	Step number for the current analysis segment.
NSTEPS	Maximum number of steps for current segment (analysis quits if exceeded).

2.2.36. TITADF Common

/TITADF/ stores the record names for the dynamic load records that are applied in the current analysis.

/TITADF/ is declared as follows.

```
COMMON /TITADF/ tadf(36)
CHARACTER tadf*4
```

The variable is as follows.

Variable	Description
TADF	Record names for dynamic load records. See /ASDFRC/.

2.2.37. TITLE Common

/TITLE/ stores the problem titles.

/TITLE/ is declared as follows.

```
COMMON /TITLE/ fname, ihed, anal, iheda
CHARACTER fname*8, ihed*40, anal*4, iheda*40
```

The variables are as follows.

Variable	Description
FNAME	Problem name. The number of characters in FNAME is NCHAR (in /CONTR/). All permanent files opened for the problem have names of the form PROBNAME.EXT, where PROBNAME is FNAME and EXT is a three character extension indicating the contents of the file.
IHED	Problem title.
ANAL	Analysis type ('GRAV', 'STAT', 'REST', 'MODE', 'SPEC', 'ACCN', 'ACCR', 'VELN', 'VELR', 'DISN', 'DISR', 'FORN' or 'FORR').
IHEDA	Analysis title.

2.2.38. WORK Common

/WORK/ provides a temporary work area for exclusive use by element subroutines.

/WORK/ is declared in main program (file MAIN.FOR) as follows.

```
COMMON /WORK/ w(*)
```

/WORK/ is declared fully in any element subroutines where it is used.

2.3. BLANK COMMON

Blank common is declared as follows in all except the main program.

```
COMMON L(*)
```

The length is defined by the parameter NTSTP in the main program (file MAIN.FOR) as follows.

```
 PARAMETER (NTSTP=65000)
COMMON L(NTSTP)


ntst = NTSTP
```

The 65000 value is about the maximum for PCs under DOS with a 640K memory limit. For workstations and PCs with larger memory limits or virtual memory systems it is probably most efficient to specify sufficient length to store all data in memory.

Arrays in L are located by their first-word-addresses (FWA). The arrays and their addresses have been listed in /STOR/. If an array is not allocated space, because it is not required for the current problem or because it is not kept in memory, its FWA is set to 1. Therefore L(1) is not used and first allocated address is L(2).

Most real variables are currently in double precision (REAL*8). It is possible that some of these variables will be made single precision in future versions.

The variable KSOFAR stores the first unallocated address in L at any stage of execution of the program. To allocate, say, an integer array NDID of dimensions (3, NTNDS), the code is as follows.

```
kndid = ksofar
ksofar = kndid + 3 × ntnds
```

Each real array is made to start on a 8-byte boundary by assigning an odd FWA. For example, to allocate real array COORD of dimensions (2, NNODS), the code is as follows.

```
kcoord = NXTODD (ksofar)
ksofar = kcoord + 2 × nnods × iprec
```

In the above, NXTODD is the simple statement function:

```
NXTODD(k) = k + MOD(k+1,2)
```

After each allocation in L, KSOFAR is checked to ensure that it is less than NTST. If KSOFAR > NTST, the blank common length is insufficient to run the problem, and the program writes an error message in the .ECH file. The parameter NTSTP in the main program (file MAIN.FOR) must be increased to run the problem.

The utility subroutines, IZERO, RZERO and DZERO are used to zero integer, real and double precision array, respectively. The utility subroutines ISHIFT and DSHIFT are used to copy integer and real arrays, respectively. The utility subroutine RDSHFT is used to copy a double precision array to a single precision array. The utility subroutines IREAD, RREAD and DREAD are used to read integer, single and double precision arrays, respectively, from an unformatted sequential file. The utility subroutines IWRITE, RWRITE and DWRITE are used to write integer, single and double precision arrays, respectively, to an unformatted sequential file.

2.4. PERMANENT FILES

The base program creates several permanent files. All permanent files have names of the form PROBNAME.EXT, where PROBNAME is the problem name (up to 8 characters), FNAME in /TITLE/, and EXT is a three character extension indicating the contents of the file.

Static load patterns and dynamic load records are stored on binary files as follows. Unit NFLOAD is used for all these files, as only one of them needs to be open at any time.

BINARY FILES FOR LOAD PATTERNS AND RECORDS		
Extension	**Unit No.**	**Contents**
.ELD	NFLOAD	Static element load patterns.
.STA	NFLOAD	Static nodal load patterns.
.VEL	NFLOAD	Initial velocity patterns.
.ACC	NFLOAD	Ground acceleration records.
.DIS	NFLOAD	Ground displacement records.
.FRC	NFLOAD	Dynamic force records.
.SPC	NFLOAD	Earthquake response spectra.

The following output files are all text files.

OUTPUT TEXT FILES		
Extension	**Unit No.**	**Contents**
.ECH	IOU	Echo of input data and analysis log consisting of event log and unbalanced load information..
.LST	NFLIS	List of analysis segments, indicating which time-history post-processing, envelope post-processing, and structure state files have been saved.
.OUT	NFOUTP	Time history and envelope results.
.SLO	NFLOG	Solution log consisting of energy balance and unbalanced load information.

The output files for post-processing are all binary files, as follows.

BINARY OUTPUT FILES FOR POST-PROCESSING		
Extension	**Unit No.**	**Contents**
.GEO	NFGEO	Geometry data for the structure for plotting.
.EXX	NFENVP	Envelope results for analysis segment XX.
.RXX	NFRES	Time-history results for analysis segment XX.

The following files are all binary.

OTHER PERMANENT FILES		
Extension	**Unit No.**	**Contents**
.MXX	NFMODE	Mode shapes and periods for analysis segment XX.
.UXX	NFMRSL	Individual modal results for unit spectral amplitude for modes in the .MXX file. This file is created the first time a response spectrum analysis is performed using these modes. The unit modal results are not recalculated if later analyses are performed with different response spectra.
.SXX	NFPERM	Structure state at end of analysis segment XX, for restart in subsequent analysis sessions.

Detailed descriptions of the structures of all binary files, except .SXX, are given in Chapter 4. See subroutines SSTATE and RSTATE for the structure of the .SXX file.

2.5. TEMPORARY FILES

2.5.1. Introduction

Temporary files (i.e., FORTAN scratch or internal files) are used for the following.

(a) Input processing.

(b) Output processing.

(c) Storage of element /INFGR/ and /INFEL/ blocks.

(d) Storage of backup /INFGR/ and /INFEL/ blocks; tangent stiffness, DTAN; and unbalanced load vector, DRU. These are used to restore the state if the time step must be repeated in the variable time-step scheme.

These files are described in the following sections.

2.5.2. Input Processing

In the DRAIN.INP file, separator lines signal the start of each new task. If item counts are needed before memory can be assigned for any input task, the input data is read twice. In the first reading, the items are counted and the input lines are written to the scratch file INPX. The program then allocates memory and re-reads the input lines from INPX.

Each input line is read into the character variable XXLINE (see /CLINE/). XXLINE is then used as a FORTRAN internal file. The data in XXLINE can be read several times if desired.

2.5.3. Output Processing

The results for all items (see /RHIST/) are obtained step-by-step during the analysis and are written to the scratch file, NFPRNT. At end of analysis segment, results for all steps are read from NFPRNT, reorganized item-by-item instead of step-by-step, and written to the .OUT file.

2.5.4. Storage of Element Data Blocks

Data is stored for each element in /INFGR/ and /INFEL/ blocks. There is one /INFGR/ block for each element group (containing data common to all elements in the group) plus one /INFEL/ block for each element (containing data unique to each element).

The base program stores these blocks in array INFB (see KINFB in /STOR/). For any element task (e.g., event factor calculation, state determination, etc.), before an element subroutine is called, the /INFGR/ and /INFEL/ data for the current element is copied from

INFB to the /INFGR/ and /INFEL/ blocks. If the updates made by the element subroutine to its /INFEL/ block are to be accepted (e.g., following state determination), then the /INFEL/ data is copied back to INFB. If the updates made by the element subroutine to its /INFEL/ block are to be ignored (e.g., following event factor calculation), then the /INFEL/ data is *not* copied back.

For a small problem, or if the blank common is large, INFB can accomodate all /INFGR/ and /INFEL/ blocks. For a large problem, however, INFB may accomodate only a few /INFGR/ and /INFEL/ blocks. For such problems, the element data is blocked. Each block has length NINFB (in 4-byte units), and the number of blocks is NBLOK (see /STOR/). The value of NINFB depends on the remaining blank common after all arrays except DTAN, DRU and DINFB have been allocated.

If the element data can not be held in memory, the base program uses three temporary files, with unit numbers NFBEG, NFCUR, and NFUPD (see /TAPES/). NFBEG holds the element data at the beginning of the current load or time step, NFCUR holds the data at the start of the current analysis substep, and NFUPD holds the updated data at the end of the current substep. The unit numbers are cycled as follows.

(a) Before the first substep, NFBEG and NFCUR are switched (i.e., the unit numbers are interchanged) so that NFCUR holds the current data.

(b) At the start of any substep, NFCUR holds the current data. As the elements are processed the base program fills INFB by reading the next block from NFCUR whenever it is required to process the current element. If the data is to be updated (e.g., following state determination), then INFB is written to NFUPD before it is refilled from NFCUR. After such an element task, NFCUR and NFUPD are switched so that NFCUR holds the current data for the next element task or substep.

(c) In the first substep after the end of the first data update, NFBEG and NFUPD are switched so that NFBEG again holds the data corresponding to the start of the step.

(d) After the end of the last substep, NFBEG and NFCUR are switched so that NFBEG holds the current data at the start of the next step.

2.5.5. Storage of Backup Element Data, Tangent Stiffness and Unbalanced Load

A duplicate set of element data, DINFB; a duplicate tangent stiffness, DTAN; and, a duplicate unbalanced load vector, DRU, are required to provide a back-up in case a time-step must be repeated in the variable time-step scheme. DINFB, DTAN and DRU are stored in blank common if possible. If not, they are stored on unit NFBEG.

If all /INFGR/ and /INFEL/ blocks can be accomodated in INFB in a single block (NBLOK=1), the available blank common is checked to see whether it can accomodate DTAN, DRU, and DINFB. The variable INCOR (in /CONTR/) is set to mean the following.

(a) INCOR=0 : NBLOK > 1; so that DINFB is already on NFBEG (see Section 2.5.4). DTAN and DRU are stored following DINFB.

(b) INCOR=1 : DINFB, DTAN and DRU are stored on NFBEG.

(c) INCOR=2 : DTAN and DRU are stored in blank common, and DINFB is stored on NFBEG.

(d) INCOR=3 : DINFB, DTAN and DRU are stored in blank common.

Because repetition of a time step is likely to occur relatively rarely, in most time steps the time required to write the backup copy to file is wasted. To save some of this time, the backup is made not at the beginning of the step but at the end of the first substep, just before the state is updated. Hence, if there are no events (and hence no substeps), a backup copy is not made.

3. BASE PROGRAM ORGANIZATION -- DRAIN-2DX AND 3DX

3.1. INTRODUCTION

The base program phases and the corresponding separator lines in the DRAIN.INP file are as follows.

No.	Base Program Phase	Separator Lines	Comment
1.	Begin Session	*START/ *RESTART	For *RESTART, phases (2), (5), and (6) are skipped
2.	Define Model	*COMPOUND, *NODECOORDS, *NODETYPES, *RESTRAINTS, *SLAVING, *MASSES, *ELEMENTGROUP, *SECTION and *GENDISP	
3.	Process Output Specification	*RESULTS	
4.	Process Load Patterns and Load Records		
	(a) Static Element Load Patterns	*ELEMLOAD	
	(b) Static Nodal Load Patterns	*NODALOAD	
	(c) Initial Velocity Paterns	*NODALVEL	
	(d) Ground Acceleration Records	*ACCNREC	
	(e) Ground Displacement Records	*DISPREC	
	(f) Dynamic Force Records	*FORCREC	
	(g) Response Spectrum	*SPECTRUM	
5.	Allocate Memory for Analysis Phase		Preparation for analysis phase.

6.	Form Initial Tangent Stiffness and Beta-K Damping		Preparation for analysis phase.
7.	Process Analysis Parameters	*PARAMETERS	
8.	Identify Analysis Type	*GRAV, *STAT, *REST, *MODE, *SPEC, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN, or *FORR	
9.	Set up Loads for Next Analysis Segment:		
	(a) Static Gravity	*GRAV	
	(b) Static	*STAT	
	(c) Restore to Static State	*REST	
	(d) Ground Acceleration	*ACCN or *ACCR	
	(e) Initial Velocity	*VELN or *VELR	
	(f) Dynamic Ground Displacement	*DISN or *DISR	
	(g) Dynamic Force	*FORN or *FORR	
10.	Perform Analysis		
	(a) Static Gravity	*GRAV	
	(b) Static	*STAT	
	(c) Restore to Static State	*REST	
	(d) Ground Acceleration	*ACCN or *ACCR	
	(e) Initial Velocity	*VELN or *VELR	
	(f) Dynamic Ground Displacement	*DISN or *DISR	
	(g) Dynamic Force	*FORN or *FORR	
	(h) Mode Shapes and Periods	*MODE	
	(i) Response Spectrum	*SPEC	
11.	End Session	*STOP	

The phases are described briefly in the following sections. Details can be obtained from the actual code, which is well commented.

3.2. BEGIN SESSION

The flow chart for this phase is as follows.

```
MAIN ─┐
      └─ CONTRL ─┐
                 ├─ INITL   for *START/*RESTART data
                 └─ RSTATE
```

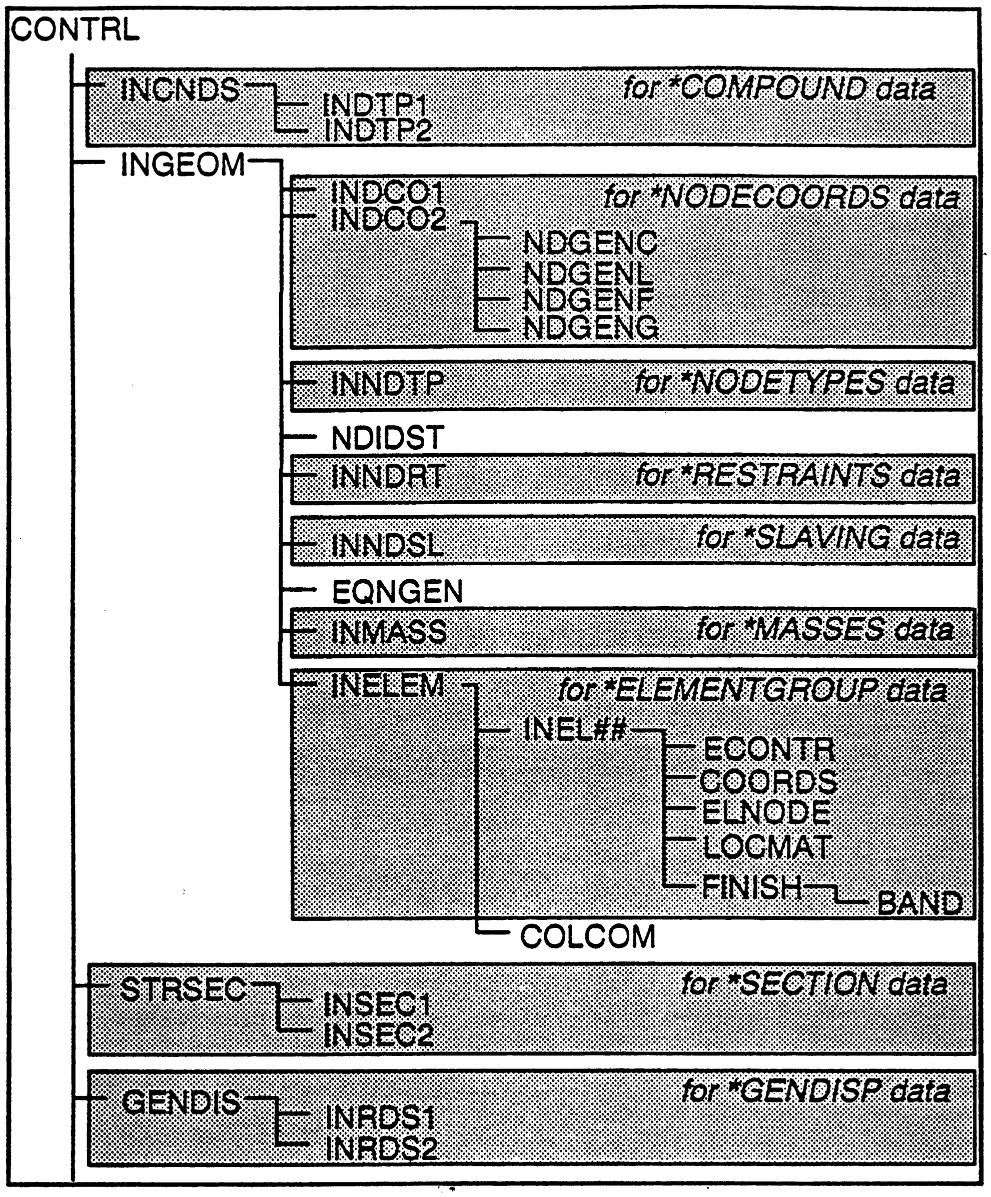
The following main tasks are performed.

1. In MAIN, open the input data file, DRAIN.INP (unit INP in /TAPES/).
2. In INITL, read the *START/*RESTART data. Set the variables LECHO in /CLINE/; NCHAR in /CONTR/; KDATA, KEXE, KENR and KPDEL in /INDIC/; and FNAME in /TITLE/.
3. In INITL, open the files .ECH (unit IOU) and .OUT (unit NFOUTP).
4. For *RESTART
   a) In CONTRL, open the .SXX file (unit NFPERM), where XX is the restart state number.
   b) In RSTATE, restore the state from the .SXX file (blank common, all labeled common blocks and INFB blocks).

3.3. DEFINE MODEL

3.3.1. Main Tasks

The flow chart for this phase is as follows.



The following main tasks are performed.

1. *COMPOUND data:

   a) In INDTP1, read the *COMPOUND data; count NDTP and NSNDS (see /CONTR/); and write input lines to INPX.

   b) In INCNDS, allocate arrays NSB, NDFSB and COSB (see KNSB, KNDFSB and KCOSB in /STOR/) in blank common.

c) In INDTP2, re-read the *COMPOUND data from INPX and set up NSB, NDFSB and COSB.

2. *NODECOORDS data:

   a) In INDCO1, read the *NODECOORDS data; count NNODS (see /CONTR/); and write input lines to INPX.

   b) In INGEOM, allocate arrays NDID and COORD (see KNDID and KCOORD in /STOR/) in blank common.

   c) In INDCO2, re-read the *NODECOORDS data from INPX and set up node numbers in first row of NDID and corresponding nodal coordinates in COORD. INDCO2 calls NDGENC for control nodes, NDGENL for straight line generation, NDGENF for frontal extrapolation, and NDGENG for grid interpolation.

   d) In INGEOM, sort COORD and first row of NDID in increasing order of node numbers.

3. In INNDTP, read the *NODETYPES data and set up second row of NDID.
4. In NDIDST, set up third row of NDID and count NTNDS (see /CONTR/).
5. In INGEOM, allocate arrays ID and IDSP (see KID and KIDSP in /STOR/) in blank common. The actual length NDSP of IDSP is unknown at this stage, and is provisionally set to 3×NNODS for 2DX and 6×NNODS for 3DX.
6. In INNDRT, read the *RESTRAINTS data, and code ID as follows.

   a) 0 : for a free (i.e., unrestrained) displacement,

   b) 1 : for a fixed (i.e., restrained) displacement, and

   c) 2 : for a spring supported displacement.

7. In INNDSL, read the *SLAVING data, and additionally code ID as follows.

   d) -MNOD for a slaved nodal displacement, where MNOD is sequence number of the master node in NDID.

8. In EQNGEN, set up ID and IDSP; set the variables NDSP (in /CONTR/); and NEQ and NEQQ (in /EQNS/).
9. In INGEOM, reduce the allocated space for IDSP to NDSP, and allocate arrays SPDSP, LSTIF, FMNOD, FMDOF and ALPHAM (see KSPDSP, KLSTIF, KFMNOD, KFMDOF and KALPHA in /STOR/) in blank common.
10. In INMASS, read the *MASSES data and set up FMNOD, FMDOF and ALPHAM.
11. Temporarily allocate an array, IELNOD, in blank common. IELNOD stores the element nodes for elements of an element group. This array is set up in ELNODE and written to the .GEO file in INELEM.
12. In INELEM
    a) Initialize LSTIF so that LSTIF(J) = J for each column J of the stiffness matrix (see Section 3.3.2b).
    b) Write geometry data to the .GEO file (See Chapter 4).
    c) Read the Group Information line of the *ELEMENTGROUP data; set the arrays BETAO, OVFAC, KELEM, KEVNT, KGEOM and NELEM in /GENINF/; and set the variables NELGR, NELG and NELTOT in /CONTR/.
    d) Call the element subroutine INEL##, where ## is the element type number, to read rest of the *ELEMENTGROUP data.
13. In INEL##
    a) Call the base program subroutine ECONTR.
    b) Read the Element Data lines of the *ELEMENTGROUP data.
    c) Call the base program subroutine COORDS to get the coordinates of any node.
    d) Set up the /INFGR/ block for the group.
    e) For each element in the group
        i) Call the base program subroutine ELNODE for each element node.
        ii) Call the base program subroutine LOCMAT for each element DOF.

iii) Set up the /INFEL/ block.

iv) Call the base program subroutine FINISH.

14. In ECONTR, set the variables NEDOF, NENOD, NINFE, NINFEL, NINFG, NINFL, NINFT and NLINF in /GENINF/ for the element group.
15. In ELNODE, set the element nodes in IELNOD.
16. In LOCMAT, set the equation numbers in the element location matrix for the element DOFs (See Section 3.3.2a).
17. In FINISH:
    a) For the first element store the /INFGR/ block on scratch file NFSCRT.
    b) For each element store the /INFEL/ block on NFSCRT.
18. In BAND, update LSTIF (see Section 3.3.2b).
19. In COLCOM, set up LSTIF (see Section 3.3.2b).
20. For *SECTION data:
    a) In INSEC1, read the *SECTION data; count NSSEC and NTRNS; update MAXCUT and MAXTDF (see /SECTON/); and write the input lines to INPX.
    b) In STRSEC, allocate arrays NECUT, SANG, DIST, IDCUT and STRNS (see KNECUT, KSANG, KDIST, KIDCUT and KSTRNS in /STOR/) in blank common.
    c) In INSEC2, re-read the *SECTION data from INPX and set up NECUT, SANG, DIST, IDCUT and STRNS.
21. For *GENDISP data:
    a) In INRDS1, read the *GENDISP data; count NRDS (see /SETREL/); and write the input lines to INPX.
    b) In GENDIS, allocate arrays NODIR and RDFAC (see KNODIR and KRDFAC in /STOR/) in blank common.

c) In INRDS2, re-read the *GENDISP data from INPX and set up NODIR and RDFAC.

3.3.2. Some Details

a) Element Location Matrix (LM array)

The LM array is used by the base program to assemble element stiffnesses, resisting forces, etc. For any element DOF the corresponding term has the following meaning.

a) + n : Unrestrained DOF. 'n' = global equation number.

b) -n : Slaved DOF.

'n'=NDISP+NLOC×4 for DRAIN-2DX

'n'=NDISP+NLOC×7 for DRAIN-3DX

where,

NLOC=sequence number of the slaved node = n/4 for 2DX and n/7 for 3DX.

and NDISP=direction of DOF= MOD(n,4) for 2DX and MOD(n,7) for 3DX.

The LM array for each element is stored following the /INFEL/ data in the /INFEL/ block. Between the /INFEL/ data and the LM array, a 4-byte integrity code is stored . After each return from an element subroutine the base program checks the integrity code to ensure that the integrity of the LM array has not been violated in the element subroutine (see NINFEL in /GENINF/).

This feature of the program has caused problems and may be changed in future versions.

b) Setting up LSTIF

To set up LSTIF (see KLSTIF in /STOR/):

1. For each column, J, of the stiffness matrix, TANK (see KTANK in /STOR/), LSTIF(J) is set equal to the row number of the first nonzero entry in that column. For this purpose:

a) LSTIF(J) is initialized to J (i.e., the diagonal entry) in INELEM (main task 12a).

b) BAND updates LSTIF by using the element LM array, assuming that all element DOFs are coupled to each other (main task 18).

2. After the data for all elements has been input, COLCOM finally sets up LSTIF and calculates the compacted column length, LENK, of the stiffness matrix (main task 19).

c) Element Processing

The variables in /CURRNT/ keep track of the current stage of element processing.

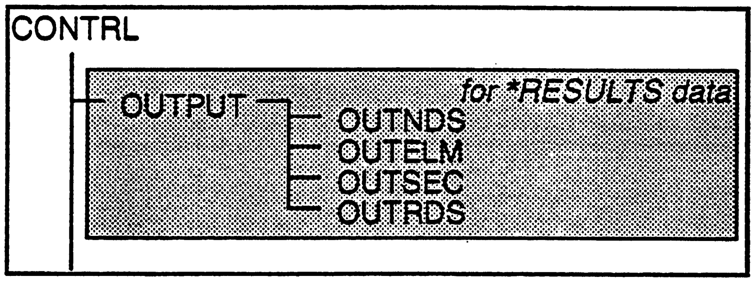
INELEM initializes IGRC to 0 before reading the first *ELEMENTGROUP separator, and increments IGRC for each *ELEMENTGROUP separator.

INELEM initializes IELC to 1 after each *ELEMENTGROUP separator. FINISH increments IELC for each element.

INELEM initializes IDFC and INODC to 0 after each *ELEMENTGROUP separator. LOCMAT increments IDFC for each element DOF. ELNODE increments INODC for each element node. IDFC and INODC are reset to 0 in FINISH for the next element.

3.4. PROCESS OUTPUT SPECIFICATION

The flow chart for this phase is as follows.



The following main tasks are performed.

1. For *START, in OUPUT

   a) Set the variable LREC in /RHIST/.

b) Allocate arrays JNOD, JELM, JSEC and JRDS (see KJNOD, KJELM, KJSEC and KJRDS in /STOR/) in blank common.

c) Initialize JNOD, JELM, JSEC and JRDS with the default output codes.

d) Initialize the variables NDPOUT, NELTH, NPSEC, NNRDS, NDPOST, NELTHP, NPSECP and NNRDSP in /RHIST/ corresponding to the default output codes.

2. In OUTPUT, read the *RESULTS data and call:

   a) OUTNDS to update JNOD, NDPOUT and NDPOST for nodal displacements.

   b) OUTELM to update JELM, NELTH and NELTHP for element results.

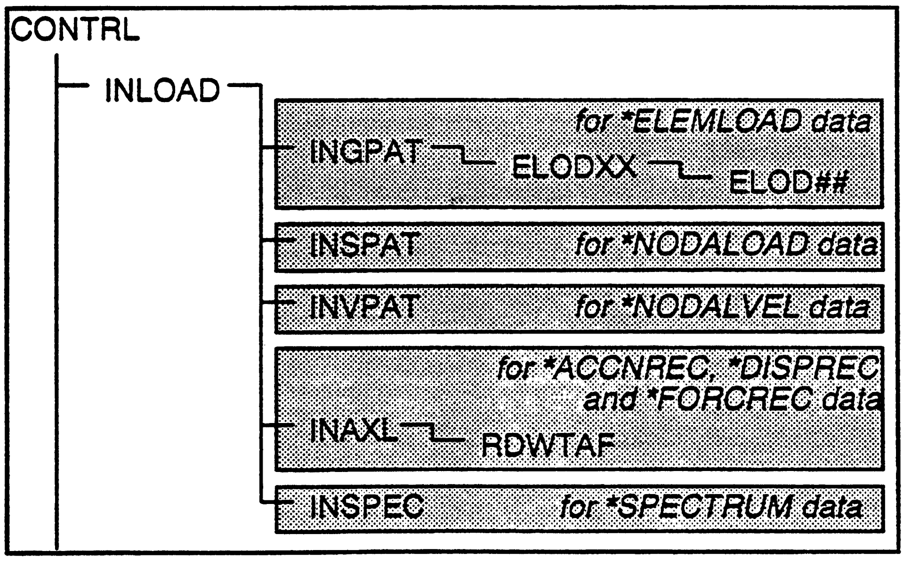
   c) OUTSEC to update JSEC, NPSEC and NPSECP for structure sections.

   d) OUTRDS to update JRDS, NNRDS and NNRDSP for generalized displacements.

3.5. PROCESS LOAD PATTERNS AND LOAD RECORDS

3.5.1. Main Tasks

The flow chart for this phase is as follows.



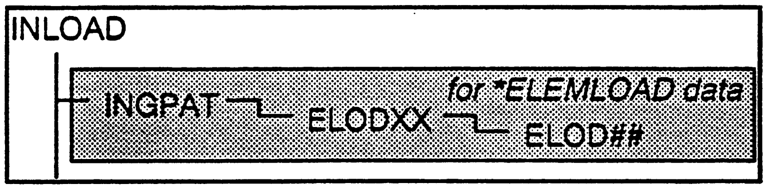
The following main tasks are performed.

1. In INLOAD, allocate work space for reading load patterns and dynamic load records. For *RESTART, use the array WKSPC in blank common (see KWKSPC in /STOR/). For *START, use all remaining blank common.
2. In INLOAD, read the separator line and for:
   a) *ELEMLOAD - call INGPAT to read the *ELEMLOAD data and write the element load pattern to the .ELD file (see .ELD file in Chapter 4).
   b) *NODALOAD - call INSPAT to read the *NODALOAD data and write the static nodal load pattern to the .STA file (see .STA file in Chapter 4).
   c) *NODALVEL - call INVPAT to read the *NODALVEL data and write the initial velocity pattern to the .VEL file (see .VEL file in Chapter 4).
   d) *ACCNREC, *DISPREC or *FORCREC - call INAXL to read the *ACCNREC, *DISPREC or *FORCREC data and write the dynamic load record to the .ACC, .DIS or .FRC file (see these files in Chapter 4).
   e) *SPECTRUM - call INSPEC to read the *SPECTRUM data and write the response spectrum to the .SPC file (see .SPC file in Chapter 4).

3.5.2. Some Details

a) Processing Element Loads - Subroutine INGPAT

The flow chart for processing element loads is as follows.



The element loads for a loaded element group are stored in SETLOD and ELFACT arrays, as follows.

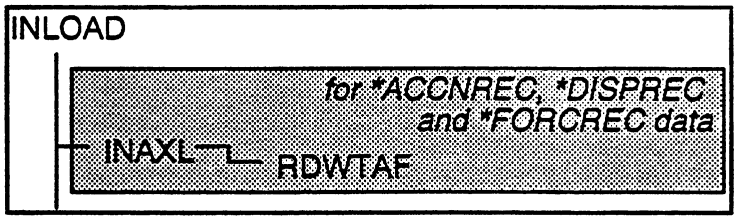
Array	Description
SETLOD(ninl,nlod)	Element load sets. Each column of SETLOD contains a load set. NINL is the length of a load set, from array NINFL in /GENINF/. NLOD is the number of element load sets.
ELFACT(nlod,nmem)	Load set scale factors for elements of the group. Each column of ELFACT contains the load set scale factors for one element. NMEM is the number of elements, from arrray NELEM in /GENINF/.

The following main tasks are performed.

1. In INGPAT, read pattern name and loaded element groups.
2. In ELODXX, call element subroutine ELOD## where ## is the element type number.
3. In element subroutine ELOD##, read the element load data for the loaded element group, and set up the SETLOD and ELFACT arrays.
4. In INGPAT, append the element load pattern at the end of the .ELD file.

b) Processing Load Records - Subroutine INAXL

The flow chart for processing dynamic load records is as follows.



The load records are stored in blocks (see /ASDFRC/, /ADFREC/ and /LOADP/) each containing MPAIRS (in /LOADP/) time-value pairs.

The following main tasks are performed.

1. In INAXL, read the pattern name and control information for the load record; and calculate the number of blocks that are required.

2. In RDWTAF, read the load record; form the time-value blocks; and write the blocks to the load record file (.ACC, .DIS or .FRC).

3.6. ALLOCATE MEMORY FOR ANALYSIS PHASE

The flow-chart for this phase is as follows.

```
MAIN
├─ CONTRL ─┬─ MEMREQ
│          └─ CONSOL
```

The following main tasks are performed.

1. In CONTRL, allocate array WKSPC in blank common (see KWKSPC in /STOR/). WKSPC is used for storing the following.
   a) Load pattern and dynamic load record input for *RESTART.
   b) Applied load patterns when setting up loads for analysis.
   c) Element stiffness matrix during tangent stiffness update.
   d) Temporary vector of length NEQQ for calculation of midstep equilibrium errror or equivalent impulse error.
   e) Nodal displacements DISP(3,ntnds) in 2DX and DISP(6,ntnds) in 3DX for output during analysis, and for updating nodal displacement envelopes (see KENP, KENN in /STOR/).
   f) Section forces SEFOR (see KSEFOR in /STOR/) for output during analysis and for updating section envelopes (see KSECEN in /STOR/).
   g) Generalized displacements RDS (see KRDS in /STOR/) for output during analysis and for updating generalized displacement envelopes (see KRDSEN in /STOR/).
2. In CONTRL, set KSEFOR and KRDS (see /STOR/) so that SEFOR and RDS may safely use parts of the WKSPC array.

3. In CONTRL, allocate the following arrays in blank common (see KREC to KENRI in /STOR/).
   a) output buffer REC.
   b) arrays DENP, DENN, ISTP and ISTN for nodal displacement envelopes.
   c) arrays SECENV and ISECEN for structure section envelopes.
   d) arrays RDSENV and IRDSEN for generalized displacement envelopes.
   e) vectors XLOD, EXTS, EXT, DEXT, RINTS, RINT, RU, DISI, VELI, ACCI, DDIS, CVEL, CACC, DIS, VEL, ACC and ENRI.
4. In CONTRL, calculate blank common available for storing /INFGR/ and /INFEL/ data.
5. In MEMREQ, set variables INCOR in /CONTR/; and NBLOK and NINFB in /STOR/.
6. In CONTRL, allocate the following arrays in blank common (see KDRU to KTANK in /STOR/):
   a) DRU if INCOR $\geq$ 2.
   b) DTAN if INCOR $\geq$ 2.
   c) BETAK if KBETA in /DAMPG/ $\neq$ 0.
   d) IINFB if INCOR = 3.
   e) IAD, INFB, EFFK and TANK.
7. In CONTRL, open the following scratch files:
   a) NFBEG if INCOR $\neq$ 3. NFBEG stores DINFB if INCOR = 2; and DINFB, DTAN and DRU if INCOR $\leq$ 2 (see Sections 2.5.4 and 2.5.5).
   b) NFCUR and NFUPD if NBLOK $\neq$ 1 for storing /INFGR/ and /INFEL/ blocks (see Section 2.5.4).
8. In CONSOL, fill the INFB buffer by reading the /INFGR/ and /INFEL/ blocks from scratch file NFSCRT and set up the IAD array.

If INFB cannot accomodate all /INFGR/ and /INFEL/ blocks (i.e., NBLOK > 1), then CONSOL performs the following steps until all /INFGR/ and /INFEL/ blocks are read from NFSCRT and all INFB blocks are written to NFBEG.

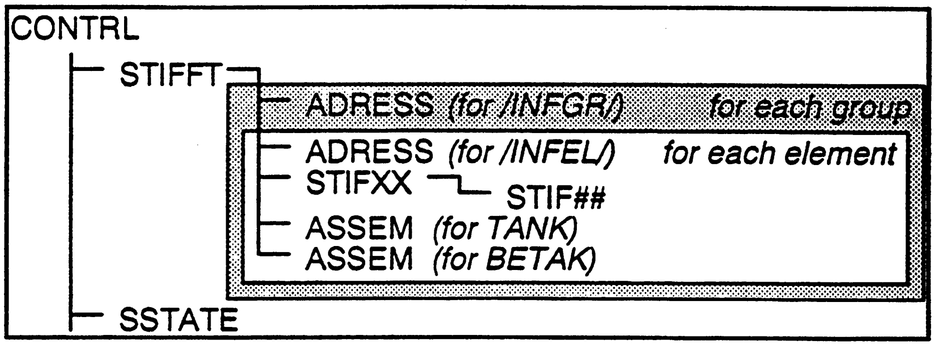
a) Fill INFB with as many consecutive /INFGR/ or /INFEL/ blocks as will fit in INFB, setting IAD for each block.

b) Write INFB to NFBEG.

c) Set IAD for the next /INFGR/ or /INFEL/ block to 1.

3.7. FORM INITIAL TANGENT STIFFNESS AND BETA-K DAMPING

The flow chart for this phase is as follows.



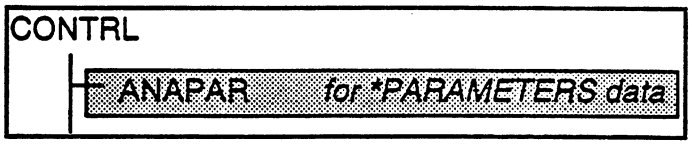
The following main tasks are performed.

1. In CONTRL, call STIFFT after detecting the first analysis separator (*GRAV, *STAT, *REST, *ACCN, *DISN, *FORN, *MODE or *SPEC).
2. In ADRESS, copy /INFGR/ and/or /INFEL/ for the current element group and element from INFB. If the required block is not in INFB, write INFB to NFUPD and refill INFB from NFCUR.
3. In STIFXX call the element subroutine STIF##, where ## is the element type number for the current group (see KELEM in /GENINF/).
4. In STIF##, form the element stiffness matrix.
5. In ASSEM, assemble the element stiffness into TANK (see KTANK in /STOR/).

6. In STIFFT, scale the element stiffness by BETAO (in /GENINF/) for the current element group to obtain the element beta-K damping matrix.
7. In ASSEM, assemble the element beta-K damping into BETAK (see KBETAK in /STOR/).
8. In CONTRL, open the .SXX file (unit NFPERM), where XX is segment number 0.
9. In SSTATE, write blank common, all labeled commons and INFB to the .SXX file. This saves the unstressed structure state.

3.8. PROCESS ANALYSIS PARAMETERS

The flow chart for this phase is as follows.



ANAPAR reads the *PARAMETERS data, and modifies variables and arrays in labeled common blocks, as follows.

Type of Analysis Parameter	**Modified Variables and Arrays**
Viscous Damping Scale Factors	ALPHA and BETA in /DAMPG/
Collapse Displacements	DISMAX and RTNMAX in /COLPSE/
Event Overshoot Scale Factors	OVFAC in /GENINF/
Element Parameters	IELPR and RELPR in /GENINF/
Output Intervals for Static Analysis	all variables in /OUTS/
Output Intervals for Dynamic Analysis	all variables in /OUTD/
Control Parameters for Dynamic Analysis	KEVEN, KENRC, KEQBC and MAXEVD in /INDIC/
Time Step Parameters for Dynamic Analysis	DTAUTO, DTCONS, DTMAX and DTMIN in /AUTO/

Parameters for Variable Time Step Scheme	DTRED, DTINC, TOLHII, TOLHIS, TOLLOI, TOLLOS, TOLMX and NSINC in /AUTO/

3.9. IDENTIFY ANALYSIS TYPE

The flow chart for this phase is as follows.

```
MAIN
 ├─ CONTRL
```

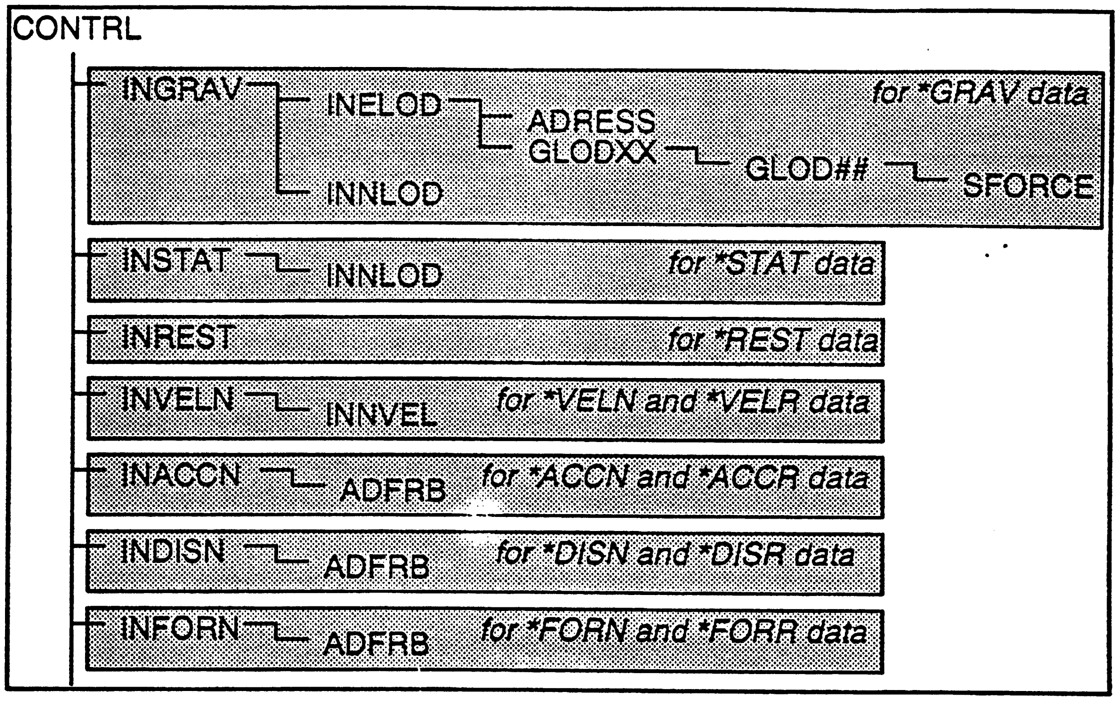
The following main tasks are performed in CONTRL.

1. Check that the separator is valid (i.e., one of *GRAV, *STAT, *REST, *MODE, *SPEC, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN, *FORR).
2. For *GRAV, check that the structure is in the unstressed state or the preceding analysis was also *GRAV.
3. For *STAT, *ACCN, *VELN, *DISN and *FORN, check that the structure is in a static state (i.e., preceding analysis was *GRAV, *STAT or *REST).
4. For *ACCR check that the preceding analysis was *ACCN or *ACCR.
5. For *VELR check that the preceding analysis was *VELN or *VELR.
6. For *DISR check that the preceding analysis was *DISN or *DISR.
7. For *FORR check that the preceding analysis was *FORN or *FORR.

3.10. SET UP LOADS FOR ANALYSIS SEGMENTS

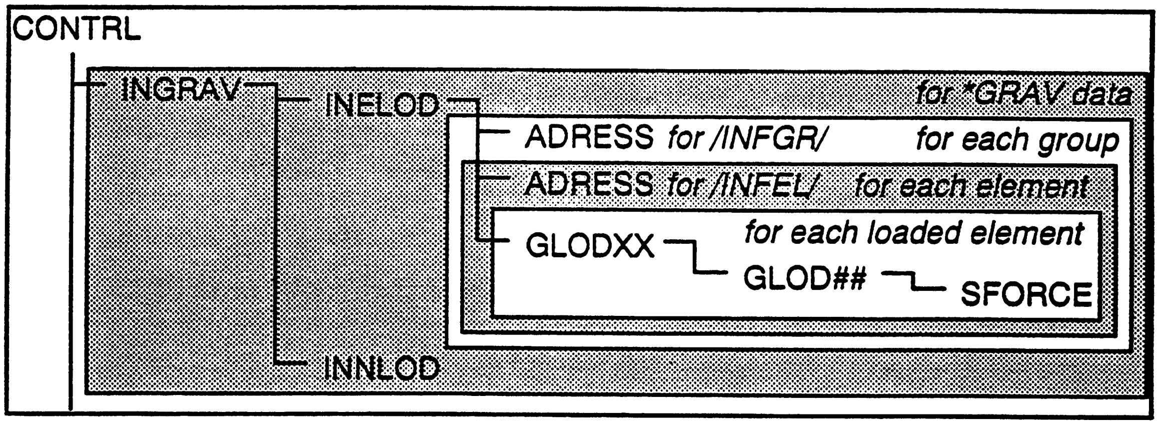
3.10.1. Main Tasks

The flow chart for this phase is as follows.



3.10.2. Loads for *GRAV Analysis - Subroutine INGRAV

The flow chart for setting up *GRAV loads is as follows.



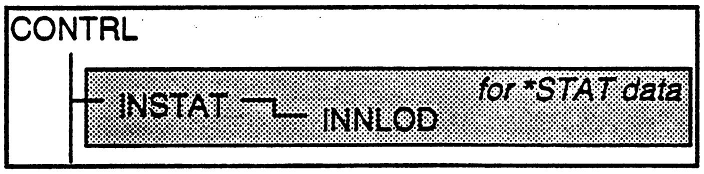
The following main tasks are performed.

1. In INGRAV, read the *GRAV data; initialize XLOD (see KXLOD in /STOR/) to zero; call INELOD to update XLOD for applied static element load patterns; call INNLOD to update XLOD for applied static nodal load patterns; and update XLOD for inertial loads.

2. In INELOD, read applied static element load patterns from .ELD file.
3. In INELOD, for each applied element load pattern and each loaded element group read SETLOD and ELFACT arrays from .ELD file.
4. In ADRESS, copy /INFGR/ and/or /INFEL/ block for current element from INFB. If required block is not in INFB, write INFB to NFUPD and refill INFB from NFCUR.
5. In INELOD, for each loaded element extract factors for the loaded element from ELFACT and call GLODXX.
6. In GLODXX, call element subroutine GLOD## where ## is the element type number for the loaded element group (see KELEM in /GENINF/).
7. In GLOD##, set up the element end clamping forces in array ELFINT(ndof); and call the base program subroutine SFORCE. NDOF is the number of element DOFs (see NEDOF in /GENINF/) for the loaded element.
8. In SFORCE, use the element LM array to assemble the contribution of ELFINT to XLOD.
9. In INNLOD, read applied static nodal load patterns from .STA file and assemble their contribution in XLOD.

3.10.3. Loads for *STAT Analysis - Subroutine INSTAT

The flow chart for setting up *STAT loads is as follows.



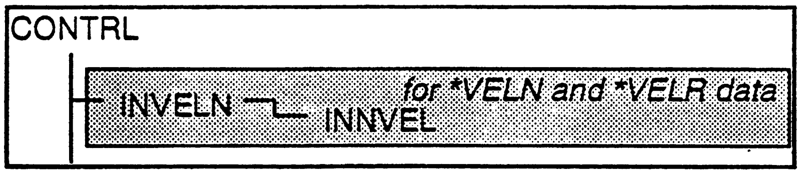
The following main tasks are performed.

1. In INSTAT, read the *STAT data and set the variables DISA, DISMA, SLFAC, TLOAD, JDOF, LDOF, KDC, KLC and MAXFP in /STAT/; NSTEPS in /TIME/; and MAXEV in /EVENT/.

2. In INSTAT, initialize XLOD (see KXLOD in /STOR/); call INNLOD to update XLOD due to applied static nodal load patterns; and update XLOD due to inertial loads.
3. In INNLOD, read applied static nodal load patterns from .STA file and assemble their contribution in XLOD.

3.10.4. Loads for *VELN or *VELR Analysis - Subroutine INVELN

The flow chart for setting up *VELN or *VELR loads is as follows.

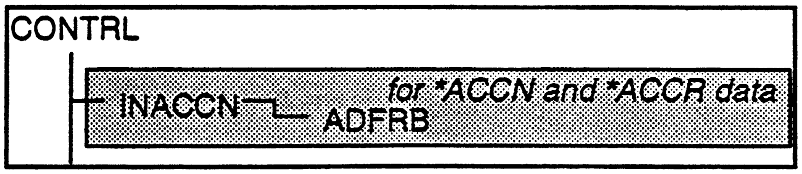


The following main tasks are performed.

1. In INVELN, read the *VELN or *VELR data and set the variables DT, TIMAX, TIM and NSTEPS in /TIME/; and KAUTO in /INDIC/.
2. For *VELN in INVELN:
   a) Call INNVEL to set up VEL (see KVEL in /STOR/) due to applied initial velocity patterns.
   b) Scale VEL to match the specified initial kinetic energy.
3. In INNVEL, read applied initial velocity patterns from .VEL file and assemble their contribution in VEL.

3.10.5. Loads for *ACCN or *ACCR Analysis - Subroutine INACCN

The flow chart for setting up *ACCN or *ACCR loads is as follows.

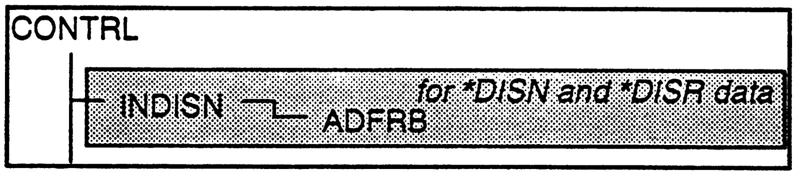


The following main tasks are performed.

1. In INACCN, read the *ACCN or *ACCR data and set the variables DT, TIMAX, TIM and NSTEPS in /TIME/; and KAUTO in /INDIC/.
2. For *ACCN in INACCN, set the variables TFAC, RECFAC, COROT, NDFREC and NPAIRS in /ADFREC/; and TADF in /TITADF/.
3. In ADFRB, read time-acceleration blocks (see /ASDFRC/, /ADFREC, and /LOADP/) for the applied records from .ACC file, and write blocks containing data for times between TIM and TIM+TIMAX on the scratch file, NFASDF.
4. In INACCN, set the variables INBL and LNBL in /ASDFRC/; and NPADF in /ADFREC/.
5. In INACCN, initialize the array NBL (in /ASDFRC/) to zero.

3.10.6. Loads for *DISN or *DISR Analysis - Subroutine INDISN

The flow chart for setting up *DISN or *DISR loads is as follows.



The following main tasks are performed.

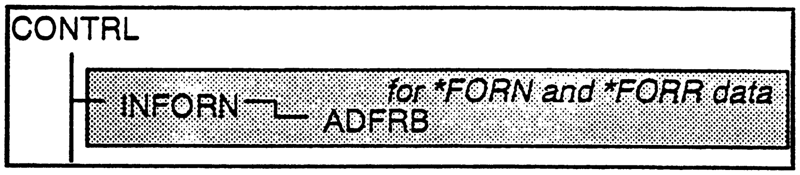
1. In INDISN, read the *DISN or *DISR data and set the variables DT, TIMAX, TIM and NSTEPS in /TIME/; and KAUTO in /INDIC/.
2. For *DISN in INDISN, set the variables TFAC, RECFAC, DTIME, NDFREC and NPAIRS in /ADFREC/; and TADF in /TITADF/.
3. In ADFRB, read time-displacement blocks (see /ASDFRC/, /ADFREC, and /LOADP/) for the applied records from .DIS file, and write blocks containing data for times

between TIM and TIM+TIMAX on the scratch file, NFASDF. If number of applied records exceeds 6, the block size on NFASDF is different from that on .DIS file.

4. In INDISN, set the variables INBL and LNBL in /ASDFRC/; and DFFAC, JDFREC, NPADF and NDSFR in /ADFREC/.
5. In INDISN, initialize the array NBL (in /ASDFRC/) to zero.

3.10.7. Loads for *FORN or *FORR Analysis - Subroutine INFORN

The flow chart for setting up *FORN or *FORR loads is as follows.



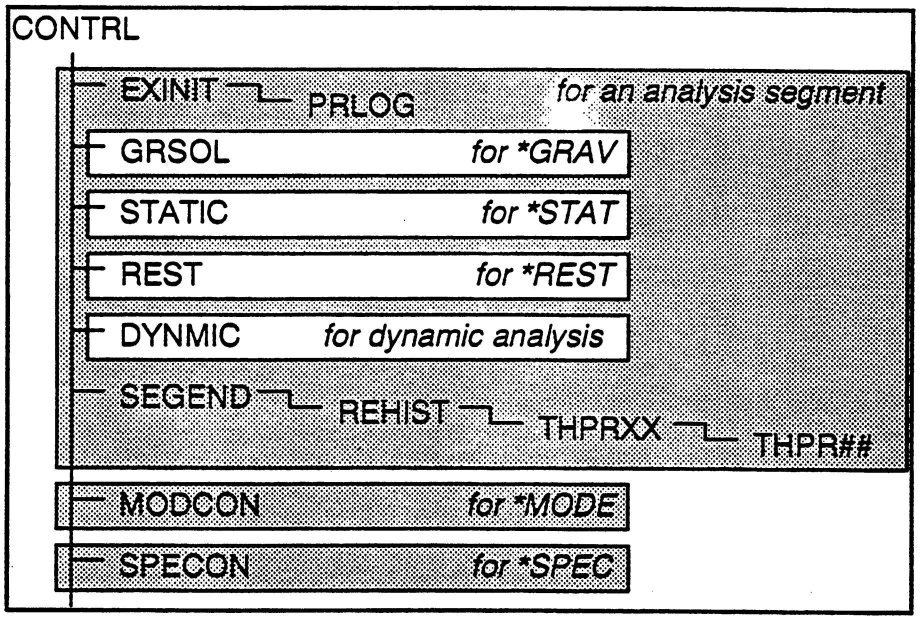
The following main tasks are performed.

1. In INFORN, read the *FORN or *FORR data and set the variables DT, TIMAX, TIM and NSTEPS in /TIME/; and KAUTO in /INDIC/.
2. For *FORN in INFORN, set the variables TFAC, RECFAC, DTIME, NDFREC and NPAIRS in /ADFREC/; and TADF in /TITADF/.
3. In ADFRB, read time-force blocks (see /ASDFRC/, /ADFREC, and /LOADP/) for the applied records from .FRC file, and write blocks containing data for times between TIM and TIM+TIMAX on the scratch file, NFASDF. If number of applied records exceeds 6, the block size on NFASDF is different from that on .FRC file.
4. In INFORN, set the variables INBL and LNBL in /ASDFRC/; and DFFAC, JDFREC, NPADF and NDSFR in /ADFREC/.
5. In INFORN, initialize the array NBL (in /ASDFRC/) to zero.

3.11. PERFORM ANALYSIS

3.11.1. Main Tasks

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In CONTRL, to perform a *GRAV, *STAT, *REST, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN or *FORR analysis:
   a) Call EXINIT.
   b) Call GRSOL for *GRAV.
   c) Call STATIC for *STAT.
   d) Call REST for *REST.
   e) Call DYNMIC for *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN or *FORR..
   f) Call SEGEND.
2. In CONTRL, to perform a *MODE analysis:

   a) Save blank common from L(KWKSPC) to L(KEFFK-1) on scratch file NFRES. This makes space available for storing mode shapes, a flexibility matrix, and other data. EFFK, TANK and some arrays before WKSPC in blank common are used in the analysis.
   b) Call MODCON to perform the analysis.
   c) On completion of the analysis, restore blank common from NFRES.

3. In CONTRL, to perform *SPEC analysis:
   a) Save blank common from L(KWKSPC) to L(KIAD-1) on scratch file NFRES. This makes space available for storing response results, and other data. IAD, INFB and some arrays before WKSPC in blank common are used in the analysis.
   b) Call SPECON to perform the analysis.
   c) On completion of the analysis, restore blank common from NFRES.

4. In EXINIT:
   a) Initialize variables KENR, KRESIS and KTIT in /INDIC/; all variables in /OUTP/; NTIME and NPTIME in /RHIST/; and IQUIT and NEVEN in /EVENT/.
   b) Increment analysis segment number, NSEG (in /CONTR/).
   c) Write heading for the segment on the .OUT file (unit NFOUTP in /TAPES/).
   d) Call PRLOG to write heading and starting energy log for the segment on the .SLO file (unit NFLOG). The heading and starting energy log for *REST is not written here, but later in REST.
   e) Update the .LST file.
   f) Open and write initial data to .RXX file (unit NFRES), where XX is the analysis segment number (see Chapter 4).
   g) Open and write initial data to .EXX file (unit NFENVP), where XX is the analysis segment number (see Chapter 4).
   h) Open scratch file, unit NFPRNT, to save time-history results for printout.

i) Write analysis title on the .ECH file.

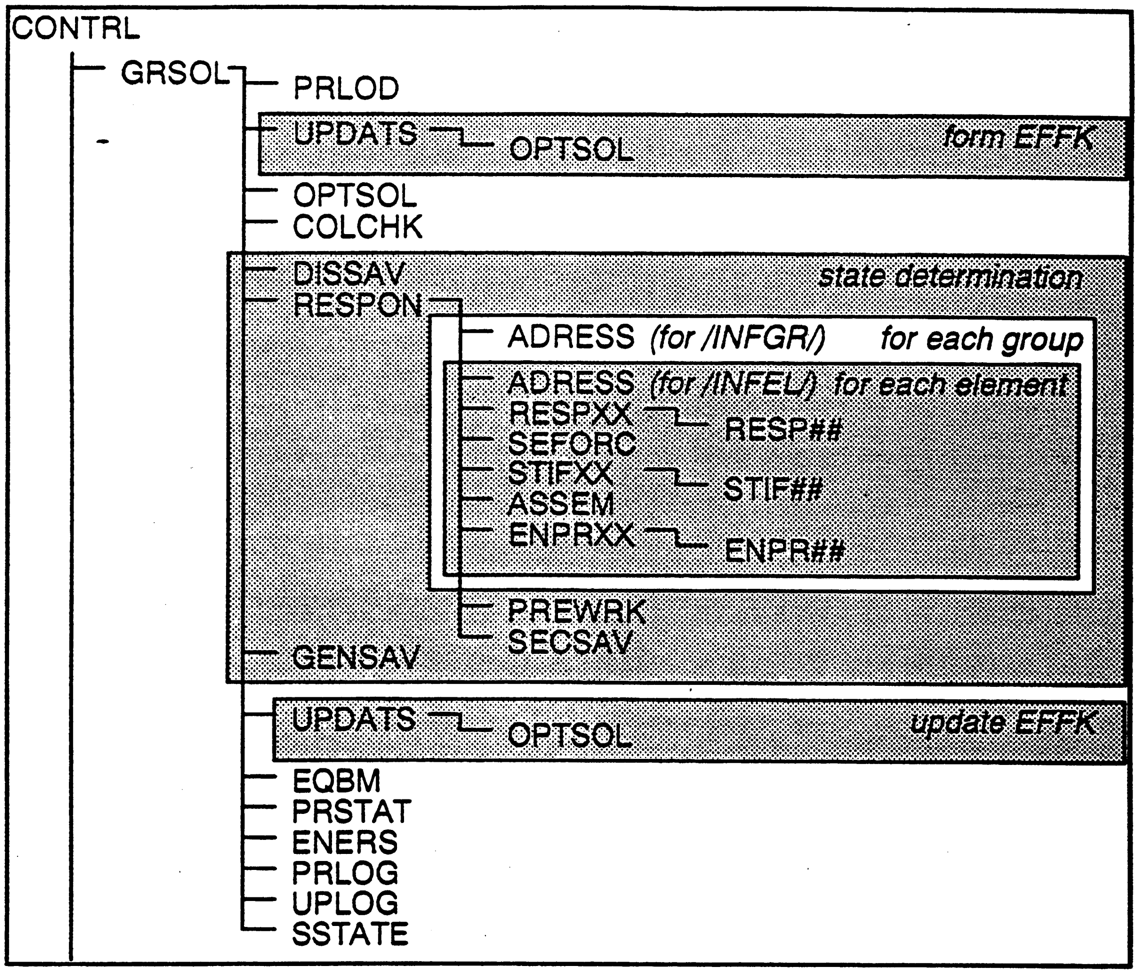
5. In GRSOL, STATIC and REST perform *GRAV, *STAT and *REST analysis, respectively, and write solution log for each substep to .ECH file (unit IOU); write energy log for each substep to .SLO file (unit NFLOG); write load-history results for printout to unit NFPRNT; write load-history results for post-processing to .RXX file (unit NFRES); write envelope results for printout to .OUT file (unit NFOUTP); write static and P-Δ work done for each element group (see ENEG and EEXT in /ENRGY/) to .OUT file (unit NFOUTP); write envelope results for post-processing to .EXX file (unit NFENVP); and write structure state data to .SXX file (unit NFPERM).
6. In DYNMIC perform the dynamic analysis, and write solution log for each substep to .ECH file (unit IOU); write energy log for each time step to .SLO file (unit NFLOG); write time-history results for printout to unit NFPRNT; write time-history results for post-processing to .RXX file (unit NFRES); write envelope results for printout to .OUT file (unit NFOUTP); write static, damping and P-Δ work done for each element group (see ENEG, ENRD and EEXT in /ENRGY/) to .OUT file (unit NFOUTP); write envelope results for post-processing to .EXX file (unit NFENVP); and write structure state data to .SXX file (unit NFPERM).
7. In SEGEND, close .RXX and .EXX files.
8. In REHIST, read results for printout from unit NFPRNT, and write the results for each output item (see /RHIST/) to .OUT file (unit NFOUTP). For element results output, REHIST copies element results to the /THELM/ block and calls THPRXX.
9. In THPRXX, call the element subroutine THPR##, where ## is the element type number for the element (see KELEM in /GENINF/).
10. In THPR##, write heading and element results from /THELM/ to the .OUT file.

11. In MODCON, read the *MODE data; perform *MODE analysis; and write periods, mass participation factors and mode shapes to .OUT file (unit NFOUTP) and .MXX file (unit NFMODE), where XX is the analysis segment number.
12. In SPECON, read the *SPEC data; read specified response spectra from .SPC file; read periods and mass participation factors from .MXX file (unit NFMODE) and write to .UXX file (unit NFMRSL); read mode shapes from .MXX file; for each mode shape calculate nodal displacements, element results, section forces and generalized displacements and write to .UXX file; calculate modal amplitudes; write response for each mode to .OUT file; form SRSS combination and write to .OUT file.
13. On completion of the analysis return to CONTRL and read the next separator line.
    a) For *PARAMETERS proceed to Process Analysis Parameters.
    b) For *STOP proceed to End Analysis Session.
    c) For an analysis separator proceed to Identify Analysis Type.

More detailed flow charts and task descriptions for GRSOL, STAT, REST, DYNMIC, MODCON and SPECON are given in the following sections.

3.11.2. *GRAV Analysis - Subroutine GRSOL

The flow chart for this subroutine is as follows.



The following main tasks are performed.

1. In PRLOD, write the gravity load vector, XLOD (see KXLOD in /STOR/), to .ECH file (unit IOU).
2. In UPDATS, form EFFK and factorize (see KEFFK in /STOR/).
3. In OPTSOL, solve for DDIS (see KDDS in /STOR/).
4. In GRSOL, update DIS (see KDIS in /STOR/).
5. In COLCHK, check if collapse displacements (see /COLPSE/) have been exceeded. If so, set IRDOF, IREVNT, IRNOD and IQUIT (see /EVENT/).
6. In GRSOL, set KSAVE, KPPSV, KPOUT, KENVS and KENVP (see /OUTP/).
7. In DISSAV:

a) Update nodal displacement envelopes (see KENP, KENN, KISTP, KISTN in /STOR/).

b) If KENVS=1, write nodal displacement envelopes for post-processing to .EXX file (unit NFENVP).

c) If KENVP=1, write nodal displacement envelopes for printout to .OUT file (unit NFOUTP).

d) If KPPSV=1, write nodal displacements for post-processing to .RXX file (unit NFRES).

e) If KPOUT=1, write nodal displacements for printout to unit NFPRNT.

8. In RESPON, initialize SEFOR, RINTS and RINT to zero (see KSEFOR, KRINTS and KRINT in /STOR/); and set KSAV=MAX(KPPSV,KPOUT).
9. In ADRESS, copy /INFGR/ and/or /INFEL/ from INFB for current element group and element.
10. In RESPXX, call element subroutine RESP##, where ## is the element type number (see KELEM in /GENINF/).
11. In RESP##, update element state (i.e, /INFEL/ block); calculate static and P-Δ work done in the element; calculate element resisting force vectors RELAS and RINIT; if KSAV=1 put element results in /THELM/.
12. In SEFORC, assemble RELAS and RINIT into structure section forces, SEFOR.
13. In RESPON, assemble RELAS and RINIT into RINTS and RINT.
14. In RESPON, update ENER and EEXT (see /ENRGY/) for work done in the element.
15. In STIFXX, call element subroutine STIF##, where ## is the element type number (see KELEM in /GENINF/).
16. In STIF##, calculate change in element stiffness, FK.
17. In ASSEM, assemble FK into TANK (see KTANK in /STOR/).
18. In RESPON, if KENVP=1, call ENPRXX for each element.

19. In ENPRXX, call element subroutine ENPR##, where ## is the element type number (see KELEM in /GENINF/).

20. In ENPR##:

    a) for first element, write heading for element envelopes to .OUT file.

    b) write element envelopes to .OUT file.

21. In RESPON:

    a) If KPPSV=1, write element results (from /THELM/) for post-processing to .RXX file (unit NFRES).

    b) If KPOUT=1, write element results (from /THELM/) for printout to unit NFPRNT.

22. In RESPON, if KENVP=1 call PREWRK.

23. In PREWRK, write energy log for each element group (see ENER, ENED and EEXT in /ENRGY/) to .OUT file.

24. In SECSAV:

    a) Transform section forces, SEFOR, from global axes to section axes.

    b) Update section envelopes (see KSECEN, KISECE in /STOR/).

    c) If KENVS=1, write section envelopes for post-processing to .EXX file (unit NFENVP).

    d) If KENVP=1, write section envelopes for printout to .OUT file (unit NFOUTP).

    e) If KPPSV=1, write section forces for post-processing to .RXX file (unit NFRES).

    f) If KPOUT=1, write section forces for printout to unit NFPRNT.

25. In RESPON, assemble support spring forces into RINTS and RINT.

26. In GENSAV:

    a) Form generalized displacements, RDS, corresponding to DIS (see KDIS in /STOR/).

b) Update generalized displacement envelopes (see KRDSEN and KIRDSE in /STOR/).

c) If KENVS=1, write generalized displacement envelopes for post-processing to .EXX file (unit NFENVP).

d) If KENVP=1, write generalized displacement envelopes for printout to .OUT file (unit NFOUTP).

e) If KPPSV=1, write generalized displacements for post-processing to .RXX file (unit NFRES).

f) If KPOUT=1, write generalized displacements for printout to unit NFPRNT.

27. In UPDATS, update effective stiffness, EFFK.

28. In EQBM, compute unbalanced load, RU (see KRU in /STOR/); determine UNBL, UNBF and UNBM (in /EVENT/).

29. In PRSTAT, write solution log to .ECH file (unit IOU).

30. If KENR≠0 (in /INDIC), then

a) In ENERS, perform energy balance computations.

b) In PRLOG, write energy log to .SLO file (unit NFLOG).

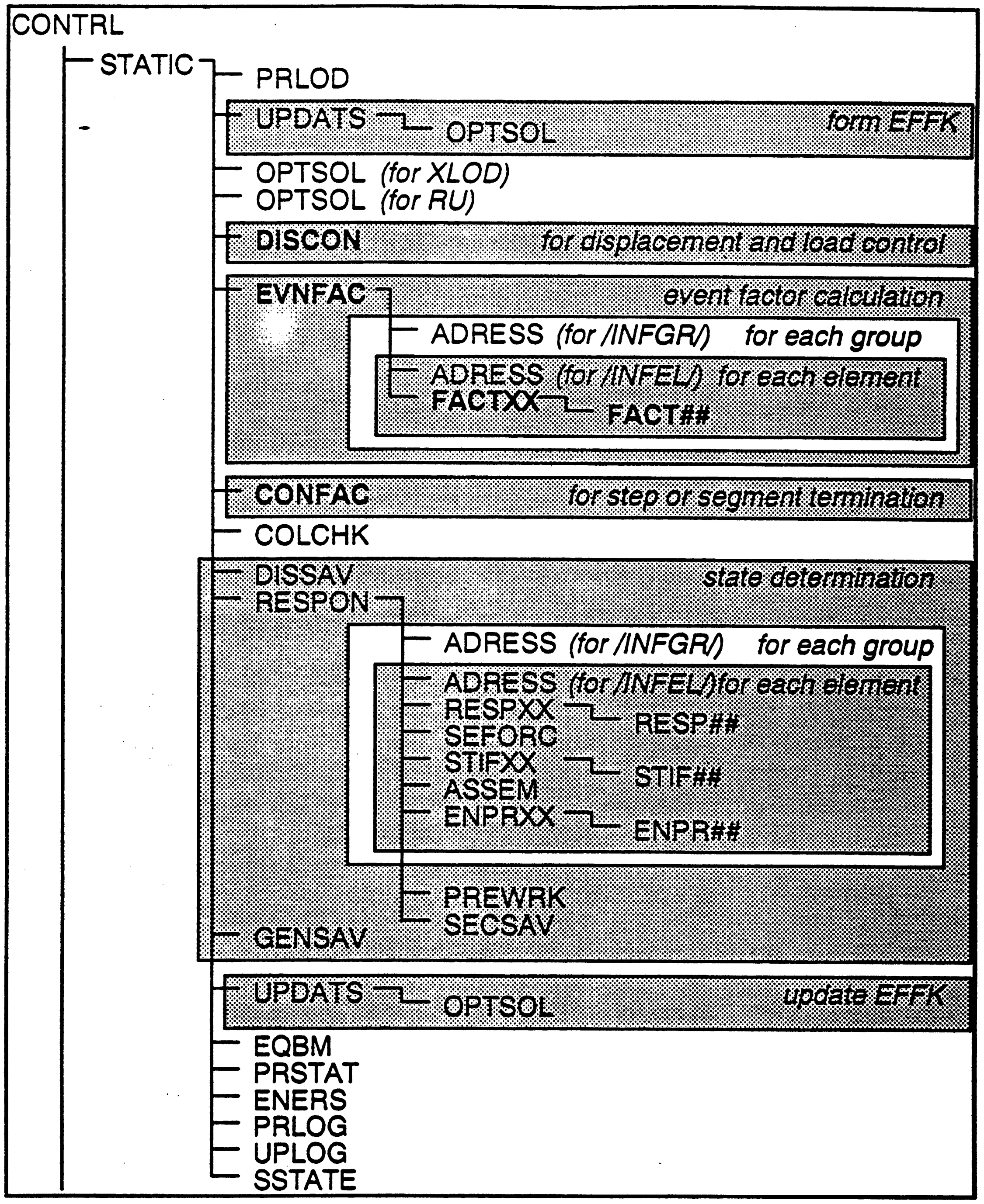
31. If KSAVE=1 (in /OUTP/), then

a) In UPLOG, open .SXX file (unit NFPERM), where XX is the analysis segment number.

b) In SSTATE, write structure state to .SXX file.

3.11.3. *STAT Analysis - Subroutine STATIC

The flow chart for this subroutine is as follows.



STATIC is very similar to GRSOL. The new subroutines in STATIC are shown in bold characters in the flow chart.

The loading for a *STAT analysis is applied in one or more steps, controlled by load factor and/or displacement increments. Within each step an event-to-event solution strategy is used, dividing each step into substeps at each event.

The following main tasks are performed in each substep.

1. In OPTSOL, solve for displacement increment, DDIS, due to XLOD, the load corresponding to a unit load factor.
2. In OPTSOL, solve for the displacement increment, RU, due to unbalanced load at the end of the preceding substep.
3. In DISCON:
   a) Calculate the proportions FF and FU of DDIS and RU, respectively, that must be applied to satisfy the load control and displacement control conditions.
   b) Update IFLIP, IPSIGN, ISIGN and MAXFP in /STAT/; and IREVNT in /EVENT/.
4. In STATIC, combine displacment increments as follows.
$$\text{DDIS} = \text{FF} \times \text{DDIS} + \text{FU} \times \text{RU}$$
5. In EVNFAC, calculate the smallest event factor for any element, FACMIN; and set variables IRELM, IREVNT and IRGRP in /EVENT/.
6. In FACTXX, call element subroutine FACT##, where ## is the element type number (see KELEM in /GENINF/).
7. In FACT##, calculate the smallest event factor for the element.
8. In STATIC, scale DDIS to event.
$$\text{DDIS} = \text{FACMIN} \times \text{DDIS}$$
9. In STATIC, update AFAC and FACC in /EVENT/; and FDIS and FLOD in /STAT/.
10. In CONFAC:

a) Set IEVEN=0 if step is complete because SLFAC or DISA (see /STAT/) has been reached.

b) Set IQUIT=1 if segment is complete because TLOAD or DISMA (see /STAT/) has been reached.

c) Set IQUIT=-1 if NSTEPS, MAXFP or MAXEV have been reached (see /EVENT/, /STAT/ and /TIME/).

11. In COLCHK, check if collapse displacements (see /COLPSE/) have been exceeded. If so, set IRDOF, IREVNT, IRNOD and IQUIT (see /EVENT/).
12. In STATIC, update variables in /OUTP/.
13. Perform state determination; update EFFK; perform unbalanced load computations and write solution log to .ECH file; perform energy balance computations and write energy log to .SLO file; and write structure state as for *GRAV analysis.
14. In STATIC:

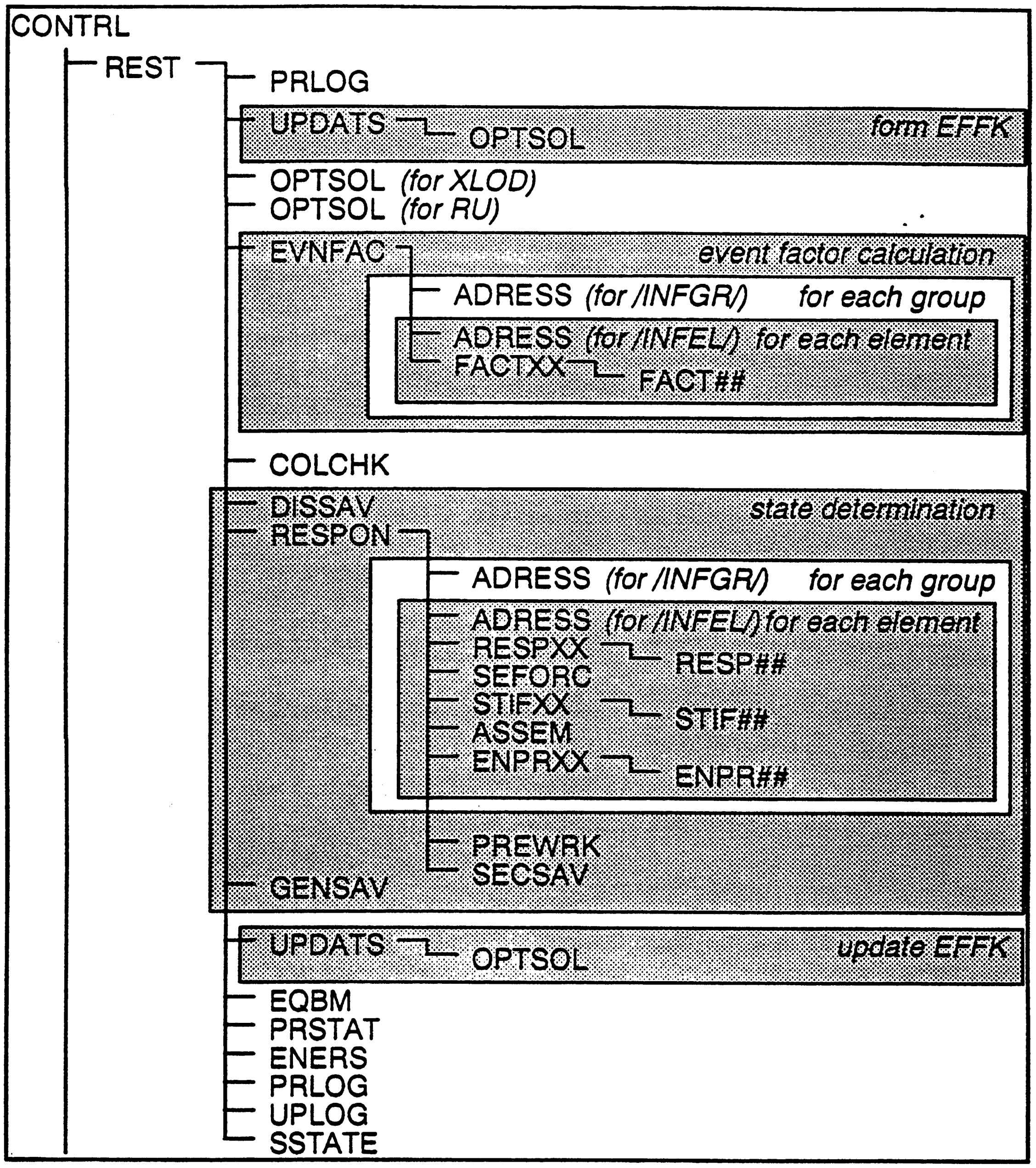
a) If IQUIT=0, proceed to next substep (if IEVEN=1) or step (if IEVEN=0).

b) If IQUIT=1, segment has been completed successfully. Return to CONTRL.

c) If IQUIT=-1, segment could not be completed. Return to CONTRL.

3.11.4. *REST Analysis - Subroutine REST

The flow chart for this subroutine is as follows.



REST is very similar to STATIC, and there are no new subroutines.

The following initial tasks are performed.

1. In REST:

   a) Compute XLOD as follows (see KXLOD, KEXTS, KRINTS and KRU in /STOR/).

$$XLOD = EXTS - RINTS - RU$$

b) Modify external loads as follows.

EXTS=RINTS

EXT=RINTS

c) Set VEL and ACC to zero (see KVEL, KACC in /STOR/).

d) Modify energy variables as follows (see /ENRGY/).

TEP = TEP + TEW - TEI - TED

TEE = TEP + TESO

TEK = 0.0

TEI = 0.0

TEW = 0.0

TEAD = 0.0

ENRD = 0.0

e) Set second column of SPDSP (see KSPDSP in /STOR/) to zero.

2. In PRLOG, write starting energy log to .SLO file (unit NFLOG).
3. In UPDATS, form EFFK (see KEFFK in /STOR/).

In *REST analysis XLOD corresponds to a load factor of unity, and is applied in a single step. Within the step, an event-to-event solution strategy is used, dividing the step into substeps at each event.

The following main tasks are performed within each substep.

1. In OPTSOL, solve for displacement increment, DDIS, due to XLOD.
2. In OPTSOL, solve for displacement increment, RU, due to unbalanced load at end of preceding substep.
4. In REST, combine displacment increments, as follows.

   DDIS = FF × DDIS + RU

   where FF is the remaining load factor.
5. In EVNFAC, calculate smallest event factor for any element, FACMIN; and set variables IRELM, IREVNT and IRGRP in /EVENT/.

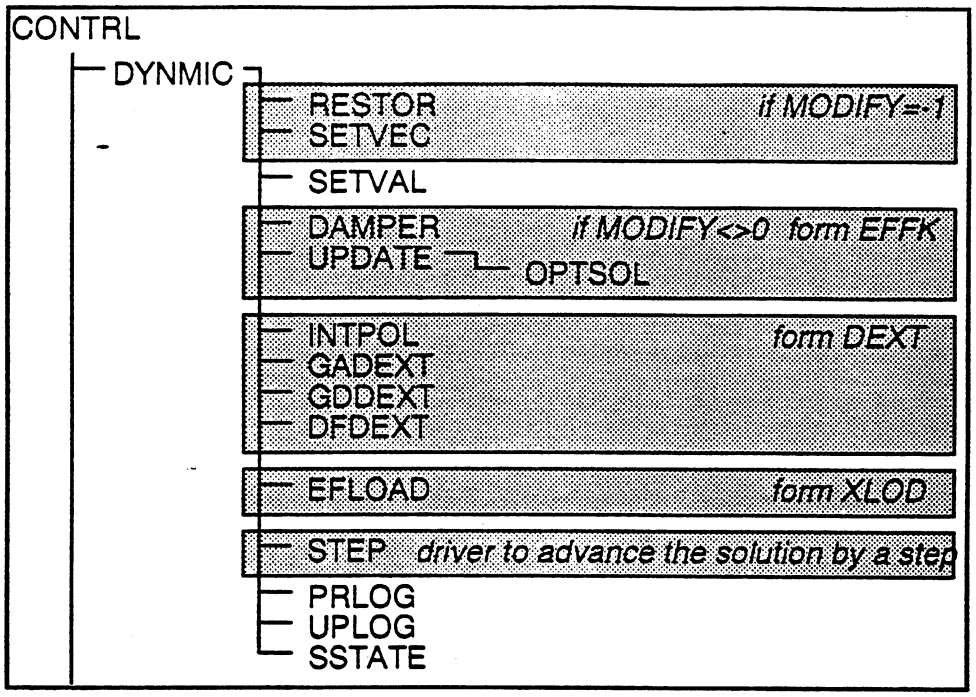
6. In REST, if FACMIN < 1.0 set IEVEN = 1, otherwise set IEVEN = 0 and IQUIT=1.
7. In REST, scale DDIS to event.

$$\text{DDIS} = \text{FACMIN} \times \text{DDIS}$$

8. In REST, update FACC in /EVENT/; and FLOD in /STAT/; and the remaining load factor, FF.
9. In REST, set IREVNT=8 and IQUIT=-1, if MAXEV has been exceeded (see /EVENT/).
10. In COLCHK, check if collapse displacements (see /COLPSE/) have been exceeded. If so, set IRDOF, IREVNT, IRNOD and IQUIT (see /EVENT/).
11. In REST, set KSAVE, KPPSV, KPOUT, KENVS and KENVP in /OUTP/.
12. Perform state determination; update EFFK; perform unbalanced load computations and write solution log to .ECH file; perform energy balance computations and write energy log to .SLO file; and write structure state; as for *STAT analysis.
13. In REST:
    a) If IQUIT=0 and IEVEN=1, proceed to next substep.
    b) If IQUIT=1, segment has been completed successfully. Return to CONTRL.
    c) If IQUIT=-1, segment could not be completed. Return to CONTRL.

3.11.5. Dynamic Analysis - Subroutine DYNMIC

The flow chart for this subroutine is as follows.



The following initial tasks are performed.

1. In DYNMIC, set MODIFY=2 (see /AUTO/) so that EFFK is formed at the start of the analysis.
2. In DYNMIC, set ISSAV=0 to indicate that a back-up of INFB, TANK and RU has not been made (see KINFB, KTANK, KRU, KDINFB, KDTAN and KDRU in /STOR/ and Section 2.5.5).

For each time step the following tasks are performed.

1. In DYNMIC, initialize ERRS, ERRI (in /AUTO/) to zero.
2. If MODIFY=-1 (i.e., preceding time step was aborted and a new step has to be taken with a smaller step size), do the following.
   a) If ISSAV=1, in RESTOR, restore INFB, TANK and RU from the back-up to the values at the start of the step. If ISSAV=0, no restoration is necessary as the time step was aborted in the first substep.

b) In SETVEC, restore TTSAVE, TTPSV, TTPOUT and TTENVP in /OUTP/ to the values at the start of the step.

c) In SETVEC, do the following (see /STOR/) so that EXT, DIS, VEL and ACC have values corresponding to the start of the step.

EXT = EXT - DEXT

DIS = DISI

VEL = VELI

ACC = ACCI

c) In SETVEC, restore work quantities in /ENRGY/ to the values corresponding to the start of the step.

d) In DYNMIC, for ground displacement analysis, set total displacement in SPDSP to the values corresponding to the start of the step.

3) In SETVAL update variables in /OUTP/; and set the following for a new step.

DISI = DIS

VELI = VEL

ACCI = ACC

ENRI = 0

4) If MODIFY $\neq$ 0 (see /AUTO/):

a) In DAMPER, update integration coefficients in /INTCOF/.

b) In UPDATE, form effective stiffness, EFFK (see KEFFK in /STOR/).

c) In DYNMIC, set MODIFY=0.

5. To form DEXT (see KDEXT in /STOR/) do the following.

a) In INTPOL, compute increments of ground accelerations (for *ACCN or *ACCR), ground displacements (for *DISN or *DISR) or dynamic forces (for *FORN or *FORR) from the dynamic force records in TASDF in /ASDFRC/.

b) In DYNMIC, initialize DEXT to zero.

c) In GADEXT, assemble contributions due to ground acceleration increments in DEXT for *ACCN or *ACCR.

d) In GDDEXT, assemble contributions due to ground displacement increments in DEXT for *DISN or *DISR.

e) In DFDEXT, assemble contributions due to dynamic force increments in DEXT for *FORN or *FORR.

6. In EFLOAD, form effective load, XLOD (see KXLOD in /STOR/), for current step.
7. In STEP, do either of the following.

   a) Advance the solution by a step and set MODIFY=1 if the time step size is to be increased for the next step.

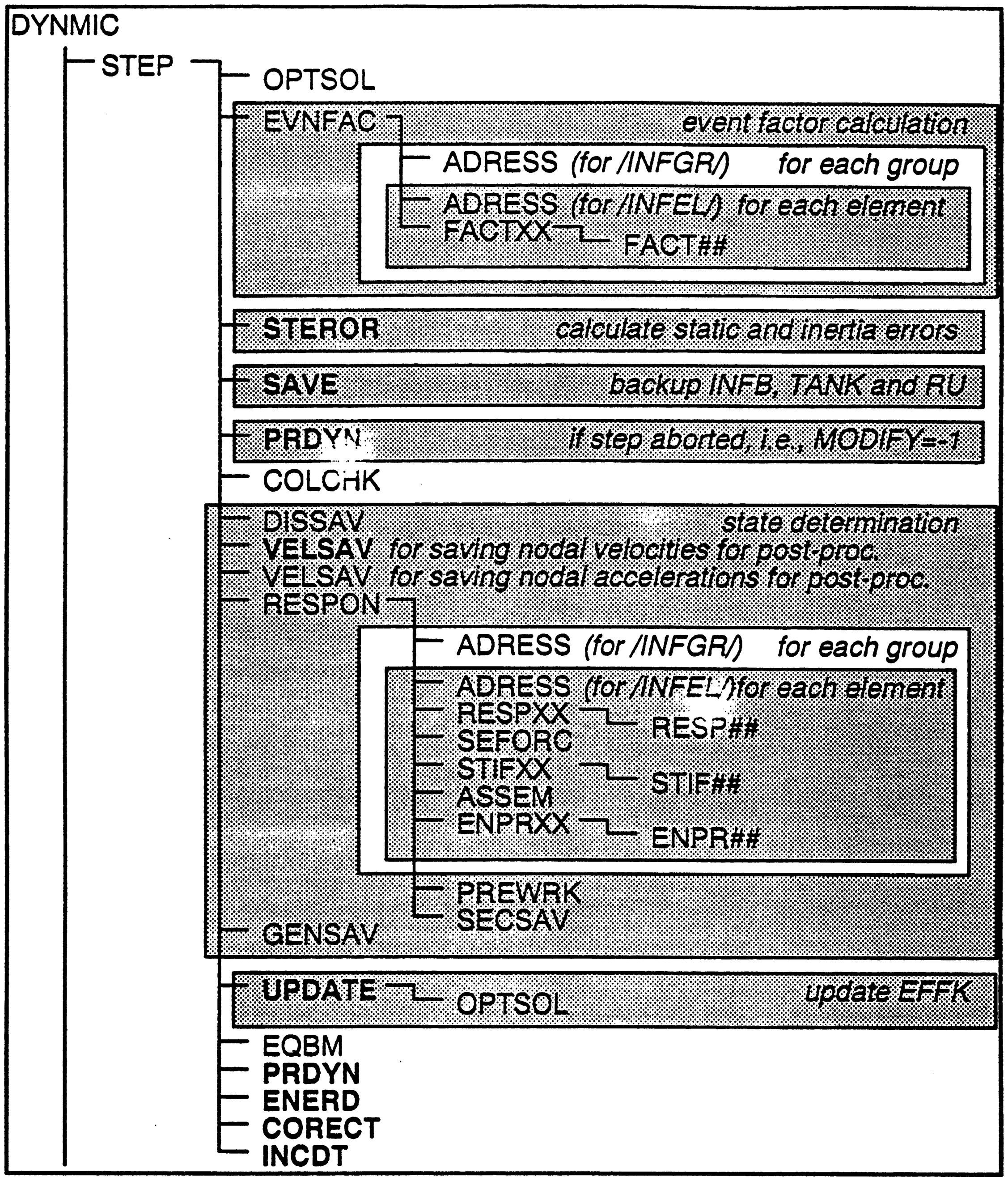
   b) Abort the step; set MODIFY=-1; and set ISSAV=1 if the step was aborted after the first substep. If ISSAV=1, a back-up copy of INFB, TANK and RU has been made in memory or on file.

8. In DYNMIC, if MODIFY =-1, repeat the step with a reduced time step size (i.e., go to step 1).
9. In DYNMIC, set ISSAV=0 for the next step.
10. In PRLOG, write energy log to .SLO file (unit NFLOG).
11. If KSAVE=1 (in /OUTP/), then

    a) In UPLOG, open .SXX file (unit NFPERM), where XX is the analysis segment number.

    b) In SSTATE, write structure state to .SXX file.

The flow chart in subroutine STEP is as follows.


DYNMIC
STEP
OPTSOL
EVNFAC
event factor calculation
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/) for each element
FACTXX
FACT##
STEROR
calculate static and inertia errors
SAVE
backup INFB, TANK and RU
PRDYN
if step aborted, i.e., MODIFY=-1
COLCHK
DISSAV
state determination
VELSAV for saving nodal velocities for post-proc.
VELSAV for saving nodal accelerations for post-proc.
RESPON
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/) for each element
RESPXX
RESP##
SEFORC
STIFXX
STIF##
ASSEM
ENPRXX
ENPR##
PREWRK
SECSAV
GENSAV
UPDATE
OPTSOL
update EFFK
EQBM
PRDYN
ENERD
CORECT
INCDT

STEP is similar to STATIC. The new subroutines in STEP are shown in bold characters in the flow chart.

If KEVEN=1 (in /INDIC/), an event-to-event solution strategy is used, dividing each step into substeps at each event. If KEVEN=0, the step consists of a single substep.

The following main tasks are performed in each substep.

1. In OPTSOL, solve for displacement increment, DDIS, due to XLOD, where XLOD is the remaining effective load.
2. Iin EVNFAC, if KEVEN=1, calculate the smallest event factor for any element, FACMIN; and set variables IRELM, IREVNT and IRGRP in /EVENT/. Otherwise set FACMIN=1.0.
3. In STEP, if FACMIN < 1.0, set IEVEN = 1, otherwise set IEVEN = 0.
4. In STEP, scale DDIS to event:

   $$\text{DDIS} = \text{FACMIN} \times \text{DDIS}$$
5. In STEP, set XLOD to the remaining effective load:

   $$\text{XLOD} = (1.0 - \text{FACMIN}) \times \text{XLOD}$$
6. In STEP, calculate CVEL and CACC for substep (see KCVEL and KCACC in /STOR/).
7. If KAUTO=2, then
   a) In STEROR, calculate static force (i.e., midstep equilibrium) and inertia force (i.e., equivalent impulse) errors and update ERRS and ERRI in /AUTO/.
   b) If ERRS < TOLHIS, ERRI < TOLHIS, IEVEN=1, and ISSAV=0, then in SAVE, backup INFB, TANK and RU, and in STEP, set ISSAV=1.
   c) If ERRS > TOLHIS or ERRI > TOLHII, then in STEP, set

   $$\text{DTNEW} = \text{DT x DTRED}$$
   $$\text{DTOLD} = \text{DT}$$
   $$\text{DT} = \text{DTNEW}$$
   $$\text{MODIFY} = -1$$

   (see /AUTO/); in PRDYN, write solution log to .ECH file; and return to DYNMIC to repeat the step with a reduced step size.
8. In STEP, update DIS, VEL and ACC.

9. In STEP, if IEVEN=1 (i.e., not last substep), set KSAVE, KPPSV, KPOUT, KENVS and KENVP in /OUTP/ to zero.
10. If IEVEN=0 (i.e., last substep), then:
    a) In COLCHK, check if collapse displacements (see /COLPSE/) have been exceeded. If so, set IRDOF, IREVNT, IRNOD and IQUIT (see /EVENT/).
    b) In STEP, set KSAVE, KPPSV, KPOUT, KENVS and KENVP in /OUTP/.
11. In DISSAV, update nodal displacement envelopes (see KENP, KENN, KISTP, KISTN in /STOR/); if KENVS=1, write nodal displacement envelopes for post-processing to .EXX file (unit NFENVP); if KENVP=1, write nodal displacement envelopes for printout to .OUT file (unit NFOUTP); if KPPSV=1, write nodal displacements for post-processing to .RXX file (unit NFRES); if KPOUT=1, write nodal displacements for printout to unit NFPRNT.
12. In VELSAV, if KPPSV=1, write nodal velocities and accelerations for post-processing to .RXX file (unit NFRES).
13. In RESPON, initialize SEFOR, RINTS and RINT to zero (see KSEFOR, KRINTS and KRINT in /STOR/); and set KSAV=MAX(KPPSV,KPOUT).
14. In ADRESS, copy /INFGR/ and/or /INFEL/ blocks for current element from INFB.
15. In RESPXX, call element subroutine RESP##, where ## is the element type number.
16. In RESP##, update element state (i.e, /INFEL/ block); calculate static, damping and P-Δ work done in element; calculate element end resisting forces, RELAS, RDAMP and RINIT; if KSAV=1, put element results in /THELM/.
17. In SEFORC, assemble RELAS, RDAMP and RINIT into structure section forces, SEFOR.
18. In RESPON, assemble RELAS, RDAMP and RINIT into RINTS and RINT.
19. In RESPON, update ENER, ENED and EEXT (see /ENRGY/) for work done in the element.

20. In STIFXX, call element subroutine STIF##, where ## is the element type number.
21. In STIF##, calculate change in element stiffness, FK.
22. In ASSEM, assemble FK into TANK (see KTANK in /STOR/).
23. In RESPON, if KENVP=1 call ENPRXX for each element.
24. In ENPRXX, call element subroutine ENPR##, where ## is the element type number.
25. In ENPR## write element envelopes to .OUT file.
26. In RESPON:
    a) If KPPSV=1, write element results (from /THELM/) for post-processing to .RXX file (unit NFRES).
    b) If KPOUT=1, write element results (from /THELM/) for printout to unit NFPRNT.
27. In RESPON, if KENVP=1 call PREWRK.
28. In PREWRK, write energy log for each element group (see ENER, ENED and EEXT in /ENRGY/) to .OUT file.
29. In SECSAV, transform section forces, SEFOR, from global axes to section axes; update section envelopes (see KSECEN, KISECE in /STOR/); if KENVS=1 write section envelopes for post-processing to .EXX file (unit NFENVP); if KENVP=1 write section envelopes for printout to .OUT file (unit NFOUTP); if KPPSV=1 write section forces for post-processing to .RXX file (unit NFRES); and if KPOUT=1 write section forces for printout to unit NFPRNT.
30. In RESPON, update RINTS and RINT for support spring forces.
31. In GENSAV, form generalized displacements, RDS, corresponding to DIS (see KDIS in /STOR/); update generalized displacement envelopes (see KRDSEN and KIRDSE in /STOR/); if KENVS=1 write generalized displacement envelopes for post-processing to .EXX file (unit NFENVP); if KENVP=1 write generalized displacement envelopes for printout to .OUT file (unit NFOUTP); if KPPSV=1 write generalized

displacements for post-processing to .RXX file (unit NFRES); and if KPOUT=1 write generalized displacements for printout to unit NFPRNT.

32. In UPDATE, update effective stiffness, EFFK due to changed TANK.
33. In STEP, assemble inertia forces and alpha-M damping forces in RINT.
34. In STEP, update EXT due to applied DEXT in the substep.
35. In EQBM, compute unbalance load, RU (see KRU in /STOR/); determine UNBL, UNBF and UNBM (in /EVENT/).
36. In PRDYN, write solution log to .ECH file (unit IOU).
37. If KENR$\neq$0 (in /INDIC), in ENERD, perform energy balance computations.
38. If IEVEN=1, in STEP, go to next substep.
39. If IEVEN=0, then
    a) In CORECT, if KENRC=1 (in /INDIC/) perform velocity correction.
    b) In CORECT, if KEQBC=1 (in /INDIC/), perform acceleration modification to improve equilibrium.
    c) In STEP, compute kinetic energy for energy log.
    d) If KAUTO=2 (in /AUTO/), in INCDT , increase time step size for next step and set MODIFY=1.

3.11.6. *MODE Analysis - Subroutine MODCON

a) Theory

The eigenproblem to be solved is

$$\underline{K}\underline{\phi} = \omega^2 \underline{M}\underline{\phi} \tag{3.1}$$

where $\underline{K}$ is the stiffness matrix, $\underline{M}$ is the diagonal mass matrix with possible zero terms on the diagonal, $\underline{\phi}$ is a mode shape, and ω is the circular frequency corresponding to $\underline{\phi}$. The massless DOFs must be eliminated from Equation 3.1, which can be written in partitioned form as follows.

$$\begin{bmatrix} \underline{K}_{dd} & \underline{K}_{d0} \\ \underline{K}_{d0}^T & \underline{K}_{00} \end{bmatrix} \begin{Bmatrix} \underline{\phi}_d \\ \underline{\phi}_0 \end{Bmatrix} = \omega^2 \begin{bmatrix} \underline{M}_d & \underline{0} \\ \underline{0} & \underline{0} \end{bmatrix} \begin{Bmatrix} \underline{\phi}_d \\ \underline{\phi}_0 \end{Bmatrix} \tag{3.2}$$

where $\underline{\phi}_d$ and $\underline{\phi}_0$ are the mode shape components at the mass and massless DOFs, respectively, and $\underline{M}_d$ is a diagonal mass matrix. Using static condensation, one can obtain, the reduced eigenproblem.

$$\left(\underline{K}_{dd} - \underline{K}_{d0}\underline{K}_{00}^{-1}\underline{K}_{d0}^T\right)\underline{\phi}_d = \underline{K}_d\underline{\phi}_d = \omega^2\underline{M}_d\underline{\phi}_d \tag{3.3}$$

In DRAIN, instead of calculating $\underline{K}_d$, the flexibility matrix $\underline{F}_d \equiv \underline{K}_d^{-1}$ is obtained by ***in effect*** solving

$$\begin{bmatrix} \underline{K}_{dd} & \underline{K}_{d0} \\ \underline{K}_{d0}^T & \underline{K}_{00} \end{bmatrix} \begin{bmatrix} \underline{F}_d \\ \underline{F}_0 \end{bmatrix} = \begin{bmatrix} \underline{I} \\ \underline{0} \end{bmatrix} \tag{3.4}$$

where $\underline{I}$ is a unit matrix. Although the DOFs are partitioned in Equation 3.4, there is no need for this in the implementation. In DRAIN, $\underline{F}_d$ is obtained a column at a time, by successively applying unit loads at each mass DOF, and extracting the displacements produced at mass DOFs.

Equation 3.3 is converted to standard form

$$\underline{\bar{F}}_d\,\underline{\bar{\phi}}_d = \frac{1}{\omega^2}\underline{\bar{\phi}}_d \tag{3.5}$$

where

$$\underline{\bar{F}}_d = \underline{M}_d^{1/2}\underline{F}_d\underline{M}_d^{1/2} \tag{3.6}$$

and

$$\underline{\bar{\phi}}_d = \underline{M}_d^{1/2}\underline{\phi}_d \tag{3.7}$$

Equation 3.5 is solved by Hessenberg's QR iteration in subroutine HQRWT, and self-orthonormal eigenvectors, $\underline{\tilde{\Phi}}_d$, are obtained. That is

$$\underline{\tilde{\Phi}}_d^T \underline{\tilde{\Phi}}_d \equiv \underline{\Phi}_d^T \underline{M}_d^{1/2} \underline{M}_d^{1/2} \underline{\Phi}_d \equiv \underline{\Phi}_d^T \underline{M}_d \underline{\Phi}_d = \underline{I} \tag{3.8}$$

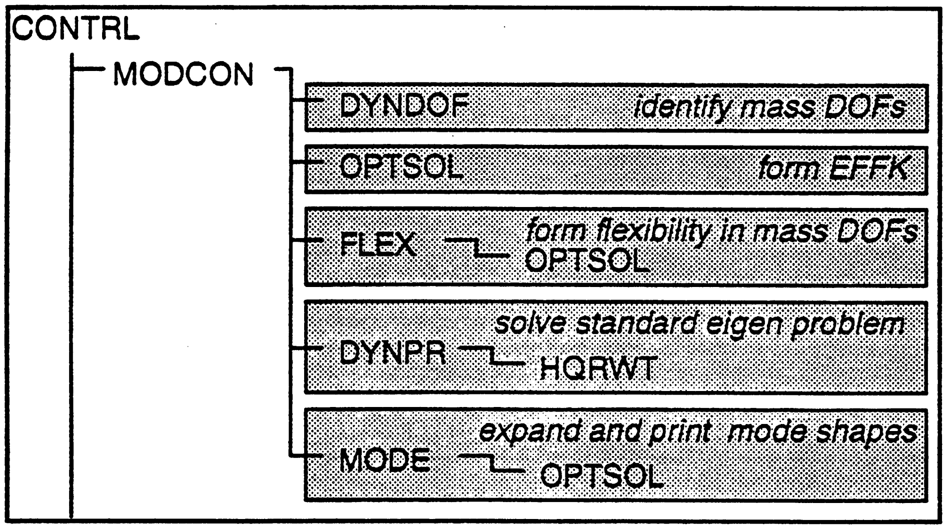
Each eigen vector $\underline{\tilde{\phi}}_d$ is premultiplied by $\underline{M}_d^{1/2}$ to obtain $\underline{M}_d \underline{\phi}_d$, and the full mode shape is obtained by ***in effect*** solving

$$\frac{1}{\omega^2}\begin{bmatrix} \underline{K}_{dd} & \underline{K}_{d0} \\ \underline{K}_{d0}^T & \underline{K}_{00} \end{bmatrix} \begin{Bmatrix} \underline{\phi}_d \\ \underline{\phi}_0 \end{Bmatrix} = \begin{Bmatrix} \underline{M}_d \underline{\phi}_d \\ \underline{0} \end{Bmatrix} \tag{3.9}$$

In DRAIN, the loading terms are placed in rows corresponding to mass DOFs and the solution vector is $\frac{1}{\omega^2}\underline{\phi}$.

b) Implementation

The flow chart for MODCON is as follows.


CONTRL
MODCON
DYNDOF
identify mass DOFs
OPTSOL
form EFFK
FLEX
form flexibility in mass DOFs
OPTSOL
DYNPR
solve standard eigen problem
HQRWT
MODE
expand and print mode shapes
OPTSOL

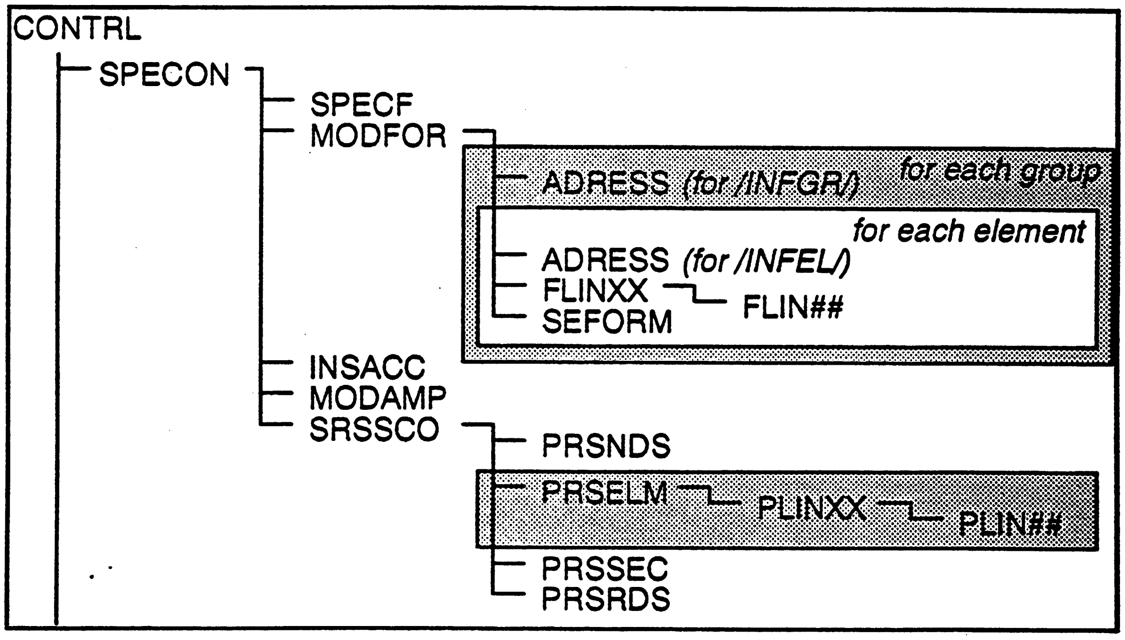
The following main tasks are performed.

1. In MODCON, read *MODE data.
2. In DYNDOF, identify the mass DOFs.

3. In FLEX, form flexibility matrix, $\underline{F}_d$.
4. In DYNPR, form $\overline{\underline{F}}_d$.
5. In $\overset{-}{H}$QRWT, solve for the eigenvectors, $\tilde{\underline{\Phi}}_d$.
6. In DYNPR, write mode periods to .OUT file (unit NFOUTP).
7. In DYNPR, form unit modal loads, $\underline{M}_d\underline{\phi}_d$.
8. In DYNPR, compute mass participation factors in translational directions and write to .OUT file.
9. In MODE, calculate expanded mode shapes, $\underline{\phi}/\omega^2$.
10. In MODE, write unit mode shapes to .OUT file.
11. In MODE, write periods, mass participation factors and mode shapes to .MXX file (unit NFMODE).

3.11.7. *SPEC Analysis - Subroutine SPECON

The flow chart for this subroutine is as follows.



The following initial tasks are performed.

1. In SPECON, read the *SPEC data.
2. In SPECF, read applied response spectra into array TASDF in /ASDFRC/.

The following tasks are performed only if a .UXX file has not already been written.

1. In SPECON, read periods and mass participation factors from .MXX file (unit NFMODE) and write to .UXX file (unit NFMRSL).
2. For each mode shape on .MXX file:
   a) In SPECON, read mode shape from .MXX file.
   b) In MODFOR, calculate nodal displacements and write to .UXX file.
   c) In MODFOR, initialize section forces, SEFOR, to zero.
   d) In element subroutine FLIN##, calculate element results for each element.
   e) In SEFORM, assemble element contribution to section forces, SEFOR.
   f) In MODFOR, write element results for each element group to .UXX file.
   g) In MODFOR, transform section forces, SEFOR, from global axes to section axes.
   h) In MODFOR, write section forces, SEFOR, to .UXX file.
   i) In MODFOR, calculate generalized displacements and write to .UXX file.

The following tasks are performed to complete the analysis.

1. In SPECON, read periods and mass participation factors from .UXX file.
2. In INSACC, calculate spectral acceleration for each mode.
3. In MODAMP, calculate modal amplitudes.
4. In SRSSCO, calculate response (i.e., nodal displacements, element results, section forces and generalized displacements) for each mode, and for SRSS combination; and write to .OUT file (unit NFOUTP). SRSSCO reads individual modal results from .UXX file. PLIN## prints the response for each element.

3.12. END SESSION

The analysis session ends with one of the following conditions.

1. IQUIT=-1 : last analysis segment could not be completed.
2. *STOP separator is read from input file.
3. Program stops because of errors in the input file or insufficient memory.

The flow chart for this phase is as follows.

```
MAIN ─┐
      └─ CONTRL
```

The following tasks are performed.

1. In CONTRL, write farewell message to the .ECH file (unit IOU).
2. In MAIN, close files and stop program execution.

4. STRUCTURE OF PERMANENT FILES -- DRAIN-2DX AND 3DX

4.1. INTRODUCTION

In section 2.4, the permanent files created by DRAIN-2DX and 3DX were listed. Some of these files are binary files that may be used by other programs, particularly for post-processing. In this chapter, the structures of these files are described in detail.

4.2. FILES FOR LOAD PATTERNS AND LOAD RECORDS

4.2.1. .ELD File - Element Load Patterns

The .ELD file contains static element load patterns, each specified by *ELEMLOAD data. Each new pattern is appended to the .ELD file, in subroutine INGPAT. The applied patterns are retrieved for *GRAV analyses in subroutine INELOD.

Each pattern consists of a number of FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
PATIT	character*40	Pattern title.
NGRPL	integer	Number of element groups loaded by the pattern.

2. For each loaded element group, two records as follows.

   a) First record.

Variable	Type	Description
IGRC	integer	Element group number.
NLOD	integer	Number of element load sets for this group.

b) Second record.

Array	Description
SETLOD(ninl,nlod)	Element load sets. Each column of SETLOD contains a load set. NINL is the length of a load set from array NINFL in /GENINF/. NLOD is the number of element load sets.
ELFACT(nlod,nmem)	Load set scale factors for elements of the group. Each column of ELFACT contains the load set scale factors for one element. NMEM is the number of elements from array NELEM in /GENINF/.

4.2.2. .STA File - Static Load Patterns

The .STA file contains static nodal load patterns, each specified by *NODALOAD data. Each new pattern is appended to the .STA file, in subroutine INSPAT. The applied patterns are retrieved for *GRAV or *STAT analysis in subroutine INNLOD.

Each pattern consists of two FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
PATIT	character*40	Pattern title.

2. Second record.

Array	Type	Description
XPAT(3,nnods) in 2DX XPAT(6,nnods) in 3DX	real	Nodal loads for the pattern. NNODS is total number of nodes (in /CONTR/).

4.2.4. .ACC, .DIS and .FRC Files - Dynamic Load Records

The .ACC, .DIS and .FRC files contain ground acceleration, ground displacement and dynamic force records, respectively. Each record is specified by *ACCNREC, *DISPREC or *FORCREC data. Each new record is appended to the corresponding file in subroutine

INAXL. The records are later retrieved in subroutine ADFRB for application in dynamic (*ACCN, *ACCR, *DISN, *DISR, *FORN, or *FORR) analysis.

The data for each record consists of a large number of time-value pairs. This data is divided into blocks, each containing 121 time-value pairs. The first pair in any block is the same as the last pair in the preceding block (see /ASDFRC/, /ADFREC/ and /LOADP).

Each record consists of a number of FORTRAN records as follows:

1. First record.

Variable	Type	Description
PATID	character*4	Record name.
PATIT	character*40	Record title.
NREC	integer	Number of (121 time-value pair) blocks for the record.

2. NREC records, each consisting of 121 time-value pairs.

Array	Type	Description
ACCP(2,MPAIRS)	real	Time-value pairs (MPAIRS is in /LOADP/).

4.2.5. .SPC File - Response Spectra

The .SPC file contains response spectra, each specified by *SPECTRUM data. Each new spectrum is appended to the .SPC file in subroutine INSPEC. The applied spectra are retrieved for *SPEC analysis in subroutine SPECF.

Each spectrum consists of two FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Spectrum name.
PATIT	character*40	Spectrum title.
KODS	integer	Spectrum type code. 0 : Acceleration. 1 : Velocity. 2 : Displacement.

2. Second record.

Array	Type	Description
PRESP(2,MPSPEC)	real	Period-response pairs. MPSPEC is in /LOADP/ and equal to 141.

4.2.6. .VEL File - Initial Velocity Patterns

The .VEL file contains initial velocity patterns, each specified by *NODALVEL data. Each new pattern is appended to the .VEL file in subroutine INVPAT. The applied patterns are retrieved for *VELN or *VELR analysis in subroutine INNVEL.

Each pattern consists of two FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
PATIT	character*40	Pattern title.

2. Second record.

Array	Type	Description
XPAT(3,nnods) in 2DX XPAT(6,nnods) in 3DX	real	Nodal velocities for the pattern. NNODS is total number of nodes (in /CONTR/).

4.3. FILES FOR POST-PROCESSING

4.3.1. .GEO File - Structure Geometry

The .GEO file contains structure geometry data. The .GEO file is set up in subroutine INELEM.

The data consists of FORTRAN records as follows.

1. First record.

Variable	Type	Description
NDTP	integer	Number of compound node types +1.
NSNDS	integer	Total number of subnodes in all compound node types.
NNODS	integer	Total number of nodes.
NTNDS	integer	Total number of nodes and subnodes.
NEQQ	integer	Total number of equations + 1.
FNAME	character*8	Problem name.
IHED	character*40	Problem title.

2. If NDTP > 1, one record as follows.

Array	Description
NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type NT is equal to NSB(nt+1)-NSB(nt).
NDFSB(3,nsnds) in 2DX NDFSB(6,nsnds) in 3DX	DOF codes for all subnodes of each compound node type, as follows. 0 : Absolute displacement. 1 : Restrained (not a DOF). 2 : Relative displacement w.r.t. main node. 3 : Special degree of freedom (i.e., not a conventional translation or rotation).
COSB(2,nsnds) in 2DX COSB(3,nsnds) in 3DX	Coordinate offsets from main node for subnodes of each compound node type.

3. One record.

Array	Description
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

4. One record.

Array	Description
COORD(2,nnods) in 2DX COORD(3,nnods) in 3DX	Nodal coordinates, in ascending node number order.

5. One record.

Array	Description
ID(3,ntnds) in 2DX ID(6,ntnds) in 3DX	Equation numbers for displacements at each node and subnode, coded as follows. NEQQ : Restrained displacement. +n : Displacement is unrestrained and unslaved. 'n'= equation number. -n : Displacement is slaved. 'n' = sequence number of master node.

6. Two records for each element group.

   a) First record.

Variable	Type	Description
IGRC	integer	Element group number.
KEL	integer	Element type number.
NMEM	integer	Number of elements.
NELNOD	integer	Number of nodes per element.
IEHD	character*40	Group title.

b) Second record.

Array	Type	Description
IELNOD(nelnod,nmem)	real	Sequence number of nodes for each element.

4.3.2. .EXX File - Result Envelopes

The .EXX files contain result envelopes for post-processing, organized in FORTRAN records as follows.

1. First record written in subroutine EXINIT.

Variable	Type	Description
IHED	character*40	Problem title (in /TITLE/).
FNAME	character*8	Problem name (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).
NSEG	integer	Analysis segment number (in /CONTR/).

2. Second record written in EXINIT.

Variable	Type	Description
NDTP	integer	Number of compound node types +1 (in /CONTR/).
NNODS	integer	Total number of nodes (in /CONTR/).
NTNDS	integer	Total number of nodes and subnodes (in /CONTR/).
NELGR	integer	Number of element groups (in /CONTR/).
NELTOT	integer	Total number of elements (in /CONTR/).
NSSEC	integer	Number of structure sections.
NRDS	integer	Number of generalized displacements.

3. Third record written in EXINIT.

Array	Description
NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type NT is equal to NSB(nt+1)-NSB(nt).
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type number (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

4. Fourth record written in EXINIT.

Array	Description
NELEM(nelgr)	Number of elements in each group.
KELEM(nelgr)	Element type number for each group.
NINFT(nelgr)	Number of output items per element for static or dynamic analyses for each group.

5. The subsequent records consist of result envelopes. Data for each envelope consists of up to 8 records. Records 1 to 4 are written in subroutine DISSAV; if NSSEC > 0, records 5 and 6 are written in subroutine SECSAV; and if NRDS > 0, records 7 and 8 are written in subroutine GENSAV. Presently there is no provision to save element envelopes on the .EXX file.

R.No.	Array	Type	Description
1.	DENP(3,ntnds) in 2DX DENP(6,ntnds) in 3DX	real*4	Positive nodal displacement envelopes.
2.	ISTP(3,ntnds) in 2DX ISTP(6,ntnds) in 3DX	integer	Step numbers for DENP.
3.	DENN(3,ntnds) in 2DX DENN(6,ntnds) in 3DX	real*4	Negative nodal displacement envelopes.
4.	ISTN(3,ntnds) in 2DX ISTN(6,ntnds) in 3DX	integer	Step numbers for DENN
5.	SECENV(3,6,nssec) in 2DX SECENV(6,6,nssec) in 3DX	real*4	Section force envelopes. 2nd index indicates the envelope type as follows. 1 : Total positive. 2 : Total negative. 3 : Static positive. 4 : Static negative. 5 : Damping positive. 6 : Damping negative.
6.	ISECEN(3,6,nssec) in 2DX ISECEN(6,6,nssec) in 3DX	integer	Step numbers for SECENV.
7.	RDSENV(2,nrds)	real*4	Positive and negative generalized displacement envelopes.
8.	IRDSEN(2,nrds)	integer	Step numbers for RDSENV.

4.3.2. .RXX File - Result Histories

The .RXX files contain time or load history results for post-processing. The .RXX file is opened in subroutine EXINIT for each analysis segment and closed in subroutine SEGEND. The data consists of a number of FORTRAN records, as follows.

1. First record written in EXINIT.

Variable	Type	Description
IHED	character*40	Problem title (in /TITLE/).
FNAME	character*8	Problem name (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).
NSEG	integer	Analysis segment number (in /CONTR/).

2. Second record written in EXINIT.

Variable	Type	Description
NDPOST	integer	Number of nodes and subnodes in a post-processing set.
NELTHP	integer	Number of elements in a post-processing set.
NPSECP	integer	Number of structure sections in a post-processing set.
NNRDSP	integer	Number of generalized displacements in a post-processing set.

3. If NDPOST > 0, one record.

Array	Description
INFNOD(3,ndpost)	Information for each node and/or subnode in post-processing set, as follows. 1: Node number. 2: Compound node type number (0 = not a compound node). 3: Subnode number (0 = main node).

4. If NELTHP > 0, one record.

Array	Description
INFELM(4,nelthp)	Information for each element in post-processing set, as follows. 1 : Element group number. 2 : Element type number. 3 : Element number. 4 : Number of output items per element (length of element results).

5. If NPSECP > 0, one record.

Array	Description
INFSEC(npsecp)	Numbers of structure sections in a post-processing set.

6. If NNRDSP > 0, one record.

Array	Description
INFRDS(nnrdsp)	Numbers of generalized displacements in a post-processing set.

7. The subsequent records consist of results for post-processing. The results at each step consists of the following records.

   a) First record, written in subroutine GRSOL, STATIC, REST or STEP.

Variable	Type	Description
KSTEP	integer	Step number.
TIME	real*4	Current time for dynamic analysis segment. Current load factor for static analysis segment.

   b) If NDPOST > 0, one record, written in subroutine DISSAV.

Array	Type	Description
DISP(6,ndpost)	real*4	Node and subnode displacements.

c) If NDPOST > 0, two records for a dynamic analysis segment (ANAL='ACCN', 'ACCR', 'VELN', 'VELR', 'DISN', 'DISR', 'FORN' or 'FORR'), written in subroutine VELSAV.

R.No.	Array	Type	Description
1.	VELO(6,ndpost)	real*4	Node and subnode velocities.
2.	ACCE(6,ndpost)	real*4	Node and subnode accelerations.

d) NELTHP records, written in subroutine RESPON.

Array	Type	Description
THOUT(*)	real*4	Element results. The length of element results depends on the element.

e) If NPSECP > 0, one record, written in subroutine SECSAV.

Array	Type	Description
SECFRC(6,npsecp) in 2DX SECFRC(12,npsecp) in 3DX	real*4	Static and damping section forces. Static followed by damping for each section.

f) If NNRDSP > 0, one record, written in subroutine GENSAV.

Array	Type	Description
GEDISP(nnrdsp)	real*4	Generalized displacements.

4.4. MODAL ANALYSIS FILES

4.4.1. .MXX File - Mode Shapes

The .MXX files contain results from mode shapes and periods analysis. The results consist of the following FORTRAN records, written in subroutine MODE.

1. First record.

Variable	Type	Description
FNAME	character*8	Problem name (in /TITLE/).
IHED	character*40	Problem title (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).

2. Second record.

Variable	Type	Description
NEQ	integer	Length of each mode shape (in /EQNS/).
NVEC	integer	Number of mode shapes.

3. Third record.

Array	Description
EVAL(nvec)	Mode periods.

4. Fourth record.

Array	Description
XLM(2,nvec) in 2DX XLM(3,nvec) in 3DX	Mass participation factors in translational directions for each mode.

5. NVEC records, one for each mode shape.

Array	Description
EVEC(neq)	Mode shape = $\underline{\phi}/\omega^2$. Where, $\underline{\phi}$ is mass normalized mode shape; $\omega = 2\pi/T$; and T = mode period.

4.4.2. .UXX File - Modal Responses

The .UXX files contain responses for unit modal amplitudes. The results consist of the following FORTRAN records, written in subroutine SPECON.

1. First record.

Variable	Type	Description
NVEC	integer	Number of mode shapes.
LENGTH	integer	Minimum length of record buffer required to read the response results.

2. Second record.

Array	Description
EVAL(nvec)	Mode periods.

3. Third record.

Array	Description
XLM(2,nvec) in 2DX XLM(3,nvec) in 3DX	Mass participation factors in translational directions for each mode.

4. NVEC sets of records, one set per mode shape as follows.

   a) One record.

Array	Type	Description
RESNDS(3,ntnds) in 2DX RESNDS(6,ntnds) in 3DX	real*4	Response node and subnode displacements.

   b) NELGR records, one for each element group.

Array	Type	Description
RESELM(nlin,nmem)	real*4	Element response results. NLIN is the number of result items per element for the group (NLINF in /GENINF/). NMEM is the number of elements for the group (NELEM in /GENINF/).

   c) If NSSEC > 0, one record.

Array	Type	Description
RESSEC(3,nssec) in 2DX RESSEC(6,nssec) in 3DX	real*4	Response static section forces.

d) If NRDS > 0, one record.

Array	**Type**	**Description**
RESRDS(nrds)	real*4	Response generalized displacements.

5. DRAIN-BUILDING -- SPECIAL FEATURES

5.1. DRAIN-BUILDING MODEL -- FLOORS AND INTERFLOORS

In all DRAIN programs the structure is modeled as an assemblage of nonlinear elements connected at nodes. In DRAIN-BUILDING there is also a higher level of organization, according to which the structure is modeled as an assemblage of ***floors*** connected by ***interfloors***. The floors and interfloors are then assemblages of elements connected at nodes. Figure 5.1 shows a simple floor and interfloor.

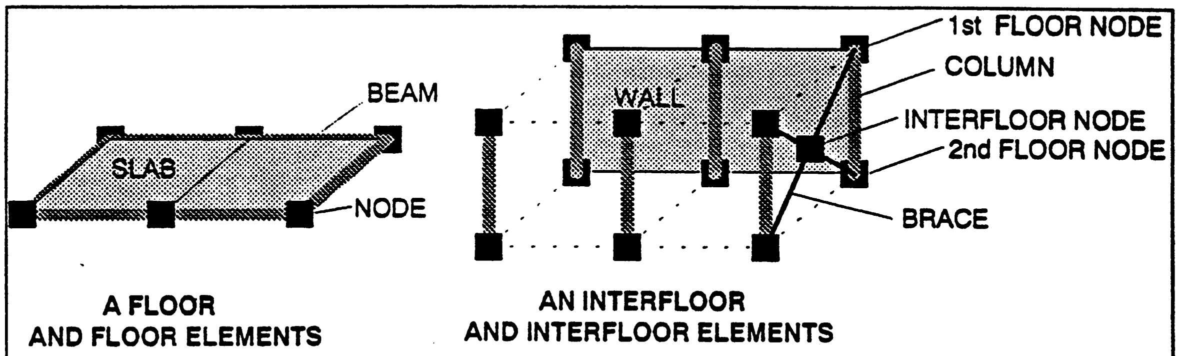


FIG. 5.1. FLOORS AND INTERFLOORS

A floor consists of elements such as slabs and beams, usually but not necessarily lying in a horizontal plane. An interfloor consists of elements such as columns, walls and braces, which connect two floors. In general both floors and interfloors are 3D structures made up of 3D elements. However, a floor could be a 2D structure.

In most cases an interfloor will connect a floor to the floor immediately above or below, as shown in Figure 5.2a. However, an interfloor can connect any two floors, as shown in Figures 5.2b through 5.2d. Note that an interfloor can connect floors at the same level (e.g., an interfloor could be a bridge connecting floors in adjacent towers), as shown in Figure 5.2c. An interfloor can also connect only one floor. For example, in a building on sloping ground the ground nodes can be regarded as belonging to the interfloor rather than a ground "floor". The interfloor is then connected only to the floor above, as shown in Figure 5.2d.

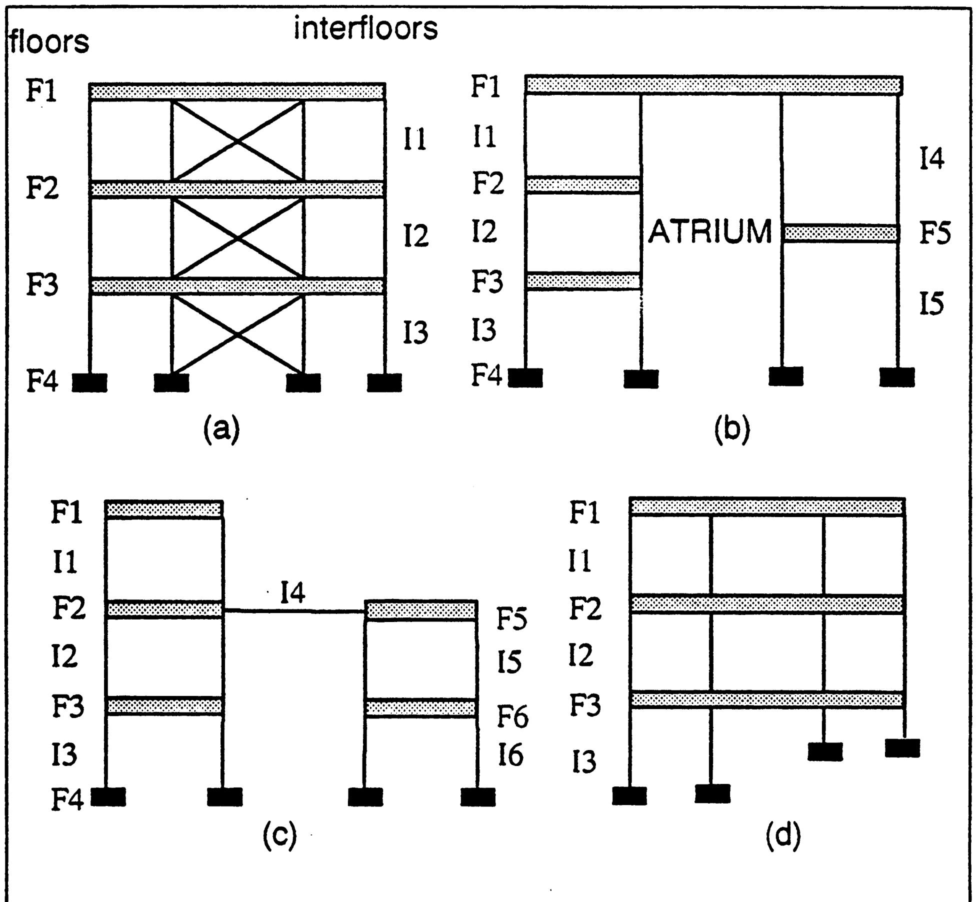


FIG. 5.2. FLOOR AND INTERFLOOR EXAMPLES

Formally, a floor is an assemblage of elements connected only to the nodes of that floor. An interfloor is an assemblage of elements connected to nodes, which belong to (a) floor 1 of the interfloor (the floor above or below), (b) floor 2 of the interfloor (the second of the two connected floors), and/or (c) the interfloor itself.

5.2. FLOOR AND INTERFLOOR TYPES AND INSTANCES

A multistory building may have many identical floors and interfloors, which differ only in their spatial locations. Floors that differ from each other only in their locations can be assigned the same *floor type*. Similarly, interfloors that differ from each other only in their spatial locations can be assigned the same ***interfloor type***. In the input data, floor and interfloor types are defined first. Floor and interfloor ***instances*** are then positioned in space to define the analysis model.

The following main tasks are performed in defining the analysis model.

1. **Define Floor Types:** For each floor type define the following.
   a) Coordinates of floor nodes, relative to a floor origin. These coordinates are used to calculate the dimensions and orientations of the floor elements.
   b) Nodal masses.
   c) Whether the nodal displacements are unrestrained, restrained, spring supported or slaved. A rigid floor diaphragm can be defined by slaving the nodes to a master node for the floor.
   d) Element properties and connectivity.
2. **Define Interfloor Types:** For each interfloor type define the following.
   a) Location of the origin of floor 2 relative to the origin of floor 1. This defines the interfloor height and the horizontal locations of the floors relative to each other. This information is needed to determine the dimensions and orientations of the interfloor elements.
   b) Coordinates of interfloor nodes (if any) relative to the origin of floor 1.
   c) Nodal masses.
   d) Whether the nodal displacements are unrestrained, restrained or spring supported.
   e) Element properties and connectivity.
3. **Specify Floor Instances:** For each floor instance define the following.
   a). Its floor type.
   b) The coordinates of its origin. This locates the floor in 3D space.
4. **Specify Interfloor Instances:** For each interfloor instance define the following.
   a) Its interfloor type.

b) Its floor 1 and floor 2 instances. This locates the interfloor in 3D space. It also provides redundant information, since floors 1 and 2 are also defined for the interfloor type. This information is checked for consistency.

5.3. ADVANTAGES OF USING FLOORS AND INTERFLOORS

There are two major advantages of using floor and interfloor types and instances. First, it is easier to prepare the input data. Second, floors and interfloors are convenient modules for setting up the data structures in the program.

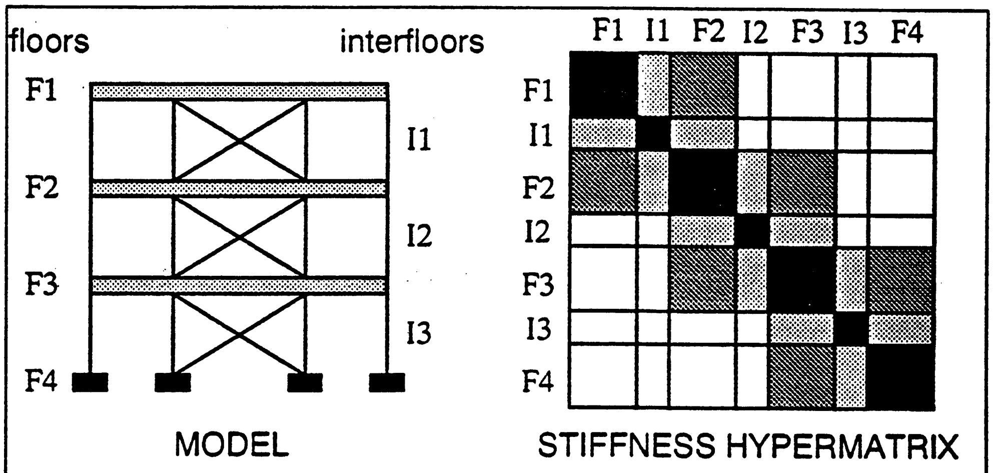
The input data is easier to prepare mainly because the nodes, elements, load patterns and results output specifications are input for floor and interfloor types, and the data does not have to be repeated for each instance. It is also simpler to define nodes and elements for a floor or interfloor type, since the coordinates and node numbering system are local to the type rather than global for the building as a whole. A useful feature of the program is that if two floor types have the same node locations but different member sizes, one type can be derived from the other without repeating all of the input data.

The program data structures are simplified for two main reasons. First, much of the data can be stored with the floor and interfloor types, so that the volume of data is reduced. The data storage for a type is very similar to that used for a complete structure in DRAIN-3DX. Second, each floor and interfloor instance can be treated as a subassembly, contributing a subassembly stiffness to the complete structure stiffness. This leads to a ***hypermatrix*** form for the structure stiffness [5], which is convenient logically and also allows very large structures to be analyzed with modest memory requirements. This aspect of the program is described in detail in the following sections.

A disadvantage of using floors and interfloors is that there is no direct concept of a ***frame***. For the design of a building it can be convenient to regard the structure as a series of multistory frames connected by floor diaphragms. A DRAIN-BUILDING model does not have this form. It is possible, however, to define a frame, by specifying that it consists of certain elements from the floor and interfloor instances. The results could then be organized frame-by-frame as a post-processing operation. This has not been done in the current version of the program, but is planned as a future extension.

5.4. HYPERMATRIX STRUCTURE OF THE STIFFNESS MATRIX

The stiffness matrix of a DRAIN-BUILDING model consists of symmetric row and column partitions, each corresponding to a floor or interfloor instance. Figure 5.3 shows a simple example.



■ Diagonal Floor Blocks

▪ Diagonal Interfloor Blocks

▨ Off-diagonal Blocks Coupling Two Floors

░ & ░ Off-diagonal Blocks Coupling a Floor and an Interfloo

FIG. 5.3. STRUCTURE OF THE STIFFNESS HYPERMATRIX

Since the matrix is symmetric, only the off-diagonal blocks above the diagonal plus the upper triangular portions of the diagonal blocks need to be stored.

The elements of a floor instance contribute stiffness coefficients to the corresponding diagonal floor block only. The elements of an interfloor instance can contribute stiffness coefficients to three diagonal and three off-diagonal blocks. The diagonal blocks correspond to the nodes in floor 1, floor 2 and the interfloor itself. The off-diagonal blocks couple the diagonal blocks. If there are no interfloor nodes the interfloor elements contribute stiffness coefficients to only two diagonal blocks and one off-diagonal block.

The blocks of the stiffness hypermatrix are stored on a direct-access file. Memory buffers are provided to store up to three diagonal and three off-diagonal blocks (less if there are no interfloor nodes). The buffer sizes are based on the largest block sizes.

During assembly of the element stiffnesses into the stiffness hypermatrix (TANK), there must generally be three diagonal blocks and three off-diagonal blocks in memory at any time. However, if the current interfloor has no interfloor nodes, only two diagonal blocks and one off-diagonal block are needed. Section 5.7 describes the assembly procedure in detail.

The base program subroutine HYPSOL has been developed to solve the hypermatrix equations. HYPSOL requires three blocks in memory at any time. The algorithm is described in Section 5.8.

The base program subroutine HYPMUL is used for forming hypermatrix-vector products. HYPMUL requires one block in memory at any time. The algorithm is described in Section 5.9.

5.5. ORDERING OF BLOCKS IN THE HYPERMATRIX

In all DRAIN programs, Crout's factorization is used to solve the equations. In DRAIN-2DX and 3DX the stiffness matrix is stored in compacted column form, omitting the terms above the first nonzero term in each column, and storing all terms below, up to and including the diagonal term. This type of storage can be termed ***compacted column, envelope,*** or ***skyline*** storage [6, 9]. In DRAIN-BUILDING, essentially the same storage scheme is used for storing the blocks of the stiffness hypermatrix. The term ***compacted***

column is used in this report for the conventional storage scheme used in DRAIN-2DX and 3DX. In this chapter, the terms ***envelope storage***, ***block envelope*** and ***envelope structure*** are used for describing the ***hypermatrix storage scheme*** of DRAIN-BUILDING.

The initially zero terms above the first nonzero term in any column remain zero during factorization, and hence never need to be stored. Initially zero terms below the first nonzero term may or may not "fill in" (i.e., become nonzero) during factorization, depending on the structure of the analysis model and the way in which the nodes are numbered. In DRAIN-2DX and 3DX it is assumed that all such terms will fill in, since the computational cost of keeping track of individual terms is not warranted. This is not the case with DRAIN-BUILDING, however, since hypermatrix blocks, not single scalars, are involved.

In DRAIN-BUILDING, the amount of block fill-in depends on the ordering of the floor and interfloor instances. Figure 5.4 shows a simple example with two different orderings and the corresponding fill-in blocks. Note that in ordering 2 some blocks within the block envelope do not suffer fill-in. Such blocks do not have to be stored or operated upon. However, initially zero blocks must be created for the filled in blocks. Ordering 1 would normally be used (possibly numbered upwards rather than downwards), and would be computationally more efficient than ordering 2 in most cases. However, DRAIN-BUILDING permits the floor instances to be input in any order.

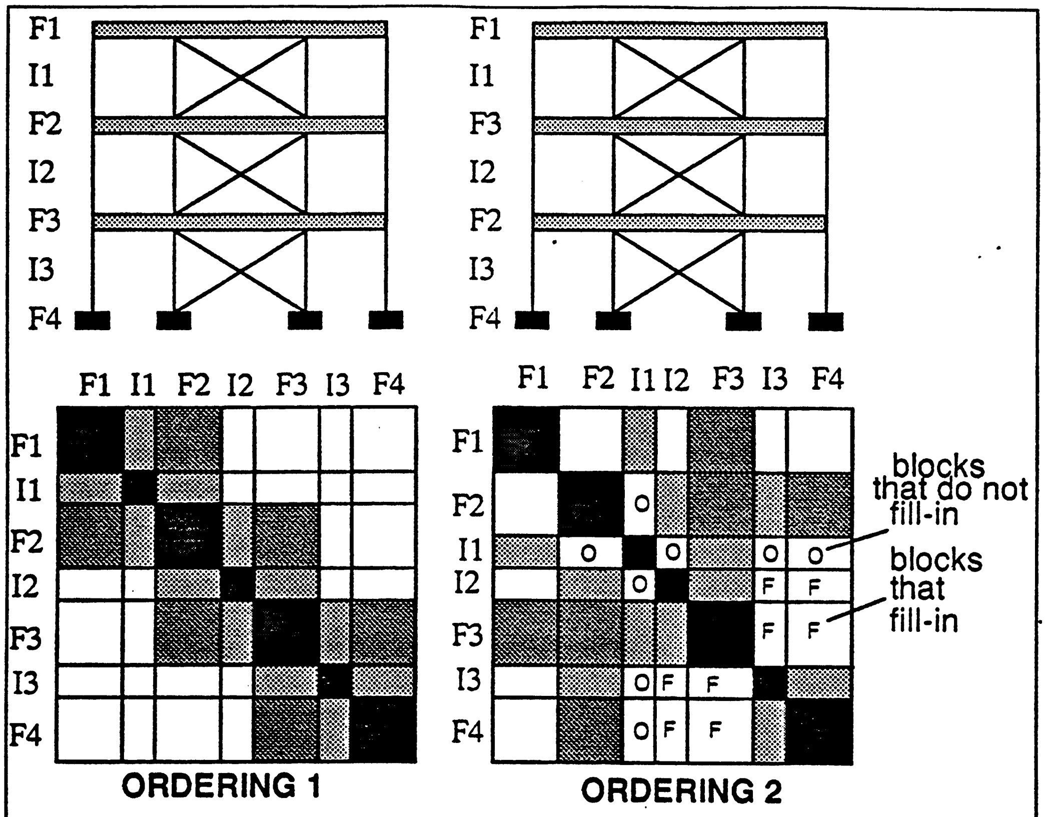


FIG. 5.4. BLOCK FILL-IN IN TWO ORDERINGS

The order of the stiffness blocks in the hypermatrix is determined by the input order of the floor instances, as follows.

1. The floor blocks are in the floor input sequence.
2. If an interfloor connects two floors, its block is placed immediately before the block for the second of the two floors. If more than two interfloors are to be placed before any floor, their blocks are placed in the input sequence of their first floors.
3. If an interfloor connects only one floor, its block is placed immediately before the block for that floor.

For a simple multistory building, if the order of input of the floor instances follows the natural top-down or bottom-up sequence, then a penta-diagonal hypermatrix results (i.e., Ordering 1 in Figure 5.4). If there are no interfloor nodes, a tri-diagonal hypermatrix results. In these cases there is no block fill-in.

Figure 5.5. shows two additional examples.

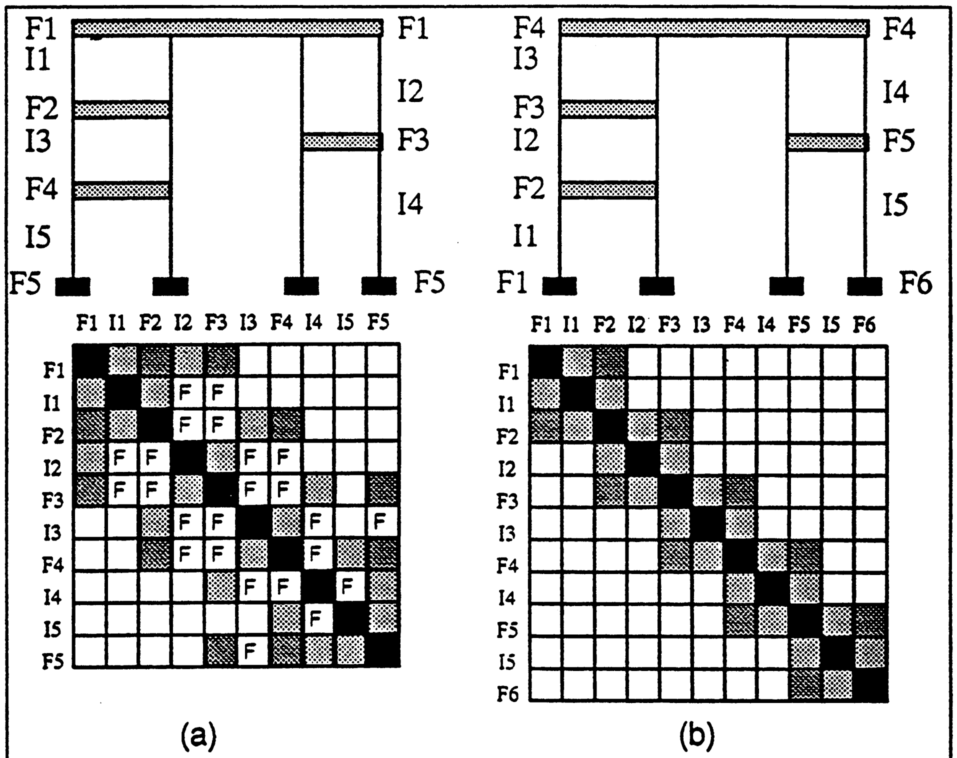

F1
I1
F2
I3
F4
I5
F5
F1
I2
F3
I4
F5
F1 I1 F2 I2 F3 I3 F4 I4 I5 F5
F
(a)
F4
I3
F3
I2
F2
I1
F1
F4
I4
F5
I5
F6
F1 I1 F2 I2 F3 I3 F4 I4 F5 I5 F6
(b)

FIG. 5.5. BLOCK FILL-IN IN TWO CASES

In Figure 5.5a, the ordering is from the top down and leads to a lot of block fill-in. In Figure 5.5b, the ground nodes are divided between floors F1 and F6, and the floor instances are specified clockwise around the frame. This results in a penta-diagonal hypermatrix.

In general the program will execute faster if (a) there is less fill-in and (b) the nonlinear elements are located at the end of the hypermatrix. The reason for (b) is that each time the stiffness changes, only the block columns starting from the first modified block column are refactorized (block columns before the first modified block column do not change). If it is known that only certain floors or interfloors become nonlinear, the program will probably execute most efficiently if these floors and interfloors are placed at the end of the hypermatrix. For example, if a simple building has base isolation but is otherwise linear, it is most efficient to order the floors top down. Conversely, if a nonlinear passive control system is placed in the top story, the most efficient ordering will be bottom up. If a yielding story or isolation system is near midheight, it may be most efficient to specify the middle floors last.

In the program, only stiffness blocks that are initially nonzero or suffer fill-in are stored and operated upon. Blocks are not stored if they are initially zero and do not suffer fill-in. Such blocks can be determined as follows.

1. Set array IND(N,N) to store "1" for each initially filled block, and "0" for each initially zero block, where N = number of diagonal blocks. All diagonal blocks and all coupling blocks are assumed to be filled.
2. For J = 2 to N and I = 1 to J-1, if IND(I,J) = 0, then
   a) For K = 1 to I-1 if IND(K,I) = 1 and IND(K,J) = 1, then reset IND(I,J) to 1.
3. If for any instance, there are no DOFs (because the instance does not have any nodes or all its nodes are restrained), then set the corresponding diagonal entry, IND(J,J), and off-diagonal entries, IND(I,J) and IND(J,I), for I $\neq$ J to 0.

The program follows essentially this procedure before setting up array IENV, which is described in Section 5.6.

5.6. HYPERMATRIX STORAGE SCHEME

5.6.1. Storage of Stiffness Blocks

The stiffness blocks of the hypermatrix are stored on a direct-access file. Two arrays, IEXNV and IENV, are used for addressing, as follows.

Array	Description
IEXNV(nfif+1) -	Location of the first nonzero block in IENV for each block column (i.e., pointer to the start of each block column in IENV). NFIF is the number of block rows (and block columns), which is equal to the number of floor and interfloor instances. For uniformity in indexing, IEXNV(nfif+1) is set equal to NZERO+1.
IENV(nzero)	Addresses (i.e., record number offsets on the direct access file - see Section 5.6.4) of blocks within the block envelope. For blocks that are initially zero and do not suffer fill-in, IENV is set to 0. NZERO is the number of blocks within the block envelope.

Figure 5.6 illustrates the scheme for ordering 2 shown in Figure 5.4.

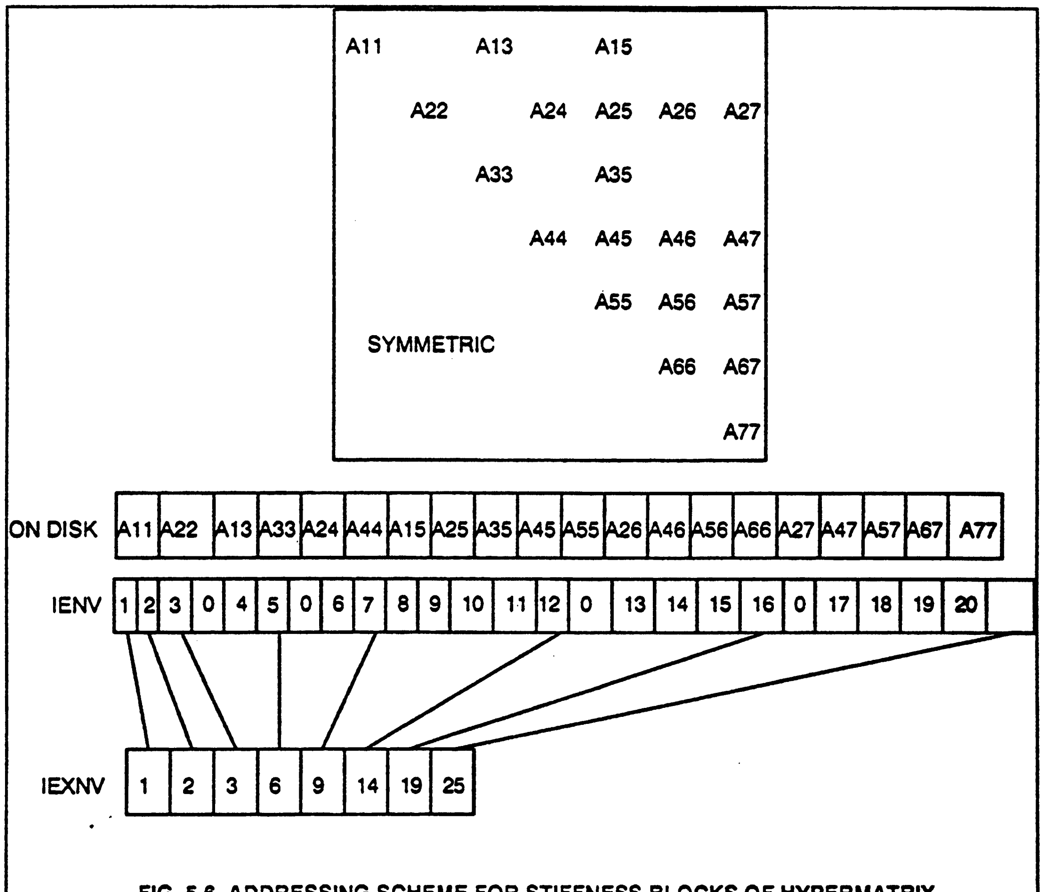


FIG. 5.6. ADDRESSING SCHEME FOR STIFFNESS BLOCKS OF HYPERMATRIX

With this scheme, some of the address computations are as follows.

1. The number of block rows, JLEN, within the block envelope, in block column J is given by

   JLEN = IEXNV(J+1) - IEXNV(J)

2. The block row number, IFST, of the first nonzero block in block column J is given by

   IFST = J + 1 - JLEN

3. The address of (I,J) block, IJLOC, in IENV is given by

   IJLOC = IEXNV(J+1) - (J-I) -1

   provided IFST $<$ I $\leq$ J.

4. The address of the diagonal (J,J) block, JJLOC, in IENV is given by

   JJLOC = IEXNV(J+1) - 1

5.6.2. Storage of a Diagonal Block

Envelope storage is used for each diagonal block. Two arrays, ISTIF and STIF are used for each block, as follows.

Array	**Description**
ISTIF(neq+1)	Location of first nonzero term in STIF for each column (i.e., pointer to the start of each column in STIF). NEQ is the number of columns (and rows) in the stiffness block. For uniformity in indexing, ISTIF(neq+1) is set equal to NSTF+1.
STIF(nstf)	Values within the envelope, from the first nonzero term to the diagonal term for each column. NSTF is the total number of terms within the envelope.

Figure 5.7 illustrates the scheme.

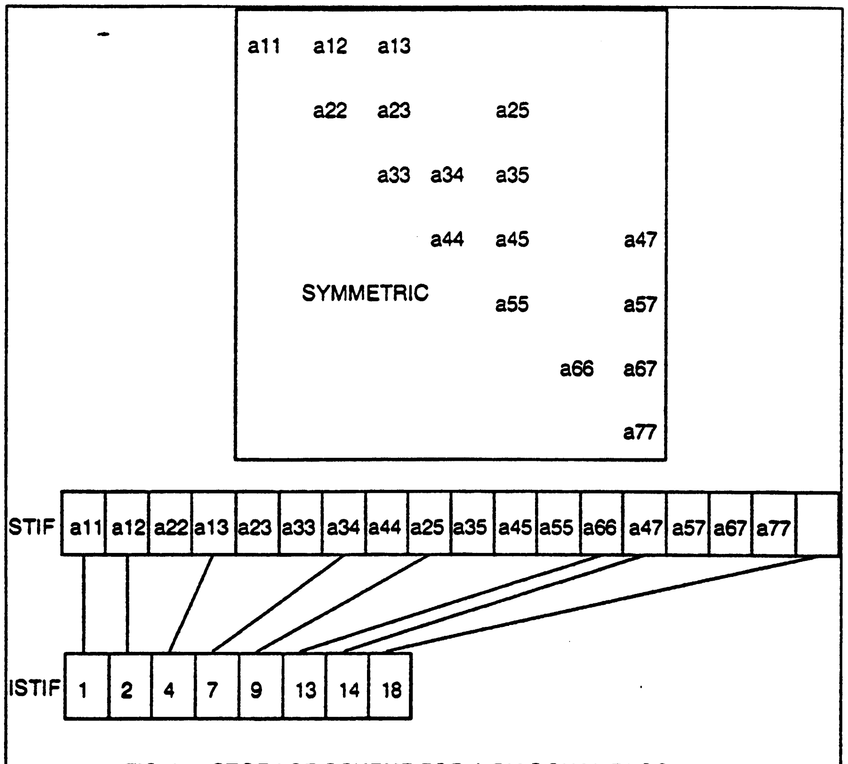


FIG. 5.7. STORAGE SCHEME FOR A DIAGONAL BLOCK

The matrix ISTIF is set as follows.

1. An array MSTIF of length NEQT, equal to the total number of structure DOFs, stores the row number of the first nonzero term for each column of the stiffness hypermatrix. This is the global row number.
2. For a diagonal stiffness block, let:
   a) NEQ be the number of rows (and columns) in the block.
   b) NFST+1 be the global number of the first row (and first column) of the block.
3. Set ISTIF(1) = 1.
4. For column J (for 1 ≤ J ≤ NEQ) of the stiffness block:
   a) The column starts at global row number, JSTRT, given by

      JSTRT = MAX (MSTIF(NFST+J),NFST+1).

   b) The column ends at global row number, JEND, given by

JEND = NFST + J.

c) The number of terms in the column, JLEN, is given by

JLEN = JEND - JSTRT + 1.

d) Set ISTIF(J+1) = ISTIF(J) + JLEN.

With this scheme, some of the address computations for a block are as follows.

1. The number of rows, JLEN, in column J is given by

   JLEN = ISTIF(J+1) - ISTIF(J)

2. The row number, IFST, of the first nonzero term in column J is given by

   IFST = J + 1 - JLEN

3. The address of term (I,J), IJLOC, in STIF is given by

   IJLOC = ISTIF(J+1) - (J-I) - 1

   provided IFST < I ≤ J

4. The address of the diagonal term (J,J), JJLOC, in STIF is given by

   JJLOC = ISTIF(J+1) - 1

The row or column numbers I, J and IFST referred to above are local to the block. The global row or column numbers are obtained by adding NFST to these numbers.

5.6.3. Storage of an Off-diagonal Block

Envelope storage is used for each off-diagonal block. Two arrays, ISTIF and STIF are used for each block, as follows.

Array	Description
ISTIF(neqc+1) -	Location of first nonzero term in STIF for each column (i.e., pointer to start of each column in STIF). NEQC is the number of columns in the stiffness block. For uniformity in indexing, ISTIF(neqc+1) is set equal to NSTF+1.
STIF(nstf)	Values within the envelope, from the first nonzero term to the last term for each column. NSTF is the total number of terms within the envelope.

Figure 5.8 illustrates the scheme.

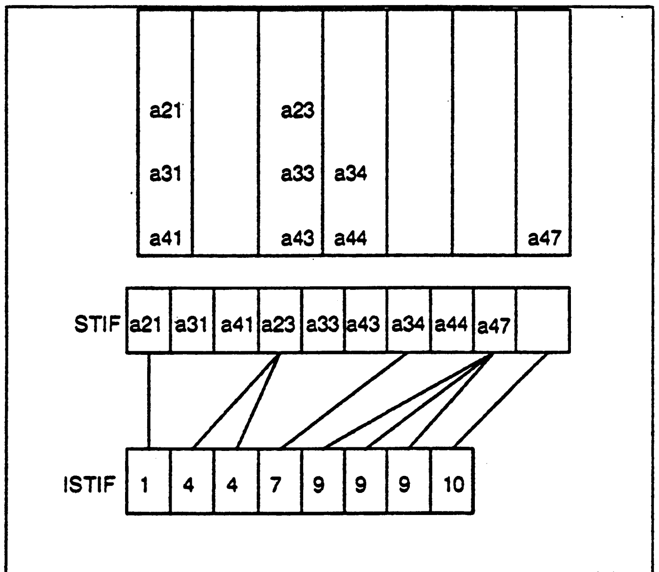


FIG. 5.8. STORAGE SCHEME FOR AN OFF-DIAGONAL BLOCK

The matrix ISTIF is set as follows.

1. An array MSTIF of length NEQT, equal to the total number of structure DOFs, stores the row number of the first nonzero term for each column of the stiffness hypermatrix.
2. For an off-diagonal stiffness block, let:
   a) NEQC be the number of columns in the block.
   b) NEQR be the number of rows in the block.
   c) NCFST+1 be the global number of the first column of the block.
   d) NRFST+1 be the global number of the first row of the block.
3. Set ISTIF(1)=1.

4. For column J (for $1 \le J \le NEQC$) of the stiffness block:
   a) The column starts at global row number, JSTRT, given by

      JSTRT = MAX (MSTIF(NRFST+J),NRFST+1)

      provided JSTRT ≤ NRFST + NEQR. Otherwise there are no terms in the column.
   b) The column ends at global row number, JEND, given by

      JEND = NRFST + NEQR.
   c) The number of terms in the column, JLEN, is given by

      JLEN = MAX (0, JEND - JSTRT + 1)
   d) Set ISTIF(J+1) = ISTIF(J) + JLEN

With this scheme, some of the address computations are as follows.

1. The number of rows, JLEN, within the envelope, in column J is given by

   JLEN = ISTIF(J+1) - ISTIF(J)

   JLEN can be zero for some columns.
2. The row number, IFST, of the first nonzero term in column J is given by

   IFST = NEQR + 1 - JLEN

   Note that if JLEN=0, then IFST > NEQR, and there are no nonzero terms in the column.
3. The address of term (I,J), IJLOC, in STIF is given by

   IJLOC = ISTIF(J) + I - IFST = ISTIF(J+1) - (NEQR-I) -1

   provided IFST < I ≤ J

The row or column numbers I, J and IFST referred to above are local to the block. The global row number is obtained by adding NRFST to the local row number. The global column number is obtained by adding NCFST to the local column number.

5.6.4. Hypermatrices in DRAIN-BUILDING

In DRAIN-BUILDING there are four hypermatrices, as follows.

1. TANK - unfactorized static tangent stiffness. TANK is updated whenever there is a stiffness change in any element.
2. EFFK - factorized effective stiffness. EFFK is updated whenever TANK is updated, or when there is change in the time step size.
3. BETAK - beta-K damping matrix (not factorized).
4. DTAN - backup of TANK, used for restoring TANK when the time step must be repeated in the variable time step scheme.

These four hypermatrices all have the same envelope structure, both at the global level (arrays IENV and IEXNV) and the local level (array ISTIF for each stiffness block).

All data blocks (STIF blocks for EFFK, TANK, BETAK and DTAN) are stored on a FORTRAN direct access file, unit NFSTFB. All index blocks (array ISTIF for each stiffness block) are stored on a second FORTRAN direct access file, NFSTFA. The array IENV directly gives the record numbers of the index blocks on NFSTFA. The record numbers of the data blocks on NFSTFB, for EFFK, TANK, BETAK and DTAN are obtained by adding the offsets NOFEK, NOFTK, NOFBK and NOFDK, respectively, to the values in IENV. These offsets are set as follows.

NOFEK = 0

NOFTK = NOFEK + NNZERO

NOFBK = NOFTK + NNZERO

NOFDK = NOFBK + NNZERO

where NNZERO is the number of nonzero blocks within the block envelope of the hypermatrix.

5.7. ASSEMBLY OF ELEMENT STIFFNESSES

For forming or updating the tangent stiffness hypermatrix (TANK), the elements are processed one floor or interfloor instance at a time. Before any instance is processed, subroutine GTBLKA writes any existing memory blocks to the direct-access file and computes the addresses of the blocks that may need to be updated while processing the current instance. If there is a stiffness change in an element, the affected stiffness blocks are copied into the memory buffers in subroutine GTBLKS and updated in subroutine ASSEMO. These blocks remain in memory until all the elements in the current instance have been processed. The blocks are then written to the direct-access file, in subroutine GTBLKA.

/STFBLK/ stores the record numbers of memory blocks.

The variables in /STFBLK/ are as follows.

Variable	**Description**
INMEM	Code for existence of blocks in memory, as follows. 0 : Blocks are not in memory. Bring them into memory if and when the stiffness of an element changes in the current instance, and reset the code to 1. 1 : Blocks are in memory. Write them to the NFSTFB file after processing elements of the current instance. Before processing the elements of any instance, the record numbers of the blocks required for the current instance are computed and INMEM is set to 0.

LSTBLK(6)	Record numbers of stiffness blocks that may change due to changes in element stiffnesses for elements of the current instance. The blocks are in the following order. For a floor: 5 : Diagonal block for the floor. 1-4 and 6 : 0. For an interfloor with interfloor nodes: 1 : Diagonal floor 1 block. 2 : Off-diagonal block coupling floor 1 and floor 2. 3 : Diagonal floor 2 block. 4 : Off-diagonal block coupling floor 1 and interfloor. 5 : Diagonal block for the interfloor. 6 : Off-diagonal block coupling floor 2 and interfloor. For an interfloor with no interfloor nodes: 1 : Diagonal floor 1 block. 2 : Off-diagonal block coupling floor 1 and floor 2. 3 : Diagonal floor 2 block. 4-6 : 0.

The stiffness blocks are stored in memory buffers STIF1 through STIF6 and ISTIF1 through ISTIF6 (see KSTIF and KISTIF in /STOR7/ in Chapter 6).

The following tasks are done in GTBLKA.

1. If INMEM=1, write the memory blocks STIF1-STIF6 to the direct access file, NFSTFB. The record numbers of the memory blocks are obtained form LSTBLK in /STFBLK/.
2. Initialize array LSTBLK to 0.
3. Set LSTBLK for the current instance.

The following tasks are done in ASSEMO.

1. If INMEM=0, call GTBLKS to read the stiffness blocks into memory.
2. Assemble the stiffness coefficients into the affected memory blocks.

The following task are done in GTBLKS.

1. Copy the required blocks STIF1-STIF6 and ISTIF1-ISTIF6 (see KSTIF and KISTIF in /STOR7/) from units NFSTFB and NFSTFA, respectively. The record numbers are obtained from LSTBLK in /STFBLK/.
2. Set INMEM=1.

The element "location matrix" (LM array) determines the terms to be modified. The value in LM for any DOF is as follows.

a) n : For an independent (unslaved) DOF , n=IFIF + LEQ*3

   where,

   IFIF = 0 if the DOF belongs to the current floor or interfloor instance,

   1 if the DOF belongs to the floor 1 of the interfloor, and

   2 if the DOF belongs to the floor 2 of the interfloor.

   LEQ = local equation number for the DOF

   Given n, IFIF and LEQ can be recovered as follows:

   IFIF = MOD(n,3).

   LEQ = n/3.

b) n : For a Z rotation slaved to the floor master node, n=IFIF + LEQ*3

   where,

   IFIF = 0 if the DOF belongs to the current floor instance,

   1 if the DOF belongs to the floor 1 of the interfloor, and

   2 if the DOF belongs to the floor 2 of the interfloor.

   LEQ = local equation number for the Z rotation of the floor master node.

   Given n, IFIF and LEQ can be recovered as for (a).

c) -n : For a X or Y translation slaved to the floor master node,

   n = IFIF + NDISP*3 + NLOC*9

   where,

   IFIF = 0, 1 or 2 as for (b).

NDISP = 1 for X translation, 2 for Y translation.

al X or 2 for translational Y DOF = MOD(n,9)/3.

NLOC = sequence number of the slaved node.

Given n, IFIF, NDISP and NLOC can be recovered as follows:

IFIF = MOD(n,3).

NDISP = MOD(n,9)/3.

NLOC = n/9.

Let for (I,J) term of the element stiffness:

1. If IFIF(I)=0 and IFIF(J)=0, then STIF5 is updated.
2. If IFIF(I)=0 and IFIF(J)=1 or vice versa, then STIF4 is updated.
3. If IFIF(I)=0 and IFIF(J)=2 or vice versa, then STIF6 is updated.
4. If IFIF(I)=1 and IFIF(J)=1, then STIF1 is updated.
5. If IFIF(I)=1 and IFIF(J)=2 or vice versa, then STIF2 is updated.
6. If IFIF(I)=2 and IFIF(J)=2 or vice versa, then STIF3 is updated.

The terms in STIF1-STIF6 which are updated depend on:

a) LEQ(I) if element DOF, I, is independent or slaved Z rotation.

b) NDISP(I) and NLOC(I) if element DOF, I, is slaved X or Y translation.

c) LEQ(J) if element DOF, J, is independent or slaved Z rotation.

d) NDISP(J) and NLOC(J) if element DOF, J, is slaved X or Y translation.

For cases (b) and (d) the array FMASTC in /CNTFIF/ gives the coordinates of the master node; the array COORD (see KCOORD in /STOR3/) for the floor gives the coordinates of the slaved node; and the array ID (see KID in /STOR3/) for the floor gives the equation numbers of the master node. See subroutine ASSEMO for implementation details.

5.8. HYPERMATRIX EQUATION SOLVER -- HYPSOL

5.8.1. Introduction

The set of equations to be solved is

$$\underline{A}\underline{x} = \underline{b} \tag{5.1a}$$

in which $\underline{A}$ is the stiffness hypermatrix, $\underline{b}$ is the load vector increment and $\underline{x}$ is the displacement increment to be determined. Equation 5.1 can be expanded to show the row and column partitions, as follows.

$$\begin{bmatrix} \underline{A}_{11} & \underline{A}_{12} & \cdots & \underline{A}_{1n} \\ \underline{A}_{21} & \underline{A}_{22} & \cdots & \underline{A}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \underline{A}_{n1} & \underline{A}_{n2} & \cdots & \underline{A}_{nn} \end{bmatrix} \begin{Bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_n \end{Bmatrix} = \begin{Bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_n \end{Bmatrix} \tag{5.1b}$$

Hypermatrix $\underline{A}$ is symmetric and $\underline{A}_{ij} = \underline{A}_{ji}^T$, where $\underline{A}_{ij}$ is the (i, j) block of $\underline{A}$.

To solve these equations, $\underline{A}$ is factored as follows.

$$\underline{A} = \underline{U}^T \underline{D}\underline{U} \tag{5.2}$$

in which $\underline{U}$ is a unit upper triangular matrix and $\underline{D}$ is a diagonal matrix.

The hypermatrices $\underline{A}$ and $\underline{U}$ have the same envelope structure. The stiffness blocks of $\underline{A}$ are replaced by those of $\underline{U}$ as the factorization proceeds. The diagonal terms in diagonal blocks, $\underline{U}_{jj}$, are known to be unity, and are used to store the diagonal terms of $\underline{D}_{jj}$.

Given the factorization (Equation 5.2), the equations are solved in three steps as follows.

$$\text{Solve } \underline{U}^T \underline{z} = \underline{b} \text{ for } \underline{z} \tag{5.3}$$

$$\text{Solve } \underline{D}\underline{y} = \underline{z} \text{ for } \underline{y} \tag{5.4}$$

$$\text{Solve } \underline{U}\underline{x} = \underline{y} \text{ for } \underline{x} \tag{5.5}$$

Equation 5.3, is a lower triangular system of equations. This is the forward substitution step.

Equation 5.4, is a diagonal system of equations. This step involves the division of each term in $\underline{z}$ by the corresponding diagonal term in $\underline{D}$.

Equation 5.5, is an upper triangular system of equations. This is the backward substitution step.

5.8.2. Factorization

In Equation 5.2,

$$\underline{D}_{ij} = \underline{0} \quad \text{if} \quad i \neq j \tag{5.6}$$

$$\underline{U}_{ij} = \underline{0} \quad \text{if} \quad i \neq L_i \quad \text{or if} \quad i > j \tag{5.7}$$

where, L_j is the block row number of the first nonzero block in block column j.

The block $\underline{A}_{ij}$ is, thus given by

$$\underline{A}_{ij} = \sum_{k=\max(L_i,L_j)}^{\min(i,j)} \left(\underline{U}_{ki}^T \underline{D}_{kk} \underline{U}_{kj} \right) \tag{5.8}$$

The factorization of $\underline{A}$ proceeds block column by block column. At the time of factorization of block column j, all blocks $\underline{A}_{ki}$ for $i < j$ and $k < i$ have been replaced by $\underline{U}_{ki}$. Also, all diagonal blocks $\underline{A}_{ii}$ for $i < j$ have been replaced by $\underline{U}_{ii}$ and $\underline{D}_{ii}$.

Equation 5.8, for $i < j$ is as follows.

$$\begin{aligned} \underline{A}_{ij} &= \sum_{k=\max(L_i,L_j)}^{i-1} \left(\underline{U}_{ki}^T \underline{D}_{kk} \underline{U}_{kj} \right) + \underline{U}_{ii}^T \underline{D}_{ii} \underline{U}_{ij} \\ \text{or} \quad \underline{U}_{ii}^T \left(\underline{D}_{ii} \underline{U}_{ij} \right) &= \underline{A}_{ij} - \sum_{k=\max(L_i,L_j)}^{i-1} \left\{ \underline{U}_{ki}^T \left(\underline{D}_{kk} \underline{U}_{kj} \right) \right\} \\ &= \underline{A}_{ij}^{*} \end{aligned} \tag{5.9}$$

In Equation 5.9, $\underline{U}_{ii}$ and $\underline{U}_{ki}$ have already been formed and are stored in place of $\underline{A}_{ii}$ and $\underline{A}_{ki}$, respectively. The block $\underline{A}_{kj}$ actually stores the product $\underline{D}_{kk}\underline{U}_{kj}$ that occurs on the R.H.S. in Equation 5.9. Each step, k, in the formation of $\underline{A}_{ij}^{*}$ constitutes a non-symmetric update of the off-diagonal block $\underline{A}_{ij}$. Once $\underline{A}_{ij}^{*}$ is formed, then Equation 5.9 is a lower triangular system of equations that is solved to replace $\underline{A}_{ij}$ by the product $\underline{D}_{ii}\underline{U}_{ij}$. Equation 5.9 is used to modify blocks $\underline{A}_{ij}$ to store the product $\underline{D}_{ii}\underline{U}_{ij}$ for $i = L_j$ to $j-1$.

Equation 5.8, for $i = j$ is as follows.

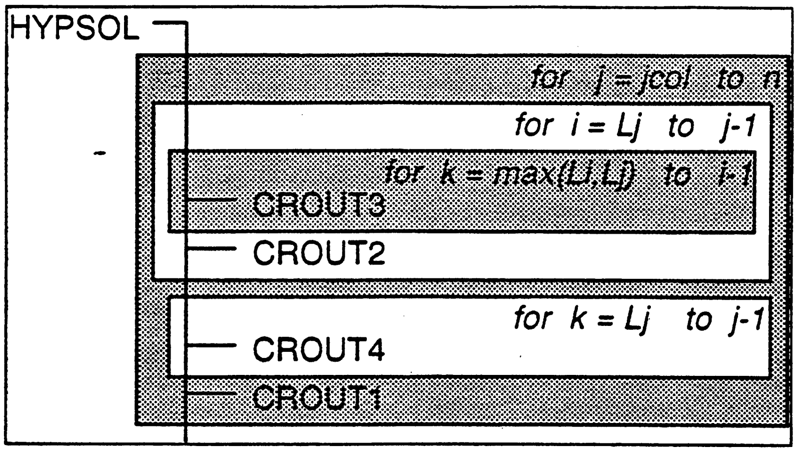
$$\begin{aligned} \underline{A}_{jj} &= \sum_{k=L_j}^{j-1}\left(\underline{U}_{kj}^{T}\underline{D}_{kk}\underline{U}_{kj}\right)+\underline{U}_{jj}^{T}\underline{D}_{jj}\underline{U}_{jj} \\ \text{or}\quad \underline{U}_{jj}^{T}\left(\underline{D}_{jj}\underline{U}_{jj}\right) &= \underline{A}_{jj} - \sum_{k=L_j}^{j-1}\left\{\underline{U}_{kj}^{T}\left(\underline{D}_{kk}\underline{U}_{kj}\right)\right\} \\ &= \underline{A}_{jj}^{*} \end{aligned} \tag{5.10}$$

In Equation 5.10, the product $\underline{D}_{kk}\underline{U}_{kj}$ is actually stored in $\underline{A}_{kj}$. Each step, k, in the formation of $\underline{A}_{jj}^{*}$, constitutes a symmetric update of the diagonal block $\underline{A}_{jj}$; and as $\underline{A}_{jj}^{*}$ is updated the block $\underline{A}_{kj}$ is modified to store $\underline{U}_{kj}$ instead of the product $\underline{D}_{kk}\underline{U}_{kj}$. Once $\underline{A}_{jj}^{*}$ is formed, then by Crout's factorization of the block, it is replaced by $\underline{U}_{jj}$ and $\underline{D}_{jj}$.

Equations 5.9 and 5.10 are thus used for factorization of each block column j for j = 1 to n of hypermatrix $\underline{A}$.

In HYPSOL, the diagonal of hypermatrix $\underline{A}$, is stored in an array, DIAG, which is always in memory. This reduces the amount of disk input-output in Equation 5.10, for the steps in which blocks $\underline{A}_{kj}$ are modified to store $\underline{U}_{kj}$ instead of the product $\underline{D}_{kk}\underline{U}_{kj}$.

The flow chart for the factorization phase is as follows.



In this chart JCOL is the number of the first block column that has changed. The block columns from j =1 to JCOL-1, are already factorized and do not change.

The following main tasks are performed for each block column j

1. For $i = L_j$ to j-1
   a) In HYPSOL, bring block $\underline{A}_{ij}$ into memory.
   b) For $k = \max(L_i, L_j)$ to i-1
      (i) In HYPSOL, bring blocks $\underline{A}_{ki}$ and $\underline{A}_{kj}$ into memory.
      (ii) In CROUT3, perform a non-symmetric update of $\underline{A}_{ij}$ as follows.
      $$\underline{A}_{ij}^{(k)} \Leftarrow \underline{A}_{ij}^{(k-1)} - \underline{U}_{ki}^{T}\left(\underline{D}_{kk}\underline{U}_{kj}\right)$$
      In CROUT3, on entry the blocks $\underline{A}_{ij}$, $\underline{A}_{ki}$ and $\underline{A}_{kj}$ store $\underline{A}_{ij}^{(k-1)}$, $\underline{U}_{ki}$ and $\underline{D}_{kk}\underline{U}_{kj}$, respectively. On exit the block $\underline{A}_{ij}$ stores $\underline{A}_{ij}^{(k)}$.
   c) In HYPSOL, bring block $\underline{A}_{ii}$ into memory.
   d) In CROUT2, solve the lower triangular system of equations as follows.
   $$\underline{U}_{ii}^{T}\left(\underline{D}_{ii}\underline{U}_{ij}\right) = \underline{A}_{ij}^{*}$$
   In CROUT2, on entry the blocks $\underline{A}_{ii}$ and $\underline{A}_{ij}$ store $\underline{U}_{ii}$ and $\underline{A}_{ij}^{*}$, respectively. On exit the block $\underline{A}_{ij}$ stores the solution $\underline{D}_{ii}\underline{U}_{ij}$.
   e) In HYPSOL, write block $\underline{A}_{ij}$ to disk file.
2. In HYPSOL, bring block $\underline{A}_{jj}$ into memory.

3. For $k = L_j$ to j-1
   a) In HYPSOL, bring block $\underline{A}_{kj}$ into memory.
   b) In CROUT4, perform a symmetric update of $\underline{A}_{jj}$ as follows.

   $$\underline{A}_{jj}^{(k)} \Leftarrow \underline{A}_{jj}^{(k-1)} - \underline{U}_{kj}^{T}\left(\underline{D}_{kk}\underline{U}_{kj}\right)$$

   In CROUT4, the input consists of blocks $\underline{A}_{jj}$ and $\underline{A}_{kj}$ and vectors $\underline{D}_{jj}$ and $\underline{D}_{kk}$. On entry, $\underline{A}_{jj}$ and $\underline{D}_{jj}$ store the off-diagonal and diagonal values, respectively, of $\underline{A}_{jj}^{(k-1)}$, and $\underline{A}_{kj}$ stores the product $\underline{D}_{kk}\underline{U}_{kj}$. On exit, $\underline{A}_{jj}$ and $\underline{D}_{jj}$ store the off-diagonal and diagonal values, respectively, of $\underline{A}_{jj}^{(k)}$, and $\underline{A}_{kj}$ stores $\underline{U}_{kj}$.
   c) In HYPSOL, write block $\underline{A}_{kj}$ to disk file.
   d) In CROUT1, perform the Crout's factorization of $\underline{A}_{jj}$ as follows.

   $$\underline{U}_{jj}^{T}\left(\underline{D}_{jj}\underline{U}_{jj}\right) = \underline{A}_{jj}^{*}$$

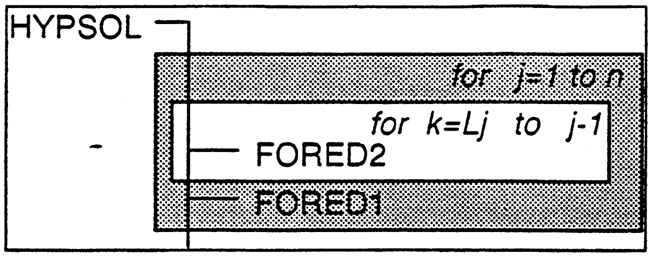
   In CROUT1, input consists of block $\underline{A}_{jj}$ and vector $\underline{D}_{jj}$. On entry these store the off-diagonal and diagonal values, respectively, of $\underline{A}_{jj}^{*}$. On exit $\underline{A}_{jj}$ stores $\underline{U}_{jj}$ and $\underline{D}_{jj}$ stores $\underline{D}_{jj}$.
   e) In HYPSOL, write block $\underline{A}_{jj}$ to disk file.

5.8.3. Forward and Backward Substitution of Load Vector

The j th block equation in Equation 5.3, is as follows.

$$\sum_{k=L_j}^{j} \underline{U}_{kj}^{T}\underline{z}_k = \underline{b}_j$$
$$\text{or}\quad \underline{U}_{jj}^{T}\underline{z}_j = \underline{b}_j - \sum_{k=L_j}^{j-1} \underline{U}_{kj}^{T}\underline{z}_k \qquad (5.11)$$

The flow chart for the forward substitution phase is as follows.



The following main tasks are performed.

1. In HYPSOL, bring block $\underline{U}_{kj}$ into memory; and in FORED2, update the load vector as follows.

   $\underline{b}_j \Leftarrow \underline{b}_j - \underline{U}_{kj}^T \underline{z}_k$

2. In FORED1, solve the following lower triangular system of equations.

   $\underline{U}_{jj}^T \underline{z}_j = \underline{b}_j$

Equation 5.4, is trivial to solve as both $\underline{z}$ and $\underline{D}$ are in memory, and $\underline{D}$ is diagonal. Each term of $\underline{z}$ is divided by the corresponding diagonal term of $\underline{D}$ to get $\underline{y}$.

The j th block equation in Equation 5.5, is as follows.

$$\begin{gathered} \sum_{k=j}^{n} \underline{U}_{jk} \underline{x}_k = \underline{y}_j \\ \text{or} \quad \underline{U}_{jj} \underline{x}_j = \underline{y}_j - \sum_{k=j+1}^{n} \underline{U}_{jk} \underline{x}_k \end{gathered} \tag{5.12}$$

Equation 5.12, can be used to solve for $\underline{x}_j$ for $j = n$ down to 1. However, a straight forward implementation of Equation 5.12 results in accessing the blocks $\underline{U}_{jk}$ in row order. In HYPSOL, $\underline{x}_J$ is eliminated from the previous block equations, immediately after it is calculated. This results in a group of $\underline{U}_{IJ}$ blocks to be accessed in column order.

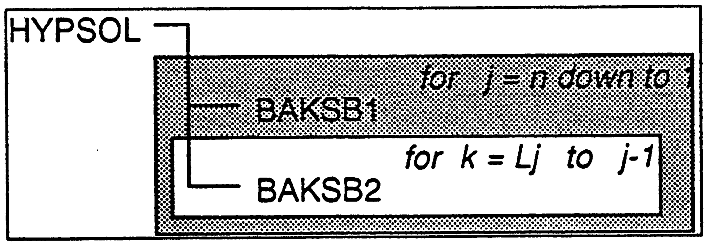
The main steps in back-substitution are as follows.

1. For $j = n$ down to 1

   a) Solve the upper triangular system, $\underline{U}_{jj} \underline{x}_j = \underline{y}_j$, for $\underline{x}_j$.

b) For $k = L_j$ to j-1, update $\underline{y}_k \Leftarrow \underline{y}_k - \underline{U}_{kj}\underline{x}_j$

The flow chart for the back-substitution phase is as follows.



The following main tasks are performed.

1. In HYPSOL, bring block $\underline{U}_{jj}$ into memory and solve the following upper triangular system of equations.

   $\underline{U}_{jj}\underline{x}_j = \underline{y}_j$, for $\underline{x}_j$.

2. In HYPSOL, bring block $\underline{U}_{kj}$ in memory; and in BAKSB2, update the load vector, $\underline{y}_k$, as follows.

   $\underline{y}_k \Leftarrow \underline{y}_k - \underline{U}_{kj}\underline{x}_j$

5.9. HYPERMATRIX-VECTOR PRODUCT -- HYPMUL

The system is represented as follows.

$$\underline{x} = \underline{A}\underline{b} \tag{5.13}$$

in which $\underline{A}$ is the hypermatrix, $\underline{b}$ is a given vector and $\underline{x}$ is a vector to be determined.

The term $\underline{x}_j$ is as follows.

$$\underline{x}_j = \sum_{k=1}^{n} \underline{A}_{jk}\underline{b}_k \tag{5.14}$$

For a symmetric hypermatrix Equation 5.14 can be written as follows.

$$\underline{x}_j = \sum_{k=1}^{j} \underline{A}_{kj}^T\underline{b}_k + \sum_{k=j+1}^{n} \underline{A}_{jk}\underline{b}_k \tag{5.15}$$

Equation 5.15, shows that an off-diagonal block $\underline{A}_{ij}$ contributes to $\underline{x}_i$ and $\underline{x}_j$, as follows.

$$\underline{x}_i \Leftarrow \underline{x}_i + \underline{A}_{ij}\underline{b}_j \tag{5.16}$$

$$\underline{x}_j \Leftarrow \underline{x}_j + \underline{A}_{ij}^T\underline{b}_i \tag{5.17}$$

Figure 5.9 shows the contributions graphically.

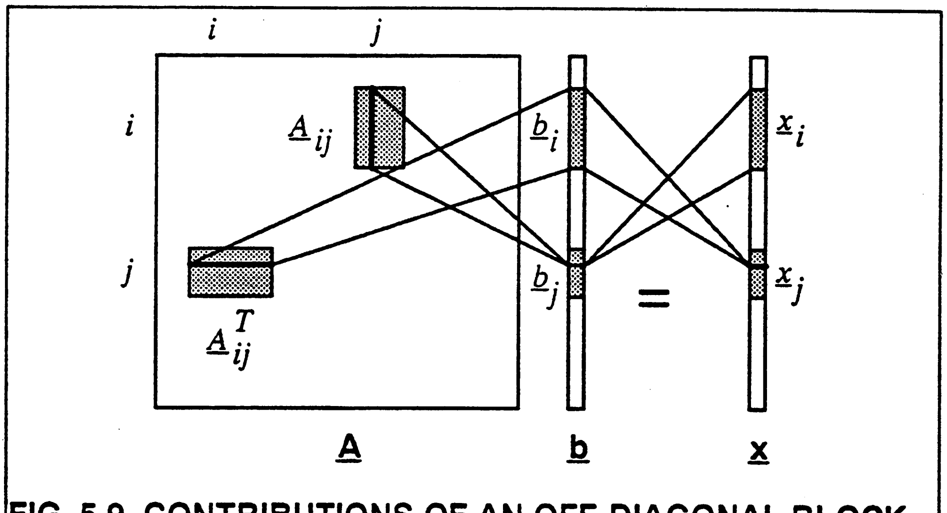


FIG. 5.9. CONTRIBUTIONS OF AN OFF-DIAGONAL BLOCK

Each diagonal block $\underline{A}_{jj}$ contributes to $\underline{x}_j$, as follows.

$$\underline{x}_j \Leftarrow \underline{x}_j + \underline{A}_{jj}\underline{b}_j \tag{5.18}$$

Figure 5.10 shows the contributions graphically.

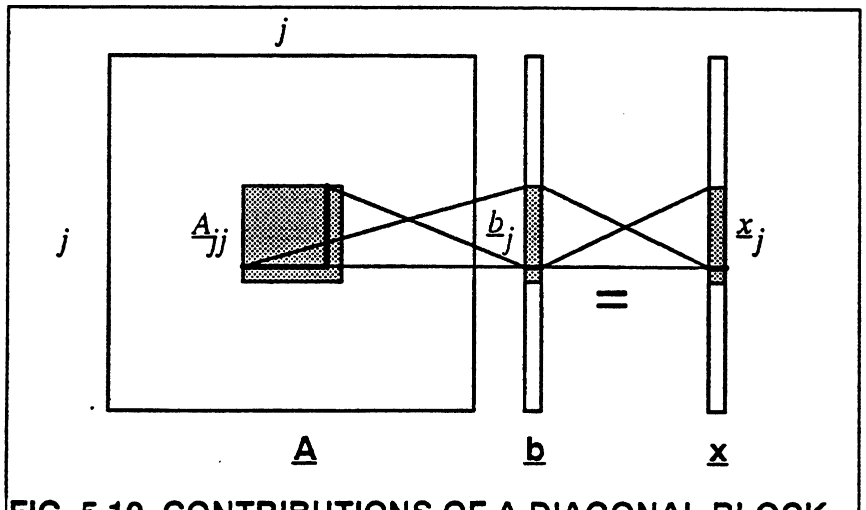


FIG. 5.10. CONTRIBUTIONS OF A DIAGONAL BLOCK

The following are the main steps in HYPMUL.

```
1. For $j = 1$ to $n$
   a) For $i = L_j$ to $j$-1
      (i) Bring $\underline{A}_{ij}$ in memory.
      (ii) Update $\underline{x}_i \Leftarrow \underline{x}_i + \underline{A}_{ij}\underline{b}_j$
      (iii) Update $\underline{x}_j \Leftarrow \underline{x}_j + \underline{A}_{ij}^T\underline{b}_i$
   b) Bring $\underline{A}_{jj}$ in memory.
   c) Update $\underline{x}_j \Leftarrow \underline{x}_j + \underline{A}_{jj}\underline{b}_j$
```

5.10. TIMES WHEN HYPERMATRICES ARE NOT USED

DRAIN-BUILDING does not always store the stiffness matrix in hypermatrix form. If the problem is small, or if blank common is large, the stiffness and beta-K damping matrices are stored in conventional compacted column form, as in DRAIN-2DX and 3DX (see INCOR, NOFEK, NOFBK and NOFDK in /CONTR/ and /STOR8/).

Memory buffers are needed for at least two compacted column matrices (i.e., INCOR=1 or 2). The first buffer always stores TANK. The second buffer usually stores EFFK but may also store BETAK if velocities are corrected to improve energy balance (see KEQBC in /INDIC/ and subroutine CORECT). A direct access file, unit NFSTFB, has three records for EFFK, BETAK and DTAN. In the state determination phase, TANK is updated if any element stiffness has changed. To form EFFK for dynamic analysis, BETAK is first read into the second buffer and then EFFK is formed in this buffer.

Usually, EFFK has to be refactorized starting from the first modified column, JCOL (see /EQNS/), unless the time step changes, in which case it has to be refactorized completely (i.e., JCOL=1). However, with only two memory buffers, BETAK is read into the buffer storing EFFK, and the factorized EFFK is overwritten. Therefore, with only two memory buffers, EFFK has to be refactorized completely at each time step that has a stiffness change. This is a disadvantage of having only two memory buffers, but in most cases the execution would still be faster than if hypermatrices are used.

If there is enough memory for three compacted column matrices (i.e., INCOR=3), then TANK, EFFK and BETAK are stored in memory, and file NFSTFB has only one record for DTAN. If the variable time step scheme is used, then TANK is backed up on NFSTFB just before the first substep is taken, and recovered from NFSTFB if the step must be repeated.

If there is enough memory (i.e., INCOR=4 or 5), then TANK, EFFK, BETAK and DTAN are all stored in memory.

Subroutine ASSEMI (identical to subroutine ASSEM of DRAIN-2DX and 3DX) is used for assembly of element stiffnesses into TANK. Subroutine UPDTSI (similar to subroutine UPDATS of DRAIN-2DX and 3DX) is used for updating EFFK for static analysis. Subroutine UPDTDI (similar to subroutine UPDATE of DRAIN-2DX and 3DX) is used for updating EFFK for dynamic analysis.

6. DATA MANAGEMENT -- DRAIN-BUILDING

6.1. INTRODUCTION

Like DRAIN-2DX and 3DX, DRAIN-BUILDING also stores data in memory and on a number of permanent and temporary (scratch) files.

Section 6.2 describes the labeled common blocks that are new or different from those in program DRAIN-3DX. Section 6.3 describes the scheme for storing arrays in blank common. Sections 6.4 and 6.5 give an overview of disk files. The details of the permanent files are presented in Chapter 8.

6.2. LABELED COMMONS

6.2.1. File PARA.H

The length of arrays in /CNTFIF/, /DISVEL/, /ENRGY/, /GENINF/, /RHIST/, /STOR3/, /STOR5/ and /TITFIF/ blocks is controlled by FORTRAN parameters contained in file PARA.H. The file, PARA.H, is included in subroutines that use these common blocks.

The parameters in PARA.H are as follows.

```
PARAMETER (MFLRTP=20)
PARAMETER (MIFRTP=20)
PARAMETER (MFIFTP=40)
PARAMETER (MFLRP=20)
PARAMETER (MIFRP=20)
PARAMETER (MFIFP=40)
PARAMETER (MAXGRP=20)
PARAMETER (MXDOFP=30)
```

These parameters are described as follows.

Parameter	Description
MFLRTP	Maximum number of floor types. The actual number of floor types is NFLRT in /FLRIFR/.
MIFRTP	Maximum number of interfloor types. The actual number of interfloor types is NIFRT in /FLRIFR/
MFIFTP	Maximum number of floor and interfloor types. The actual number of floor and interfoor types is NFIFT in /FLRIFR/.
MFLRP	Maximum number of floor instances. The actual number of floor instances is NFLR in /FLRIFR/.
MIFRP	Maximum number of interfloor instances. The actual number of interfloor instances is NIFR in /FLRIFR/.
MFIFP	Maximum number of floor and interfloor instances. The actual number of floor and interfloor instances is NFIF in /FLRIFR/.
MAXGRP	Maximum number of element groups in any floor or interfloor type. The actual number of element groups are in array NELGRF in /CNTFIF/.
MXDOFP	Maximum number of DOFs for any element. The actual number of element DOFs are in array NEDOF in /GENINF/.

The default values in PARA.H should be adequate for most problems. If any of these values is exceeded, then the base program writes error messages in .ECH file before quitting execution. The values in PARA.H must be suitably changed and the program recompiled to run the problem.

In DRAIN-BUILDING, floors and interfloors are numbered in the order of input as follows.

1. Floor types are numbered from 1 to NFLRT, in arrays having data for floor types only.

2. Interfloor types are numbered from 1 to NIFRT, in arrays having data for interfloor types only.
3. Floor and interfloor types are numbered from 1 to NFIFT (=NFLRT+NIFRT), in arrays having data for both floor and interfloor types. The floor types have numbers from 1 to NFLRT and the interfloor types have numbers from NFLRT+1 to NFIFT in these arrays.
4. Floor instances are numbered from 1 to NFLR, in arrays having data for floor instances only.
5. Interfloor instances are numbered from 1 to NIFR, in arrays having data for interfloor instances only.
6. Floor and interfloor instances are numbered from 1 to NFIF (=NFLR+NIFR), in arrays having data for both floor and interfloor instances. The floor instances have numbers from 1 to NFLR and the interfloor instances have numbers from NFLR+1 to NFIF in these arrays.

6.2.2. CNTFIF Common

/CNTFIF/ stores control information for floors and interfloors.

/CNTFIF/ is declared as follows.

```
 COMMON /CNTFIF/ alphaf(4,MFIFP), flrtdc(6,MFLRTP), fmastc(2,MFLRP),
1                fmastm(2,MFLRP), xyzifr(3,MIFRTP), xyzfif(3,MFIFP),
2                jirfd(MFLRTP), jifrgt(2,MIFRTP), jflr(MFLRP),
3                jifr(2,MIFRP), jfift(MFIFP), jfif(2,MFIFP),
4                nnodsf(MFIFTP), ntndsf(MFIFTP), ndspf(MFIFTP),
5                nelgf(MFIFTP), nelgrf(MFIFTP), nelttf(MFIFTP),
6                nssecf(MIFRTP), mxcutf(MIFRTP), ntrnsf(MIFRTP),
7                mxtdff(MIFRTP), nrdsf(MFIFTP), neqf(MFIFTP),
8                neqof(MFIFP), niadf(MFIFTP), nblokf(MFIFP)
```

The variables are as follows.

Variable	Description
ALPHAF	Mass scale factors for each floor and interfloor instance. 1 : Dead load mass scale factor. 2 : Live load mass scale factor. 3 : Alpha-M damping factor. 4 : Scale factor for translational Z masses. Translational dead load and live load masses are specified at nodes of each floor and interfloor type. These masses are scaled and combined to obtain the nodal masses for an instance. The translational Z (vertical) mass for an instance is obtained by scaling the nodal masses by the corresponding scale factor. The scale factors for Z masses may be set equal to 0.0 to neglect the effect of vertical inertia forces (see *PARAMETERS data). The alpha-M damping is obtained by assembling the masses scaled by alpha-M damping factors, for each instance .
FLRTDC	Diaphragm center data for each floor type. 1 : X coordinate of diaphragm center relative to the floor origin. 2 : Y coordinate of diaphragm center relative to the floor origin. 3 : Translational dead load mass at diaphragm center. 4 : Translational live load mass at diaphragm center. 5 : Rotational Z dead load mass at diaphragm center. 6 : Rotational Z live load mass at diaphragm center.
FMASTC	Master node coordinates for each floor instance. 1 : X coordinate relative to floor origin. 2 : Y coordinate relative to floor origin. The master node is located at the center of mass of the slaved nodes. Interfloor elements can contribute masses to the floor nodes, and so the location of the master node can be different for floor instances having the same type.
FMASTM	Master node masses for each floor instance. 1 : Translational X and Y masses. 2 : Rotational Z mass. The masses specified at the diaphragm center and on slaved nodes are transferred to the floor master node.

XYZIFR	X, Y and Z coordinates of the origin of floor 2 relative to the origin of floor 1 for each interfloor type.
XYZFIF	X, Y and Z coordinates of the origin of each floor and interfloor instance. The interfloor origin is the same as its floor 1 origin.
JIRFD	Rigid floor diaphragm code for each floor type, as follows. 0 : No rigid floor diaphragm. 1 : Rigid floor diaphragm, having all displacement DOFs (X and Y translations and Z rotation). 2 : Rigid floor diaphragm, free only to translate in X direction. 3 : Rigid floor diaphragm, free only to translate in Y direction. 4 : Rigid floor diaphragm, fixed in all displacement DOFs (X and Y translations and Z rotation).
JIFRGT	Floors connected by each interfloor type. 1 : Floor 1 (geometry) type number. 2 : Floor 2 (geometry) type number (0 = no floor 2). Only the floor geometry information is used in defining an interfloor type (for calculating dimensions of interfloor elements). Thus, two interfloor instances having the same type, may connect floor instances of different types, provided these floors have the same geometry.
JFLR	Output code for diaphragm center displacements (X and Y translations and Z rotation) for each floor instance, as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
JIFR	Floors connected by each interfloor instance. 1 : Floor 1 number. 2 : Floor 2 number (0 = no floor 2).
JFIFT	Type number for each floor and interfloor instance.
JFIF	Order of floors and interfloors in stiffness hypermatrix. 1 : Instance number for each block row (or column). 2 : Block row (or column) number for each instance. The first row maps the block rows (or columns) of the hypermatrix to the floor and interfloor instances. The second row has the reverse mapping.

NNODSF	Number of nodes in each floor and interfloor type.
NTNDSF	Number of nodes and subnodes in each floor and interfloor type.
NDSPF	Number of support springs in each floor and interfloor type.
NELGF	Number of element groups with nonzero /INFGR/ blocks in each floor and interfloor type.
NELGRF	Number of element groups in each floor and interfloor type.
NELTTF	Total number of elements in each floor and interfloor type.
NSSECF	Number of structure sections in each interfloor type. **Note:** Structure sections are not defined for floors.
MXCUTF	Largest number of elements cut by any section in each interfloor type.
NTRNSF	Number of force transformations for sections of each interfloor type.
MXTDFF	Largest number of DOFs in any force transformation for sections of each interfloor type.
NRDSF	Number of generalized displacements for each floor and interfloor type.
NEQF	Number of equations (i.e., DOFs) contributed by each floor and interfloor type.
NEQOF	Equation number offset for each floor and interfloor instance. The equations numbers for instance I of type IT start from NEQOF(I)+1 and end at NEQOF(I)+NEQF(IT).
NIADF	Length of IAD array (=NELTTF + NELGF) for each floor and interfloor type. See KIAD in /STOR3/.
NBLOKF	Number of disk blocks storing /INFGR/ and /INFEL/ blocks for each floor and interfloor instance. See KINFB in /STOR6/.

6.2.3. CONTR Common

/CONTR/ stores overall control information.

/CONTR/ is declared as follows.

```
COMMON /CONTR/ incor, nchar, ndspt, ndtp, nseg, nsnds,
1                        nofek, noftk, nofbk, nofdk
```

The variables are as follows.

Variable	Description
INCOR	Code for storage of /INFEL/ and /INFGR/ data, INFB; tangent stiffness, TANK; effective stiffness, EFFK; beta-K damping, BETAK; duplicate tangent stiffness, DTAN; and duplicate /INFEL/ and /INFGR/ data, DINFB. -1: INFB and DINFB are blocked and stored on files NFBEG, NFCUR and NFUPD; and TANK, EFFK, BETAK and DTAN are hypermatrices and stored on file NFSTFB. 0 : INFB is stored in memory; TANK, EFFK, BETAK and DTAN are hypermatrices and stored on file NFSTFB; and DINFB is stored on file NFBEG. 1 : INFB and DINFB are blocked and stored on files NFBEG, NFCUR and NFUPD; TANK and EFFK are stored in memory; and BETAK and DTAN are stored on file NFSTFB. 2 : INFB is stored in memory; TANK and EFFK are stored in memory; BETAK and DTAN are stored on file NFSTFB; and DINFB is stored on file NFBEG. 3 : INFB is stored in memory; TANK, EFFK and BETAK are stored in memory; DTAN is stored on file NFSTFB; and DINFB is stored on file NFBEG. 4 : INFB is stored in memory; TANK, EFFK, BETAK and DTAN are stored in memory; and DINFB is stored on file NFBEG. 5 : INFB, TANK, EFFK, BETAK, DTAN and DINFB are stored in memory. **Notes:** a) If INCOR $\leq$ 0, then hypermatrix storage scheme is used for TANK, EFFK, BETAK and DTAN. The address blocks are stored on file NFSTFA. b) If INCOR > 0, then compacted column storage scheme is used for TANK, EFFK, BETAK and DTAN. This is identical to the scheme used in DRAIN-2DX and 3DX.
NCHAR	Number of characters in FNAME (see /TITLE/). Used for opening permanent files.
NDSPT	Total number of support springs.
NDTP	Number of compound node types + 1.

NSEG	Current analysis segment number.
NSNDS	Total number of subnodes in all compound node types.
NOFEK	Record number offset for EFFK blocks in file NFSTFB if INCOR ≤ 0. Record number for EFFK in file NFSTFB if INCOR = 1 or 2. The space in memory for EFFK may be used temporarily for BETAK, in which case EFFK is backed up on NFSTFB.
NOFTK	Record number offset for TANK blocks in file NFSTFB if INCOR ≤ 0.
NOFBK	Record number offset for BETAK blocks in file NFSTFB if INCOR ≤ 0. Record number for BETAK in file NFSTFB if INCOR = 1 or 2.
NOFDK	Record number offset for DTAN blocks in file NFSTFB if INCOR ≤ 0. Record number of DTAN in file NFSTFB if INCOR = 1, 2 or 3.

6.2.4. CURRNT Common

For any element task (e.g., event factor calculation, stiffness formation, state determination, etc.), an element subroutine is called for each element. During model definition, an element subroutine is called for each element group. /CURRNT/ stores information on the current element.

/CURRNT/ is declared as follows.

```
COMMON /CURRNT/ ififc, igrc, ielc, idfc, inodc
```

The variables are as follows.

Variable	Description
IFIFC	Current floor or interfloor instance number during analysis. Current floor or interfloor type number during model definition. **Note:** In model definition phase, IFIFC = NFLRT when a floor type is processed, and IFIFC > NFLRT when an interfloor type is processed. During analysis, IFIFC ≤ NFLR when a floor instance is processed, and IFIFC > NFLR when an interfloor instance is processed (see NFLRT and NFLR in /FLRIFR/).
IGRC	Current element group number.
IELC	Current element number.
IDFC	Current element DOF number. Used only in model definition phase.
INODC	Current element node number. Used only in model definition phase.

6.2.5. DIMENS Common

Array dimensions may be passed through argument lists or common blocks. If a dimension is 0, the FORTRAN compiler may treat it as a fatal error (e.g., the Lahey compiler). /DIMENS/ mainly stores dummy values for dimensions which may be zero.

/DIMENS/ is declared as follows.

```
 COMMON /DIMENS/ mxsecd, mxrdsd, ndspd, nssecd, mxcutd,
1                mxtdfd, ntrnsd, nrdsd, nnodsd, ntndsd,
2                nelttd, nnod1d, ntnd1d, nnod2d, ntnd2d
```

The variables are as follows.

Variable	Description
MXSECD	MAX(1,MAXSEC), where MAXSEC is maximum number of structure sections for any instance.
MXRDSD	MAX(1,MAXRDS), where MAXRDS is maximum number of generalized displacements for any instance.

NDSPD	MAX(1,NDSPT), where NDSPT (in /CONTR/) is the total number of support-springs.
NSSECD	MAX(1,NSSEC), where NSSEC is the number of structure sections in the current instance (see NSSECF in /CNTFIF/).
MXCUTD	MAX(1,MAXCUT), where MAXCUT is maximum number of elements cut by any structure section in the current instance (see MXCUTF in /CNTFIF/).
MXTDFD	MAX(1,MAXTDF), where MAXTDF is maximum number of DOFs for any cut-element in the current instance (see MXTDFF in /CNTFIF/).
NTRNSD	MAX(1,NTRNS), where NTRNS is the number of force transformations for structure sections in the current instance (see NTRNSF in /CNTFIF/).
NRDSD	MAX(1,NRDS), where NRDS is the number of generalized displacements in the current instance (see NRDSF in /CNTFIF/).
NNODSD	MAX(1,NNODS), where NNODS is the number of nodes in the current instance (see NNODSF in /CNTFIF/).
NTNDSD	MAX(1,NTNDS), where NTNDS is the number of nodes and subnodes in the current instance (see NTNDSF in /CNTFIF/).
NELTTD	MAX(1,NELTOT), where NELTOT is the total number of elements in the current instance (see NELTTF in /CNTFIF/).
NNOD1D	MAX(1,NNODS1), where NNODS1 is the number of nodes in floor 1 of the current interfloor instance.
NTND1D	MAX(1,NTNDS1), where NTNDS1 is the number of nodes and subnodes in floor 1 of the current interfloor instance.
NNOD2D	MAX(1,NNODS2), where NNODS2 is the number of nodes in floor 1 of the current interfloor instance.
NTND2D	MAX(1,NTNDS2), where NTNDS2 is the number of nodes and subnodes in floor 2 of the current interfloor instance.

6.2.6. DISVEL Common

Element end displacements and velocities are sent to the element subroutines for certain element tasks (e.g., event factor calculation, state determination), and element end forces are returned. /DISVEL/ provides memory for these values. Currently these values are transferred through argument lists. In future programs, /DISVEL/ may be used to transfer these values, in which case /DISVEL/ will become a part of the interface between the base program and the element subroutines.

/DISVEL/ is declared as follows.

```
COMMON /DISVEL/ ddise(MXDOFP), vele(MXDOFP), dise(MXDOFP),
1                  relas(MXDOFP), rdamp(MXDOFP), rinit(MXDOFP)
```

The variables are as follows.

Variable	Description
DDISE	Increment in element end displacements.
VELE	Element end velocities.
DISE	Element end displacements.
RELAS	Element end static forces.
RDAMP	Element end damping forces.
RINIT	Element end initial forces due to element loads applied in static gravity analysis.

6.2.7. ENRGY Common

/ENRGY/ stores the work quantities for checking energy balance. In addition to the current values, the values at the beginning of the current time step are also stored. These are backup values in case the time step must be repeated.

/ENRGY/ is declared as follows.

```
COMMON /ENRGY/ tek, tei, ted, tes, tew, tep, teso, ten, tee, tead,
1                teki, teii, tedi, tesi, tewi, tepi, tesoi, teni, teei, teadi,
2                eneg(MXGRP,MFIFP), enrd(MXGRP,MFIFP),
3                eext(MXGRP,MFIFP), enegi(MXGRP,MFIFP),
4                enrdi(MXGRP,MFIFP), eexti(MXGRP,MFIFP)
```

The variables are as follows.

Variable	Description
TEK	Total kinetic energy (from nodal masses and velocities).
TEI	Total inertia work (work done by inertia forces).
TED	Total damping work (TEAD+TEBD).
TES	Total element static work.
TEW	Total work done by dynamic nodal loads (consists of work done by effective loads in ground acceleration analysis and by support reactions in ground displacement analysis).
TEP	Total work done by static nodal loads.
TESO	Total second-order work (i.e., work done by P-Δ shears).
TEN	Energy error (= TES+TEI+TED-TEE).
TEE	Total external work (=TEP+TEW+TESO).
TEAD	Total alpha-M damping work.
TEKI	TEK at start of time step.
TEII	TEI at start of time step.
TEDI	TED at start of time step.
TESI	TES at start of time step.
TEWI	TEW at start of time step.
TEPI	TEP at start of time step.
TESOI	TESO at start of time step.
TENI	TEN at start of time step.
TEEI	TEEI at start of time step.
TEADI	TEAD at start of time step.
ENEG	Static work done in each element group.
ENRD	Beta-K damping work done in each element group.
EEXT	Second-order work done in each element group.
ENEGI	ENEG at start of time step.
ENRDI	ENRD at start of time step.
EEXTI	EEXT at start of time step.

6.2.8. EQNS Common

/EQNS/ stores variables used in the solution of equations.

/EQNS/ is declared as follows.

```
 COMMON /EQNS/ neqt, neqqt, neqqmx, neqfmx, neqimx,
1                    nstfd, nstfo, lenk, jcol, maxdof
```

The variables are as follows.

Variable	Description
NEQT	Total number of equations.
NEQQT	NEQT+1.
NEQQMX	1+MAX(neqfmx,neqimx).
NEQFMX	Maximum number of DOFs in any floor instance. See NEQF in /CNTFIF/.
NEQIMX	Maximum number of DOFs in any interfloor instance. See NEQF in /CNTFIF/.
NSTFD	Length of a diagonal memory block (see KSTIF in /STOR7/). $=\text{NEQQMX} \times (\text{NEQQMX}-1)/2$
NSTFO	Length of an off-diagonal memory block (see KSTIF in /STOR7/). $=(\text{NEQQMX}-1) \times (\text{NEQQMX}-1)$
LENK	Length of compacted column stiffness matrix.
JCOL	First (block) column in stiffness (hyper) matrix that has changed due to element events in the most recent state determination. All (block) columns from this point must be refactorized, but (block) columns up to this point do not change.
MAXDOF	Maximum number of DOFs for any element in the current structure. Must be less than MXDOFP (see /DISVEL/).

6.2.9. EVENT Common

/EVENT/ stores variables used in the event-to-event scheme.

/EVENT/ is declared as follows.

```
 COMMON /EVENT/ afac, facc, unbl(6), unbf, unbm,
1                irdof, irelm, irend, irevnt, irfif, irfifc,
2                irgrp, irnod, iquit, maxev, neven
```

The variables are as follows.

Variable	Description
AFAC	Accumulated event factor for current load or time step (proportion of step "used up").
FACC	Event factor for current substep.
UNBL	Maximum equilibrium unbalance in each displacement direction.
UNBF	Maximum force unbalance.
UNBM	Maximum moment unbalance.
IRDOF	Governing displacement direction if collapse displacement exceeded.
IRELM	Element number with the smallest event factor.
IREND	Event type code for the element with the smallest event factor. The meaning of this code depends on the element type. See element User Guides.

IREVNT	Code for the event type. 0 : No event. 1 : Element event. 2 : Load factor increment reached for load or time step. 3 : Controlled displacement increment reached for load step. 4 : Load removed to satisfy displacement control. 5 : Load factor increment or time increment reached for analysis segment. 6 : Controlled displacement increment reached for segment. 7 : Maximum number of steps reached for segment. 8 : Maximum number of events reached for load or time step. 9 : Maximum number of successive direction changes (flip-flops) exceeded for the step. 10 : Collapse translation exceeded. 11 : Collapse rotation exceeded. 12 : Structure unstable (detected by negative or zero term on the diagonal during factorization of the effective stiffness matrix). 13 : Displacement control failed to prevent flip-flop.
IRFIF	Instance number of element with smallest event factor.
IRFIFC	Instance number for which collapse displacement exceeded.
IRGRP	Group number of element with smallest event factor.
IRNOD	Node number at which collapse displacement exceeded.
IQUIT	Termination code at end of current load or time step. 0 : Proceed to the next step as current analysis segment has not been completed ($0 \leq$ IREVNT ≤ 4). 1 : Proceed to next segment as current segment has been successfully completed ($5 \leq$ IREVNT ≤ 6). -1 : Quit analysis without completing current segment as analysis cannot proceed further ($7 \leq$ IREVNT ≤ 13).
MAXEV	Maximum number of events allowed in a load or time step.
NEVEN	Number of events in current step.

6.2.10. FLRIFR Common

/FLRIFR/ mainly stores the number of floors and interfloors.

/FLRIFR/ is declared as follows.

```
_COMMON /FLRIFR/ nflrt, nifrt, nfift, nflr, nifr, nfif, nfifp,
1                        nzero, nnzero, maxgr
```

The variables are as follows.

Variable	Description
NFLRT	Number of floor types.
NIFRT	Number of interfloor types.
NFIFT	Number of floor and interfloor types (=NFLRT+NIFRT).
NFLR	Number of floor instances.
NIFR	Number of interfloor instances.
NFIF	Number of floor and interfloor instances (=NFLR+NIFR).
NFIFP	NFIF+1.
NZERO	Number of stiffness blocks within the block envelope of the stiffness hypermatrix.
NNZERO	Number of nonzero stiffness blocks within the block envelope of the stiffness hypermatrix.
MAXGR	Maximum number of element groups allowed in any floor or interfloor type (=MAXGRP. See PARA.H).

6.2.11. GENINF Common

/GENINF/ stores data for each element group.

/GENINF/ is declared as follows.

```
COMMON /GENINF/ betao(MAXGRP,MFIFTP), ovfac(MAXGRP,2,MFIFP),
1               relpr(2,MAXGRP,MFIFP), ielpr(2,MAXGRP,MFIFP),
2               kelem(MAXGRP,MFIFTP), kevnt(MAXGRP,MFIFTP),
3               kgeom(MAXGRP,MFIFTP), nedof(MAXGRP,MFIFTP),
4               nelem(MAXGRP,MFIFTP), nenod(MAXGRP,MFIFTP),
5               ninfe(MAXGRP,MFIFTP), ninfel(MAXGRP,MFIFTP),
6               ninfg(MAXGRP,MFIFTP), ninfl(MAXGRP,MFIFTP),
7               ninft(MAXGRP,MFIFTP), nlinf(MAXGRP,MFIFTP)
```

The variables are as follows.

Variable	**Description**
BETAO	Stiffness proportional damping factor for each group of each floor and interfloor type.
OVFAC	Event overshoot scale factor for each group of each floor and interfloor instance. 1 : for static analyses. 2 : for dynamic analyses.
RELPR	Real element parameters for each group of each floor and interfloor instance.
IELPR	Integer element parameters for each group or each floor and interfloor instance.
KELEM	Element type number for each group of each floor and interfloor type.
KEVNT	Event code for each group of each floor and interfloor type, as follows. 0 : Suppress event factor calculation. 1 : Calculate event factor.

KGEOM	$P-\Delta$ analysis code for each group of each floor and interfloor type, as follows. 0 : Ignore $P-\Delta$ effects. 1 : Consider $P-\Delta$ effects and allow geometric stiffness to change for static analyses only. 2 : Consider $P-\Delta$ effects and allow geometric stiffness to change for both static and dynamic analyses.
NEDOF	Number of element DOFs for each group of each floor and interfloor type.
NELEM	Number of elements in each group of each floor and interfloor type.
NENOD	Number of nodes per element for each group of each floor and interfloor type.
NINFE	Length (in 4-byte units) of /INFEL/ block for each group of each floor and interfloor type.
NINFEL	Location of integrity violation variable in /INFEL/ for each group of each floor and interfloor type. Certain element data is stored at the end of /INFEL/ by the base program. If an element subroutine writes more than the specified /INFEL/ length, this data is destroyed. To warn against this during element development, this variable is checked after each call to an element subroutine. If the variable has been over-written, the program writes an error message.
NINFG	Length (in 4-byte units) of /INFGR/ block for each group of each floor and interfloor type.
NINFL	Length (in 4-byte units) of one element load set for each group of each floor and interfloor type.
NINFT	Number of output items per element for static or dynamic analyses for each group (length of /THELM/) of each floor and interfloor type.
NLINF	Number of output items per element for response spectrum analysis for each group of each floor and interfloor type.

6.2.12. RHIST Common

/RHIST/ stores counts of numbers of items for results printout and post-processing.

/RHIST/ is declared as follows.

```
COMMON /RHIST/ lrec, ntime, nptime, ndpout(MFIFP), nelth(MFIFP),
1              npsec(MIFRP), nnrds(MFIFP), ndpost(MFIFP),
2              nelthp(MFIFP), npsecp(MIFRP), nnrdsp(MFIFP)
```

The variables are as follows.

Variable	Description
LREC	Length of buffer REC (fwa = KREC in /STOR6/) for output of time-history and/or envelope results.
NTIME	Number of printout sets in the current analysis segment.
NPTIME	Number of post-processing sets in the current analysis segment.
NDPOUT	Number of nodes in a printout set for each instance.
NELTH	Number of elements in a printout set for each instance.
NPSEC	Number of structure sections in a printout set for each instance.
NNRDS	Number of generalized displacements in a printout set for each instance.
NDPOST	Number of nodes in a post-processing set for each instance.
NELTHP	Number of elements in a post-processing set for each instance.
NPSECP	Number of structure sections in a post-processing set for each instance.
NNRDSP	Number of generalized displacements in a post-processing set for each instance.

6.2.13. SECTON Common

/SECTON/ is used in DRAIN-2DX and 3DX to store structure section information. In DRAIN-BUILDING this information is stored in /CNTFIF/.

6.2.14. SETREL Common

/SETREL/ is used in DRAIN-2DX and 3DX to store generalized displacement information. In DRAIN-BUILDING this information is stored in /CNTFIF/.

6.2.15. STFBLK Common

For the hypermatrix storage scheme, /STFBLK/ stores the record numbers of tangent stiffness blocks that are currently in memory or that are required to be brought in memory when a stiffness change occurs for an element of the current instance.

/STFBLK/ is declared as follows.

```
COMMON /STFBLK/ inmem, lstblk(6)
```

The variables are as follows.

Variable	**Description**
INMEM	Code for existence of blocks in memory, as follows. 0 : Blocks are not in memory. Bring them into memory if and when the stiffness of an element changes in the current instance, and reset the code to 1. 1 : Blocks are in memory. Write them to the NFSTFB file after processing elements of the current instance. Before processing the elements of any instance, the record numbers of the blocks required for the current instance are computed and INMEM is set to 0.

LSTBLK	Record numbers of stiffness blocks that may change due to changes in element stiffnesses for elements of the current instance. The blocks are in the following order. For a floor: 5 : Diagonal block for the floor. 1-4 and 6 : 0. For an interfloor with interfloor nodes: 1 : Diagonal floor 1 block. 2 : Off-diagonal block coupling floor 1 and floor 2. 3 : Diagonal floor 2 block. 4 : Off-diagonal block coupling floor 1 and interfloor. 5 : Diagonal block for the interfloor. 6 : Off-diagonal block coupling floor 2 and interfloor. For an interfloor with no interfloor nodes: 1 : Diagonal floor 1 block. 2 : Off-diagonal block coupling floor 1 and floor 2. 3 : Diagonal floor 2 block. 4-6 : 0.

6.2.16. STOR Common

/STOR/ is used in DRAIN-2DX and 3DX to store first word addresses of arrays in blank common, plus some variables used frequently with the storage scheme. In DRAIN-BUILDING this information is now distributed in /STOR1/, /STOR2/, /STOR3/, /STOR4/, /STOR5/, /STOR6/, /STOR7/ and /STOR8/.

6.2.17. STOR1 Common

/STOR1/ stores the **first word addresses** of some arrays in blank common, plus some variables used frequently with the storage scheme.

/STOR1/ is declared as follows.

```
 COMMON /STOR1/ ntst, ninfb, ntblok, ksofar, kielnd,
1                 kfmnd, kxlodd, kwksp, nwkspc, nwksp
```

The variables are as follows.

Variable	Description
NTST	Length of blank common in 4-byte units.
NINFB	Length of INFB array = size of each disk block used to store /INFGR/ and /INFEL/.
NTBLOK	Total number of disk blocks used to store /INFGR/ and /INFEL/ for all elements of all instances.
KSOFAR	Next unallocated address in blank common.
NWKSPC	Size of work space buffer in real (i.e., 8-byte) words.
NWKSP	Size of work space buffer in integer (i.e., 4-byte) words.

The arrays stored in blank common are as follows.

FWA	Array	Description
KIELND	IELNOD(nenod,nmem)	Temporary array for storing element end nodes for each element group of a floor or interfloor type. This array is written to .GEO file after the input of the element group is complete. NENOD is the number of nodes per element and NMEM is the number of elements in the group. IELNOD is coded as follows. n : 'n'= IFIF + 3 × NLOC where, IFIF = 0 if the node belongs to the floor or interfloor. 1 if the node belongs to floor 1 of the interfloor. 2 if the node belongs to floor 2 of the interfloor. NLOC = sequence number of the node in its floor or interfloor. The data can be recovered as follows. IFIF = MOD (n,3) NLOC = n/3

KFMND	FMND(2,nodtot))	Temporary array used for accumulating nodal (dead load and live load) masses for a floor or interfloor type during model definition. This array is written to .MAS file (unit NFMASS) after the input of the floor or interfloor type is complete. For a floor type, NODTOT equals the number of nodes in the floor. For an interfloor type, NODTOT equals the total number of nodes in the interfloor, its floor 1 and its floor 2. Note that in DRAIN-BUILDING, nodal masses can be specified directly as well as through elements.
KXLODD	XLOD (neqt)	Effective nodal load increment for the current load or time step. KXLODD = KXLOD in /STOR6/.
KWKSP	WKSPC(nwkspc) for real work space. WKSPC(nwksp) for integer work space.	Work space for use by base program subroutines. KWKSP = KWKSPC in /STOR6/

6.2.18 STOR2 Common

/STOR2/ stores the first word addresses of those arrays in blank common, which contain data for the compound node types.

/STOR2/ is declared as follows.

```
COMMON /STOR2/ knsb, kndfsb, kcosb
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KNSB	NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type, NT, is equal to NSB(nt+1)-NSB(nt).
KNDFSB	NDFSB(6,nsnds)	DOF codes for all subnodes of each compound node type, as follows. 0 : Absolute displacement. 1 : Restrained (not a DOF). 2 : Relative displacement w.r.t. main node. 3 : Special degree of freedom (i.e., not a conventional translation or rotation).
KCOSB	COSB(3,nsnds)	Coordinate offsets from main node for subnodes of each compound node type.

6.2.19. STOR3 Common

/STOR3/ stores the first word addresses of those arrays in blank common, which contain data for floor and interfloor types or output specification for floor and interfloor instances.

/STOR3/ is declared as follows.

```
 COMMON /STOR3/ kndid(MFIFTP), kcoord(MFIFTP), kid(MFIFTP),
1                   kidsp(MFIFTP), knecut(MIFRTP), kdist(MIFRTP),
2                   kidcut(MIFRTP), kstrns(MIFRTP), knodir(MIFRTP),
3                   krdfac(MFIFTP), kiad(MFIFTP), kfmnod(MFIFP),
4                   kjnod(MFIFP), kjelm(MFIFP), kjsec(MFIFP),
5                   kjrds(MFIFP)
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KNDID	NDID(3,nnods)	Node identification array for each floor and interfloor type. NNODS is the number of nodes in the floor or interfloor (see NNODSF in /CNTFIF/). 1 : Node number. 2 : Compound node type number (0 = not a compound node). The compound node types are numbered in the order of input. 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array. Note that for a repeated floor geometry type, KNDID stores the fwa of NDID of the parent floor type.
KCOORD	COORD(3,nnods)	Nodal coordinates, in ascending node number order for each floor and interfloor type. The coordinates are relative to the origin of the floor or interfloor type. Note that for a repeated floor geometry type, KCOORD stores the fwa of COORD of the parent floor type.

KID	ID(6,ntnds)	Equation numbers for displacements at each node and subnode, for each floor and interfloor type. NTNDS is the number of nodes and subnodes in the floor or interfloor (see NTNDSF in /CNTFIF/). ID is coded as follows. 0 : Restrained displacement. +n : Displacement is unrestrained and unslaved. 'n'= local equation number. -n : Displacement is slaved to the floor master node. 'n' = NTNDS. For an interfloor there are no slaved displacements. For a floor, the DOFs of the floor master node are at location NTNDS. The coordinates of the floor master node are in FMASTC (in /CNTFIF/).
KIDSP	IDSP(ndsp)	Local equation numbers for spring supported displacements for each floor and interfloor type. NDSP is the number of support springs in the floor or interfloor (see NDSPF in /CNTFIF/).
KNECUT	NECUT(nssec)	Number of elements cut by each structure section for each interfloor type. NSSEC is the number of sections in the interfloor (see NSSECF in /CNTFIF/).
KDIST	DIST(3,maxcut,nssec)	X, Y and Z coordinate offsets from structure section centers to element cuts for each interfloor type. MAXCUT is the maximum number of cut-elements in the interfloor (see MXCUTF in /CNTFIF/).
KIDCUT	IDCUT(3,maxcut,nssec)	Identification array for cut-elements of structure sections for each interfloor type, as follows. 1 : Group number. 2 : Element number. 3 : Force transformation number.

KSTRNS	STRNS(6,maxtdf,ntrns)	Force transformation matrices for structure sections for each interfloor type. MAXTDF is the largest number of DOFs for any transformation and NTRNS is the number of transformations (see MXTDFF and NTRNSF in /CNTFIF/) in the interfloor.
KNODIR	NODIR(8,nrds)	Displacements forming generalized displacements for each floor and interfloor type. NRDS is the number of generalized displacements for the floor or interfloor (see NRDSF in /CNTFIF/). NODIR is coded as follows. n : 'n' = IFIF + IDIR*3 + NLOC*21 Where, IFIF = 0 for the floor or interfloor. 1 for floor 1 of the interfloor. 2 for floor 2 of the interfloor. IDIR = DOF direction (1 to 6). NLOC = Sequence number of the node. The data is recovered as follows. IFIF = MOD (n,3) IDIR = MOD (n,21)/3 NLOC = n/21
KRDFAC	RDFAC(8,nrds)	Participation factors for displacements forming the generalized displacement for each floor and interfloor type.
KFMNOD	FMNOD(nnods)	Nodal masses for each floor and interfloor instance. FMNOD stores the sum of dead load and live load masses. Note: In DRAIN-BUILDING only translational masses are specified at nodes.

KIAD	IAD(niad)	Relative first word addresses in INFB for each /INFGR/ and /INFEL/ block, for each floor and interfloor type. NIAD is obtained from NIADF in /CNTFIF/ for the floor or interfloor. The absolute first word addresses are obtained by combination of IADOF (see KIADOF in /STOR4/) and IAD.
KJNOD	JNOD(nnods)	Output codes for nodal displacements for each floor and interfloor instance. JNOD is coded as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KJELM	JELM(neltot)	Output codes for element results for each floor and interfloor instance. JELM is coded as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KJSEC	JSEC(nssec)	Output codes for structure section results for each interfloor instance. JSEC is coded as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.
KJRDS	JRDS(nrds)	Output codes for generalized displacements for each floor and interfloor instance. JRDS is coded as follows. 0 : Neither post-processing nor printout. 1 : Post-processing only, no printout. 2 : Both post-processing and printout.

6.2.20. STOR4 Common

/STOR4/ stores the first word addresses of some arrays in blank common.

/STOR4/ is declared as follows.

```
COMMON /STOR4/ kidspt, kspdsp, kiadof, kiexnv, kienv, kfmdof, kalpha
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KIDSPT	IDSPT(ndspt)	Global equation numbers for each spring supported displacement. NDSPT is in /CONTR/.
KSPDSP	SPDSP(ndspt,2)	Imposed displacements at spring supports. (1: Total ; 2: Increment for current step).
KIADOF	IADOF(nfif)	Offset in IAD for each instance. See KIAD in /STOR3/.
KIEXNV	IEXNV(nfifp)	Location of the first nonzero block in IENV for each block column (i.e., pointer to the start of each block column in IENV). NFIFP-1 is the number of block rows (and block columns), which is equal to the number of floor and interfloor instances. For uniformity in indexing, IEXNV(nfifp) is set equal to NZERO+1.
KIENV	IENV(nzero)	Addresses (i.e., record number offsets on the direct access file - see Section 5.6) of blocks within the block envelope. For blocks that are initially zero and do not suffer fill-in, IENV is set to 0. NZERO is the number of blocks within the block envelope.
KFMDOF	FMDOF(neqt)	Mass matrix (diagonal).
KALPHA	ALPHAM(neqt)	Alpha-M damping matrix (diagonal).

6.2.21. STOR5 Common

/STOR5/ stores the first word addresses of arrays in blank common, which contain result envelopes.

/STOR5/ is declared as follows.

```
COMMON /STOR5/ kenp(MFIFP), kenn(MFIFP),
1              kistp(MFIFP), kistn(MFIFP),
2              ksecen(MIFRP), kisece(MIFRP),
3              krdsen(MFIFP), kirdse(MFIFP)
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KENP	DENP(6,ntnds)	Positive nodal displacement envelopes for each floor and interfloor instance.
KENN	DENN(6,ntnds)	Negative nodal displacement envelopes for each floor and interfloor instance.
KISTP	ISTP(6,ntnds)	Step numbers for DENP.
KISTN	ISTN(6,ntnds)	Step numbers for DENN.
KSECEN	SECENV(6,6,nssec)	Section force envelopes for each interfloor instance. 2nd index indicates: 1 : Total positive. 2 : Total negative. 3 : Static positive. 4 : Static negative. 5 : Damping positive. 6 : Damping negative.
KISECE	ISECEN(6,6,nssec)	Step numbers for SECENV.
KRDSEN	RDSENV(2,nrds)	Positive and negative generalized displacement envelopes for each floor and interfloor instance.
KIRDSE	IRDSEN(2,nrds)	Step numbers for RDSENV.

6.2.22. STOR6 Common

/STOR6/ mainly stores the first word addresses of vectors in blank common.

/STOR6/ is declared as follows.

```
 COMMON /STOR6/ kwkspc, ksefor, krds, krec, kxlod, kexts, kext, kdext,
1               krints, krint, kru, kdisi, kveli, kacci, kdds, kcvel,
2               kcacc, kdis, kvel, kacc, kenri, kdru, kinfb, kdinfb
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KWKSPC	WKSPC(nwkspc) for real work space. WKSPC(nwksp) for integer work space.	Work space for use by base program subroutines.
KSEFOR	SEFOR(12,mxsecd)	Current structure section forces (6 static values followed by 6 damping values for each section) for the current interfloor instance. MXSECD is in /DIMENS/.
KRDS	RDS(mxrdsd)	Current generalized displacement magnitudes for the current floor or interfloor instance. MXRDSD is in /DIMENS/.
KREC	REC(lrec)	Buffer for output of time-history and/or envelope results for post-processing and/or printout.
KXLOD	XLOD(neqt)	Effective nodal load increment for the current load or time step.
KEXTS	EXTS(neqt)	Total static load.
KEXT	EXT(neqt)	Total static + dynamic load.
KDEXT	DEXT(neqt)	Dynamic load increment for the current time step. For *ACCN or *ACCR analysis DEXT is the load increment due to ground acceleration increments. For *DISN or *DISR analysis DEXT is the load increment required to impose the specified ground displacement increments.

KRINTS	RINTS(neqt)	Static resisting force.
KRINT	RINT(neqt)	Total (static + damping + inertia) resisting force.
KRU	RU(neqt)	Unbalanced load (EXT-RINT).
KDISI	DISI(neqt)	Displacements at start of current time step.
KVELI	VELI(neqt)	Velocities at start of current time step.
KACCI	ACCI(neqt)	Accelerations at start of time step.
KDDS	DDIS(neqqt)	Displacement increment in current substep.
KCVEL	CVEL(neqt)	Velocity increment in current substep.
KCACC	CACC(neqt)	Acceleration increment in current substep.
KDIS	DIS(neqt)	Total displacements.
KVEL	VEL(neqt)	Total velocities.
KACC	ACC(neqt)	Total accelerations.
KENRI	ENRI(neqt)	Work done by inertia forces in current time step.
KDRU	DRU(neqt)	Backed up RU at start of a time step. Used if time step is repeated.
KINFB	INFB(ninfb)	Buffer for /INFGR/ and /INFEL/ blocks.
KDINFB	IINFB(ninfb)	Backed up INFB (if KDINFB ≠ 1). Used if time step is repeated. See INCOR in /CONTR/.

6.2.23. STOR7 Common

For the hypermatrix storage scheme (INCOR $\leq$ 0, see /CONTR/), /STOR7/ stores the first word addresses of buffers in blank common.

/STOR7/ is declared as follows.

```
COMMON /STOR7/ kdiag, kstif(6), kistif(6)
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KDIAG	DIAG(neqt)	Diagonal of effective stiffness hypermatrix. NEQT is in /EQNS/.
KSTIF(1)	STIF1(nstfd)	Buffer for a diagonal stiffness block. NSTFD is in /EQNS/.
KSTIF(2)	STIF2(nstfo) *or* STIF2(nstfd)	Buffer for an off-diagonal or diagonal stiffness block. NSTFO is in /EQNS/.
KSTIF(3)	STIF3(nstfd)	Buffer for a diagonal stiffness block.
KSTIF(4)	STIF4(nstfo) *or* STIF4(nstfd)	Buffer for an off-diagonal or diagonal stiffness block.
KSTIF(5)	STIF5(nstfd)	Buffer for a diagonal stiffness block.
KSTIF(6)	STIF6(nstfo) *or* STIF6(nstfd)	Buffer for an off-diagonal or diagonal stiffness block.
KISTIF(1)	ISTIF1(neqqmx)	Index array for STIF1. Locations of the first nonzero terms in STIF1 for each column. NEQQMX is in /EQNS/).
KISTIF(2)	ISTIF2(neqqmx)	Index array for STIF2.
KISTIF(3)	ISTIF3(neqqmx)	Index array for STIF3.
KISTIF(4)	ISTIF4(neqqmx)	Index array for STIF4.
KISTIF(5)	ISTIF5(neqqmx)	Index array for STIF5.
KISTIF(6)	ISTIF6(neqqmx)	Index array for STIF6.

In HYPSOL, DIAG and even numbered buffers are used to store the blocks of the effective stiffness. In HYPMUL, STIF2 and ISTIF2 are used to store the blocks of tangent stiffness or beta-K damping.

During state determination, for a floor instance, STIF5 and ISTIF5 are used to store the floor diagonal block. For an interfloor instance all buffers STIF1-6 and ISTIF1-6 may be used. See Section 5.6.

6.2.24. STOR8 Common

For the compacted column storage scheme (INCOR > 0, see /CONTR/), /STOR8/ stores the first word addresses of compacted column matrices in blank common. See Section 5.10.

/STOR8/ is declared as follows.

```
COMMON /STOR8/ kdtan, kbetak, keffk, ktank, klstif
```

The arrays stored in blank common are as follows.

FWA	Array	Description
KDTAN	DTAN(lenk)	Backed up TANK if INCOR ≥4 (see /CONTR/). Otherwise DTAN is on file NFSTFB. Used if time step is repeated.
KBETAK	BETAK(lenk)	Beta-K damping matrix if INCOR≥3 (see /CONTR/). Otherwise BETAK is on file NFSTFB. If INCOR=1 or 2, then KBETAK is set equal to KEFFK.
KEFFK	EFFK(lenk)	Current effective stiffness matrix (factorized).
KTANK	TANK(lenk)	Current tangent stiffness matrix (unfactorized).
KLSTIF	LSTIF(neqqt)	Location of diagonal terms in compacted column stiffness and beta-K damping matrices.

6.2.25. TAPES Common

/TAPES/ stores the unit numbers for disk files. The unit numbers are assigned in BLOCK DATA (file BLOCK.FOR).

/TAPES/ is declared as follows.

```
 COMMON /TAPES/ inp, iou, inpx, nfbeg, nfcur, nfupd, nfscrt, nfres,
1               nfperm, nfload, nflis, nfoutp, nflog, nfgeo, nfprnt,
2               nfmode, nfmrsl, nfasdf, nfenvp, nffift, nfmass, nfstfa, nfstfb
```

The variables are as follows.

Unit No.	Associated File(s)
INP	Input file, DRAIN.INP.
IOU	Echo file, .ECH.
INPX	Input scratch file. Used for counting input items before data is actually read.
NFBEG	Scratch file storing /INFGR/ and /INFEL/ blocks at start of current step. Each disk block may contain one or more /INFGR/ and /INFEL/ blocks. The size of each disk block is NINFB (in /STOR1/).
NFCUR	Scratch file storing /INFGR/ and /INFEL/ blocks at start of current substep.
NFUPD	Scratch file storing updated /INFGR/ and /INFEL/ blocks, at end of current substep.
NFSCRT	Not used.
NFRES	Results post-processing file, .RXX, where XX is analysis segment number.
NFPERM	Structure state file, .SXX, where XX is analysis segment number.
NFLOAD	Files containing load patterns and dynamic load records (i.e., .ELD, .STA, .SPC, .ACC, .VEL, .DIS, .FRC).
NFLIS	Analysis list file, .LST.
NFOUTP	Printout file, .OUT.
NFLOG	Solution log file, .SLO.
NFGEO	Geometry file, .GEO.
NFPRNT	Scratch file for saving results for printout. Results are saved step-by-step, then reorganized and written to .OUT file item-by-item.
NFMODE	Mode shapes file, .MXX, where XX is analysis segment number.
NFMRSL	Modal response file(s), .UXX, for unit spectral acceleration amplitude. XX is the segment number.
NFASDF	Scratch file for dynamic load records. Used during dynamic analysis.
NFENVP	Envelope files, .EXX, where XX is analysis segment number, for envelope post-processing.

NFFIFT	.TYP file, storing the individual /INFGR/ and /INFEL/ blocks for each element group and element, respectively, for each floor and interfloor type.
NFMASS	.MAS file, storing nodal mass data for each floor and interfloor type.
NFSTFA	Direct access scratch file, storing index blocks (ISTIF) for the hypermatrix storage scheme.
NFSTFB	Direct access scratch file, storing data blocks (STIF) for the hypermatrix storage scheme; or stiffness and damping matrices for the compacted column storage scheme (see INCOR in /CONTR/).

6.3. BLANK COMMON

All variable length arrays are stored sequentially in blank common. Blank common is declared as follows.

```
COMMON L(*)
```

The total space occupied by all variable length arrays is limited only by the size of array L, defined by the parameter NTSTP in the main program (file MAIN.FOR) as follows.

```
 PARAMETER (NTSTP=100000)
COMMON L(NTSTP)


ntst = NTSTP
```

Arrays in L are located by their first word addresses (FWA). The arrays and their addresses have been listed in /STOR1-8/. If an array is not allocated space, because it is not required for the current problem or because it is not kept in memory, its FWA is set to 0 or 1. Therefore L(1) is not used and first allocated address is L(2).

Most real variables are currently in double precision (REAL*8). It is possible that some of these variables will be made single precision in future versions.

The variable KSOFAR stores the first unallocated address in L at any stage of execution of the program. To allocate, say, an integer array NDID of dimensions (3, NTNDS) for floor or interfloor type IFT, the code is as follows.

```
kndid(ift) = ksofar
ksofar = kndid(ift) + 3 × ntnds
```

Each real array is made to start on a 8-byte boundary by assigning an odd FWA. For example, to allocate a real array COORD of dimensions (3, NNODS) for floor or interfloor type IFT, the code is as follows.

```
kcoord(ift) = NXTODD (ksofar)
ksofar = kcoord(ift) + 3 × nnods × iprec
```

In the above, NXTODD is the simple statement function:

```
NXTODD(k) = k + MOD(k+1,2)
```

After each allocation in L, KSOFAR is checked to ensure that it is less than NTST. If KSOFAR > NTST, then blank common length is insufficient to run the problem, and the program writes an ERROR message in the .ECH file. The parameter NTSTP in the main program (file MAIN.FOR) must be increased to run the problem.

6.4. PERMANENT FILES

The base program creates several permanent files. All permanent files have names of the form PROBNAME.EXT, where PROBNAME is the problem name (up to 8 characters), FNAME in /TITLE/, and EXT is a three character extension indicating the contents of the file.

All permanent files created in DRAIN-2DX or 3DX are also created in DRAIN-BUILDING. In DRAIN-BUILDING, two additional files with extensions .TYP and .MAS are created. The description of these files follows.

FILES STORING DATA FOR FLOOR AND INTERFLOOR TYPES		
Extension	**Unit No.**	**Contents**
.TYP	NFFIFT	Individual /INFGR/ and /INFEL/ blocks for each element group and element, respectively, for each floor and interfloor type.
.MAS	NFMASS	Dead and live load nodal masses for each floor and interfloor type (see KFMND in /STOR1/).

In the current version of DRAIN-BUILDING these files could as well be scratch files. In a customized version (involving relatively minor changes in the program), however, these files could function as a library of standard floor and interfloor types, from which the model for analysis could be assembled.

6.5. TEMPORARY FILES

Temporary files (i.e., FORTAN scratch or internal files) are used for the following.

(a) Input processing.

(b) Output processing.

(c) Storage of element /INFGR/ and /INFEL/ blocks.

(d) Storage of duplicate element /INFGR/ and /INFEL/ blocks.

(e) Storage of index and data blocks for the hypermatrix storage scheme (if INCOR $\leq$ 0).

(f) Storage of EFFK, BETAK and DTAN for the compacted column storage scheme (if 0< INCOR $\leq$3).

The details for (a), (b) and (c) are as for DRAIN-2DX and 3DX.

Duplicate /INFGR/ and /INFEL/ blocks are stored in DINFB if INCOR=5. Otherwise, they are stored on NFBEG. NFBEG is not used for storage of DTAN and DRU as is the case for DRAIN-2DX and 3DX. DRU is always stored in memory. DTAN is stored in memory if INCOR $\geq$ 4, otherwise on file, NFSTFB.

The details for item (e) and (f) are described in Sections 5.6 and 5.10, respectively.

7. BASE PROGRAM ORGANIZATION -- DRAIN-BUILDING

7.1. INTRODUCTION

The base program phases and the corresponding separator lines in the DRAIN.INP file are as follows.

No.	Base Program Phase	Separator Lines	Comment
1.	Begin Session	*START/ *RESTART	For *RESTART, phases (2), (9), and (10) are skipped.
2.	Define Compound Node Types	*COMPOUND	Part of Model Definition.
3.	Define Floor Types	*FLOORTYPE, *NODECOORDS, *NODETYPES, *UNSLAVE, *RESTRAINTS, *MASSES, *ELEMENTGROUP, *GENDISP	Part of Model Definition.
4.	Define Interfloor Types	*INTERFLOORTYPE, *NODECOORDS, *NODETYPES, *RESTRAINTS, *MASSES, *ELEMENTGROUP, *SECTION, *GENDISP	Part of Model Definition.
5.	Define Floor and Interfloor Instances	*FLOOR, *INTERFLOOR	Part of Model Definition.
6.	Process Output Specification	*RESULTS	

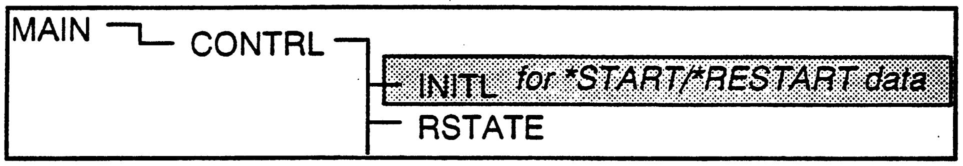
7.	Process Load Patterns and Load Records		
	(a) Static Element Load Patterns	*ELEMLOAD	
	(b) Static Nodal Load Patterns	*NODALOAD	
	(c) Initial Velocity Patterns	*NODALVEL	
	(d) Ground Acceleration Records	*ACCNREC	
	(e) Ground Displacement Records	*DISPREC	
	(f) Dynamic Force Records	*FORCREC	
	(g) Response Spectrum	*SPECTRUM	
8.	Allocate Memory for Analysis Phase		Preparation for analysis phase.
9.	Form Initial Tangent Stiffness and Beta-K Damping		Preparation for analysis phase.
10.	Process Analysis Parameters	*PARAMETERS	
11.	Identify Analysis Type	*GRAV, *STAT, *REST, *MODE, *SPEC, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN, or *FORR	
12.	Set up Loads for Analysis Segments:		
	(a) Static Gravity	*GRAV	
	(b) Static	*STAT	
	(c) Restore to Static State	*REST	
	(d) Ground Acceleration	*ACCN or *ACCR	
	(e) Initial Velocity	*VELN or *VELR	
	(f) Ground Displacement	*DISN or *DISR	
	(g) Dynamic Force	*FORN or *FORR	

13.	Perform Analysis		
	(a) Static Gravity	*GRAV	
	(b) Static	*STAT	
	(c) Restore to Static State	*REST	
	(d) Ground Acceleration	*ACCN or *ACCR	
	(e) Initial Velocity	*VELN or *VELR	
	(f) Ground Displacement	*DISN or *DISR	
	(g) Dynamic Force	*FORN or *FORR	
	(h) Mode Shapes and Periods	*MODE	
	(i) Response Spectrum	*SPEC	
14.	End Session	*STOP	

The phases are described briefly in the following sections. Details can be obtained from the actual code, which is well commented.

7.2. BEGIN SESSION

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In MAIN, open the input data file, DRAIN.INP (unit INP in /TAPES/).
2. In INITL, read the *START/*RESTART data. Set the variables LECHO in /CLINE/; NCHAR in /CONTR/; KDATA, KEXE, KENR and KPDEL in /INDIC/; and FNAME in /TITLE/.
3. In INITL, open the .ECH file (unit IOU) and .OUT file (unit NFOUTP).
4. For *RESTART

a) In CONTRL, open the .SXX file (unit NFPERM), where XX is the restart state number.

b) In RSTATE, restore the state from .SXX file (blank common, all labeled common blocks, INFB blocks, ISTIF blocks and STIF blocks).

5. For *START

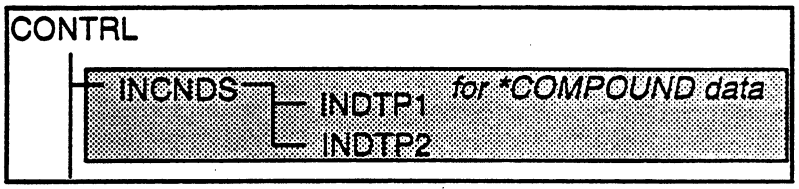
a) In CONTRL, open the .GEO file (unit NFGEO), and write the problem name and problem title to it.

b) In CONTRL, open the .MAS file (unit NFMASS) to store the nodal dead and live load masses for each floor and interfloor type.

c) In CONTRL, open the .TYP file (unit NFFIFT) to store the /INFGR/ and /INFEL/ blocks for each group and element, respectively, for each floor and interfloor type.

7.3. DEFINE COMPOUND NODE TYPES

The flow chart for this phase is as follows.



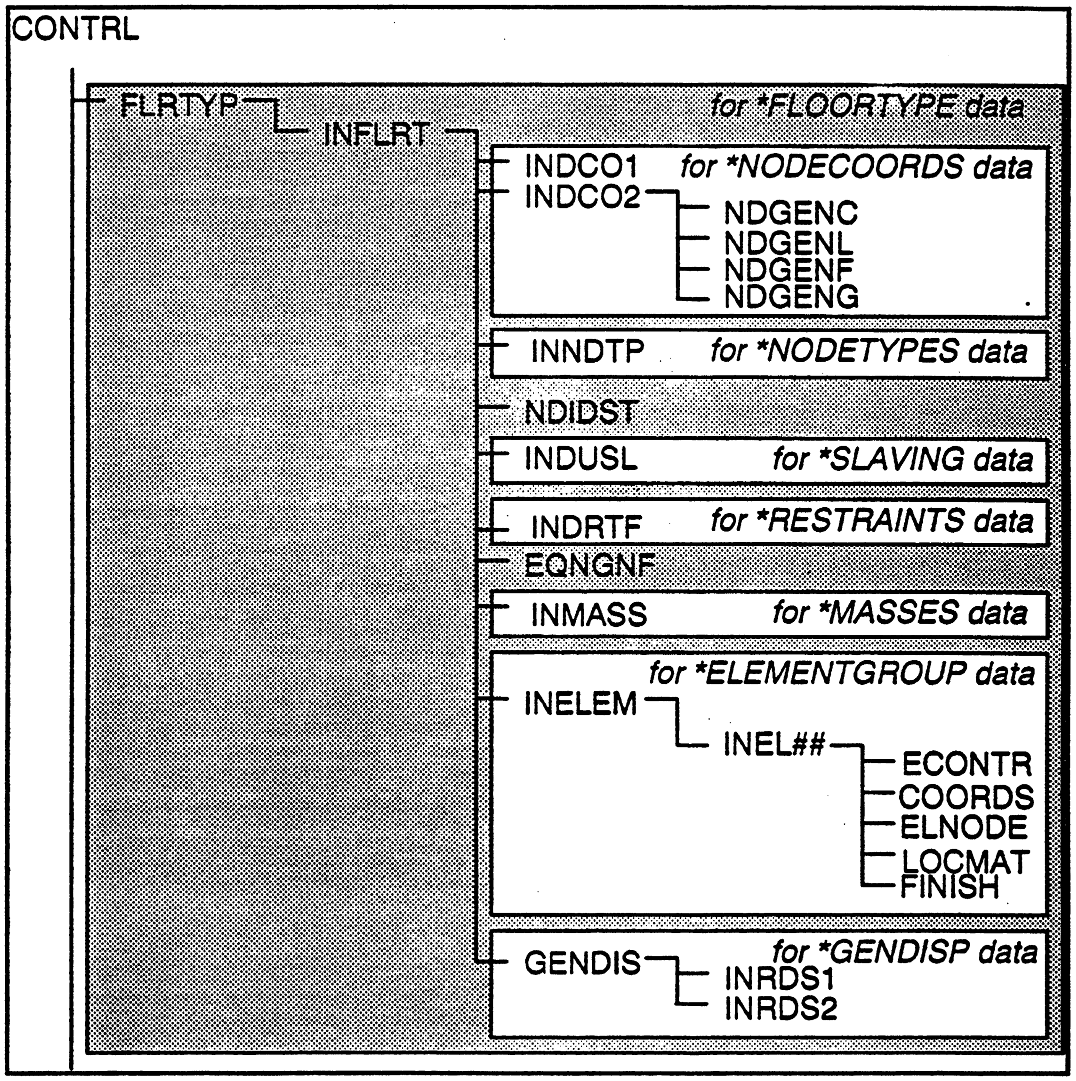
The following main tasks are performed.

1. In INDTP1, read the *COMPOUND data ; count NDTP and NSNDS (see /CONTR/); and write the input lines to INPX.
2. In INCNDS, allocate arrays NSB, NDFSB and COSB (see KNSB, KNDFSB and KCOSB in /STOR2/) in blank common.
3. In INDTP2, re-read the *COMPOUND data from INPX and set up NSB, NDFSB and COSB.

4. In INDTP2, write the data for the compound node types on the .GEO file (unit NFGEO).

7.4. DEFINE FLOOR TYPES

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In FLRTYP, initialize the variables NFLRT and NFIFT in /FLRIFR/ to zero.
2. For each *FLOORTYPE separator, in FLRTYP do the following.
   a) Increment NFLRT and NFIFT.

   b) Read the *FLOORTYPE data; set variables IDFIFT(nfift) and IDFRGT(nflrt) in /TITFIF/; and variable JIRFD(nflrt) and array FLRTDC(1:6,nflrt) in /CNTFIF/.
   c) Write a floor type output record to the .GEO file.
   d) Call INFLRT for reading the rest of the data for the floor type.
3. In INFLRT, for a repeated floor geometry type do the following.
   a) Set variables NNODSF(nfift) and NTNDSF(nfift) in /CNTFIF/ equal to the corresponding values for the parent floor geometry type.
   b) Set first word addresses KNDID(nfift) and KCOORD(nfift) in /STOR3/ equal to the corresponding values for the parent floor geometry type.
4. If not a repeated floor geometry type, then do the following.
   a) In INDCO1, read the *NODECOORDS data; count NNODSF(nfift) (in /CNTFIF/) and write input lines to INPX.
   b) In INFLRT, allocate arrays NDID and COORD for the floor type (i.e., fwa KNDID(nfift) and KCOORD(nfift) in /STOR3/) in blank common.
   c) In INDCO2, re-read the *NODECOORDS data from INPX and set up node numbers in first row of NDID and corresponding nodal coordinates in COORD. INDCO2 calls NDGENC for control nodes, NDGENL for straight line generation, NDGENF for frontal extrapolation, and NDGENG for grid interpolation.
   d) In INFLRT, sort COORD and first row of NDID in increasing order of node numbers.
   e) In INNDTP, read the *NODETYPES data and set up the second row of NDID; and count the number of subnodes in the floor type, NSBT.
   f) In NDIDST, set up the third row of NDID.
   g) In INFLRT, set NTNDSF(nfift) (in /CNTFIF/) equal to NNODS(nfift) + NSBT + 1.

5. In INFLRT, allocate arrays ID and IDSP (i.e., fwa KID(nfift) and KIDSP(nfift) in /STOR3/) in blank common. The actual length NDSP of IDSP is unknown at this stage, and is provisionally set to 6×NNODSF(nfift).
6. In INDUSL, read the *UNSLAVE data and code the ID array as follows.
   a) -NTNDS for slaved DOFs.
7. In INDRTF, read the *RESTRAINTS data and additionally code the ID array as follows.
   b) 0: for a free (i.e., unrestrained) displacement,
   c) 1: for a fixed (i.e., restrained) displacement, and
   d) 2: for a spring supported displacement.
8. In EQNGNF, set up the arrays ID and IDSP; and variables NDSPF(nfift) and NEQF(nfift) in /CNTFIF/.
9. In INFLRT, reduce the allocated space for array IDSP to NDSPF(nfift).
10. In INFLRT, temporarily allocate and initialize to zero the arrays FMND and IELND (see KFMND and KIELND in /STOR1/) in blank common.
11. In INMASS, read the *MASSES data and update the array FMND.
12. In INELEM
    a) Read Group Information data for the *ELEMENTGROUP separators; set the arrays BETAO, KELEM, KEVNT, KGEOM and NELEM in /GENINF/ for the floor type; and set the variables NELGRF(nfift), NELGF(nfift) and NELTTF(nfift) in /CNTFIF/.
    b) Call the element subroutine INEL##, where ## is the element type number, to read the element data.
    c) Write the element group data and IELND array to the .GEO file (unit NFGEO).
13. In INEL##
    a) Call the base program subroutine ECONTR.

b) Read the Element Data for the *ELEMENTGROUP separator.

c) Call the base program subroutine COORDS to get the coordinates of any node.

d) Set the /INFGR/ block for the group.

e) For each element of the group

   i) Call the base program subroutine ELNODE for each element node.

   ii) Call the base program subroutine LOCMAT for each element DOF.

   iii) Set the /INFEL/ block.

   iv) Call the base program subroutine FINISH.

14. In ECONTR, set the variables NEDOF, NENOD, NINFE, NINFEL, NINFG, NINFL, NINFT and NLINF in /GENINF/ for the element group.

15. In ELNODE, set the element nodes in IELNOD and update array FMND due to the mass contribution from the elements.

16. In LOCMAT, code the element LM (Location Matrix) array for the element DOF, as follows.

    n = IFIF + NDISP*3 + NSUB*21 + NLOC*336

    where,

    'n' is the value in the LM array.

    IFIF = 0 for the floor node.

    NDISP = displacement direction (1, 2 or 3 for X, Y and Z translations, respectively; and 4, 5 and 6 for X, Y and Z rotations, respectively).

    NSUB = subnode number (0 = for the main node).

    NLOC = sequence number of the node in NDID array.

    The data is recovered as follows.

    IFIF = MOD(n,3)

    NDISP = MOD(n,21)/3

    NSUB = MOD(n,336)/21

NLOC = n/336

17. In FINISH:

    a) For the first element store the /INFGR/ block on the .TYP file (unit NFFIFT).

    b) For each element store the /INFEL/ block on NFFIFT.

18. In INFLRT, write the array FMND to the .MAS file (unit NFMASS).

19. In INFLRT, set the variable NIADF(nfift) in /CNTFIF/ and allocate array IAD for the floor type (i.e., fwa KIADF(nfift) in /STOR3/) in blank common.

20. For *GENDISP data:

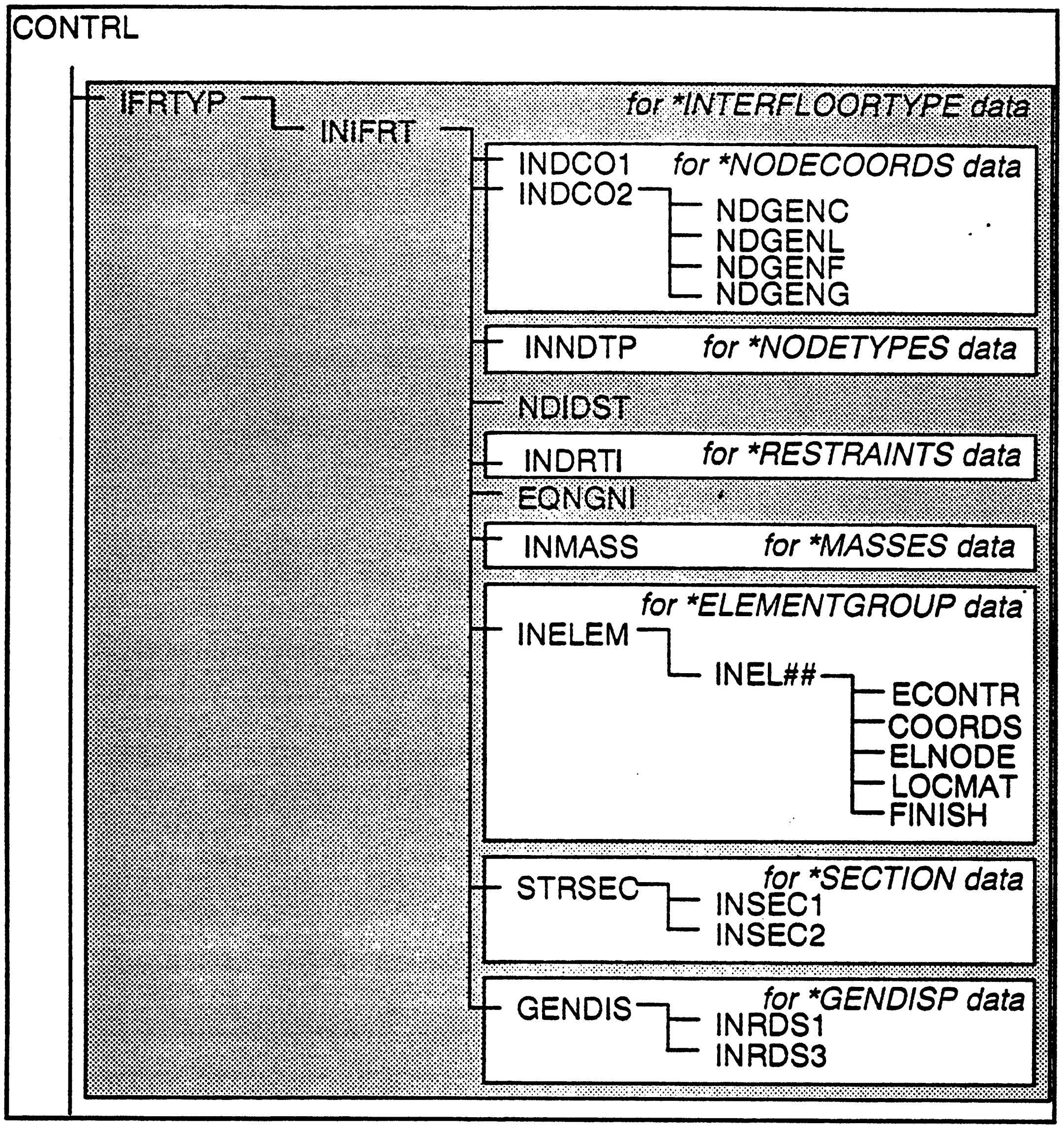
    a) In INRDS1, read the *GENDISP data; count NRDSF(nfift) (in /CNTFIF/); and write the input lines to INPX.

    b) In GENDIS, allocate arrays NODIR and RDFAC (i.e., fwa KNODIR(nfift) and KRDFAC(nfift) in /STOR3/) in blank common.

    c) In INRDS2, re-read the *GENDISP data from INPX and set arrays NODIR and RDFAC.

7.5. DEFINE INTERFLOOR TYPES

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In IFRTYP, initialize the variable NIFRT in /FLRIFR/ to zero.
2. For each *INTERFLOORTYPE separator, in IFRTYP do the following.
   a) Increment NIFRT and NFIFT.
   b) Read the *INTERFLOORTYPE data; set variable IDFIFT(nfift) in /TITFIF/; and arrays JIFRGT(1:2,nifrt) and XYZIFR(1:3,nifrt) in /CNTFIF/.
   c) Write an interfloor type output record to the .GEO file.
   d) Call INIFRT for reading the rest of the data for the interfloor type.

3. *NODECOORDS data:

   a) In INDCO1, read the *NODECOORDS data; count NNODSF(nfift) and write the input lines to INPX.

   b) In INIFRT, allocate arrays NDID and COORD for the interfloor type (i.e., fwa KNDID(nfift) and KCOORD(nfift) in /STOR3/) in blank common.

   c) In INDCO2, re-read the *NODECOORDS data from INPX and set up node numbers in first row of NDID and corresponding nodal coordinates in COORD. INDCO2 calls NDGENC for control nodes, NDGENL for straight line generation, NDGENF for frontal extrapolation, and NDGENG for grid interpolation.

   d) In INIFRT, sort COORD and first row of NDID in increasing order of node numbers.

4. In INNDTP, read the *NODETYPES data; set up the second row of NDID; and count the number of subnodes in the interfloor type, NSBT.
5. In NDIDST, set up the third row of NDID.
6. In INFLRT, set NTNDSF(nfift) (in /CNTFIF/) equal to NNODS(nfift) + NSBT.
7. In INFLRT, allocate arrays ID and IDSP (i.e., fwa KID(nfift) and KIDSP(nfift) in /STOR3/) in blank common. The actual length NDSP of IDSP is unknown at this stage, and is provisionally set to $6 \times$NNODSF(nfift).
8. In INDRTI, read the *RESTRAINTS data and code the ID array as follows.

   b) 0: for a free (i.e., unrestrained) displacement,

   c) 1: for a fixed (i.e., restrained) displacement, and

   d) 2: for a spring supported displacement.

9. In EQNGNI, set the arrays ID and IDSP; and variables NDSPF(nfift) and NEQF(nfift) in /CNTFIF/.
10. In INIFRT, reduce the allocated space for array IDSP to NDSPF(nfift).

11. In INIFRT, temporarily allocate and initialize to zero the arrays FMND and IELND (see KFMND and KIELND in /STOR1/) in blank common.
12. In INMASS, read the *MASSES data and update the array FMND.
13. In INELEM
    a) Read Group Information data for the *ELEMENTGROUP separators; set the arrays BETAO, KELEM, KEVNT, KGEOM and NELEM in /GENINF/ for the interfloor type; and set the variables NELGRF(nfift), NELGF(nfift) and NELTTF(nfift) in /CNTFIF/.
    b) Call the element subroutine INEL##, where ## is the element type number, to read the element data.
    c) Write the element group data and IELND array to the .GEO file (unit NFGEO).
14. In INEL##
    a) Call the base program subroutine ECONTR.
    b) Read the Element Data for the *ELEMENTGROUP separator.
    c) Call the base program subroutine COORDS to get the coordinates of any node.
    d) Set the /INFGR/ block for the group.
    e) For each element of the group
        i) Call the base program subroutine ELNODE for each element node.
        ii) Call the base program subroutine LOCMAT for each element DOF.
        iii) Set the /INFEL/ block.
        iv) Call the base program subroutine FINISH.
15. In ECONTR, set the variables NEDOF, NENOD, NINFE, NINFEL, NINFG, NINFL, NINFT and NLINF in /GENINF/ for the element group.
16. In ELNODE, set the element nodes in IELNOD and update array FMND due to the mass contribution from the elements.

17. In LOCMAT, code the element LM (Location Matrix) array for the element DOF, as follows.

    a) n = IFIF + NDISP*3 + NSUB*21 + NLOC*336

       where,

       'n' is the value in the LM array.

       IFIF = 0 for an interfloor node; 1 for a floor 1 node; and 2 for floor 2 node.

       NDISP = displacement direction (1, 2 or 3 for X, Y and Z translations, respectively; and 4, 5 and 6 for X, Y and Z rotations, respectively).

       NSUB = subnode number (0 = for the main node).

       NLOC = sequence number of the node in NDID array.

       The data can be recovered as follows.

       IFIF = MOD(n,3)

       NDISP = MOD(n,21)/3

       NSUB = MOD(n,336)/21

       NLOC = n/336

18. In FINISH:

    a) For the first element store the /INFGR/ block on the .TYP file (unit NFFIFT).

    b) For each element store the /INFEL/ block on NFFIFT.

19. In INIFRT, write the array FMND to the .MAS file (unit NFMASS).

20. In INIFRT, set the variable NIADF(nfift) in /CNTFIF/ and allocate array IAD for the interfloor type (i.e., fwa KIADF(nfift) in /STOR3/) in blank common.

21. *SECTION data:

    a) In INSEC1, read the *SECTION data; count NSSECF(nifrt) and NTRNS(nifrt); update MXCUTF(nifrt) and MXTDFF(nifrt) (see /CNTFIF/); and write the input lines to INPX.

b) In STRSEC, allocate arrays NECUT, SANG, DIST, IDCUT and STRNS (i.e., fwa KNECUT(nifrt), KDIST(nifrt), KIDCUT(nifrt) and KSTRNS(nifrt) in /STOR3/) for the interfloor type in blank common.

c) In INSEC2, re-read the *SECTION data from INPX and set arrays NECUT, DIST, IDCUT and STRNS.

21. *GENDISP data:

a) In INRDS1, read the *GENDISP data; count NRDSF(nfift) (in /CNTFIF/); and write the input lines to INPX.

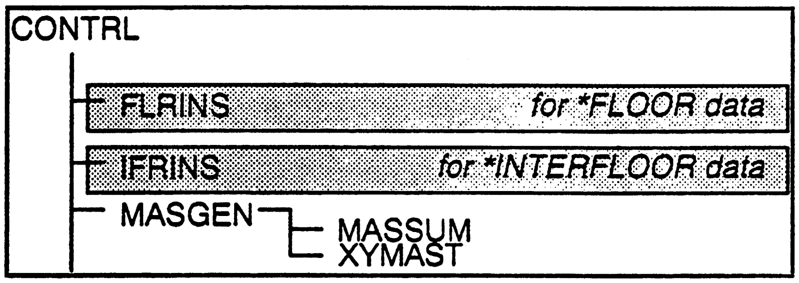
b) In GENDIS, allocate arrays NODIR and RDFAC (i.e., fwa KNODIR(nfift) and KRDFAC(nfift) in /STOR3/) in blank common.

c) In INRDS3, re-read the *GENDISP data from INPX and set arrays NODIR and RDFAC.

22. In CONTRL, close the .TYP and .MAS files (units NFFIFT and NFMASS).

7.6. DEFINE FLOOR AND INTERFLOOR INSTANCES

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In FLRINS, initialize the variables NFLR and NFIF in /FLRIFR/ to zero.

2. For each *FLOOR separator, do the following in FLRINS.

a) Increment NFLR and NFIF.

   b) Read the *FLOOR data, and set variables IDFIF(nfif) in /TITFIF/; JFIFT(nfift), XYZFIF(1:3,nfif) and ALPHAF(1:4,nfif) in /CNTFIF/.

   c) Write the floor instance data to the .GEO file (unit NFGEO).

3. In IFRINS, set the variable NIFR in /FLRIFR/ to zero.

4. For each *INTERFLOOR separator, do the following in IFRINS.

   a) Increment NIFR and NFIF.

   b) Read the *INTERFLOOR data, and set variables IDFIF(nfif) in /TITFIF/; and JFIFT(nfift), JIFR(1:2,nifr), XYZFIF(1:3,nfif) and ALPHAF(1:4,nfif) in /CNTFIF/.

   c) Write the interfloor instance data to the .GEO file (unit NFGEO).

5. In MASGEN, allocate arrays FMNOD (i.e., fwa KFMNOD(1:nfif) in /STOR3/) for floor and interfloor instances in blank common.

6. Do the following for each floor instance.

   a) In MASGEN, initialize the FMNOD array for the floor instance to zero.

   b) In MASGEN, read the dead and live load mass data for the floor type from the .MAS file (unit NFMASS).

   c) In MASSUM, assemble the dead and live load mass contributions in FMNOD.

   d) For each interfloor instance connecting this floor instance do the following.

      i) In MASGEN, read the dead and live load mass data for the corresponding interfloor type.

      ii) In MASSUM, assemble the dead and live load mass contributions from the interfloor to the floor nodes in FMNOD.

   e) In XYMAST calculate the center of mass of the slaved floor nodes and set FMASTC(1:2,nflr) for the floor instance.

7. Do the following for each interfloor instance.

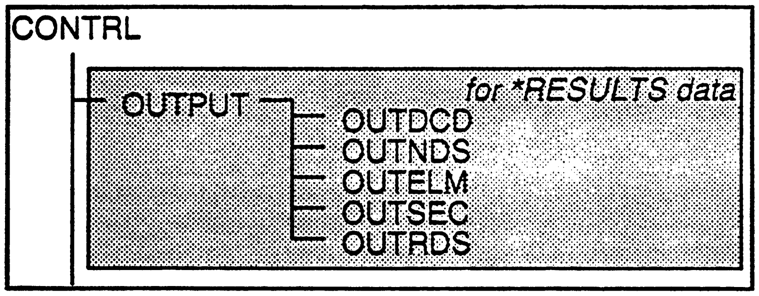
   a) In MASGEN, initialize the FMNOD array for the interfloor instance to zero.

b) In MASGEN, read the dead and live load mass data for the interfloor type from the .MAS file (unit NFMASS).

c) In MASSUM, assemble the dead and live load mass contributions in FMNOD.

7.7. PROCESS OUTPUT SPECIFICATION

The flow chart for this phase is as follows.



The following main tasks are performed.

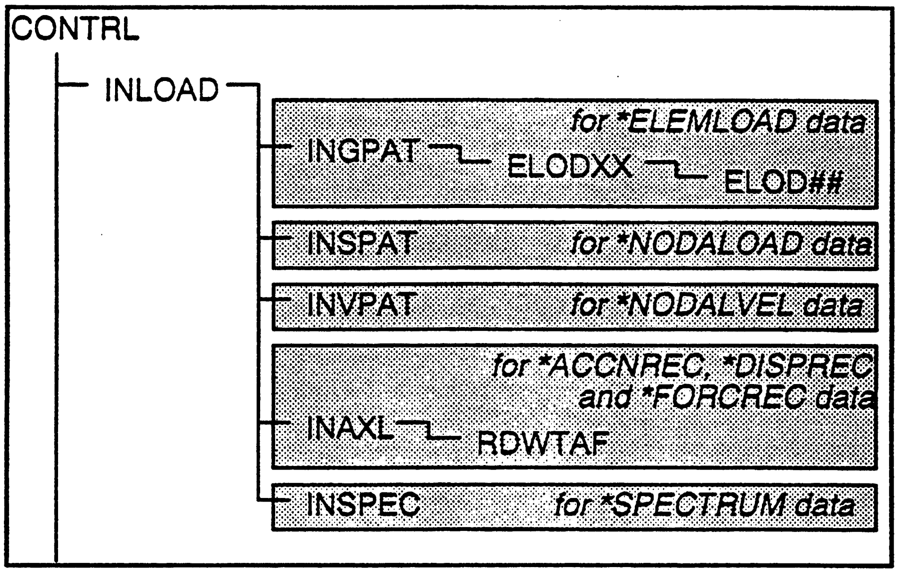
1. For *START, in OUPUT
   a) Set the variable LREC in /RHIST/.
   b) Allocate arrays JNOD, JELM, JSEC and JRDS (see KJNOD, KJELM, KJSEC and KJRDS in /STOR3/) for each floor and interfloor instance in blank common.
   c) Initialize arrays JNOD, JELM, JSEC and JRDS with the default output codes.
   d) Initialize the arrays NDPOUT, NELTH, NPSEC, NNRDS, NDPOST, NELTHP, NPSECP and NNRDSP in /RHIST/ corresponding to the default output codes.
2. In OUTPUT, read the *RESULTS data and call
   a) OUTDCD to update JFLR in /CNTFIF/ for each affected floor instance.
   b) OUTNDS to update JNOD, NDPOUT and NDPOST for each affected floor or interfloor instance.
   c) OUTELM to update JELM, NELTH and NELTHP for each affected floor or interfloor instance.

d) OUTSEC to update JSEC, NPSEC and NPSECP for each affected interfloor instance.

e) OUTRDS to update JRDS, NNRDS and NNRDSP for each affected floor or interfloor instance.

7.8. PROCESS LOAD PATTERNS AND LOAD RECORDS

The flow chart for this phase is as follows.



The following main tasks are performed.

1. In INLOAD, allocate work space for reading the load patterns and dynamic load records. For *RESTART, use the array WKSPC (see KWKSP in /STOR1/) in blank common. For *START, use all remaining blank common.
2. In INLOAD, read the separator line and for:
   a) *ELEMLOAD - call INGPAT to read the *ELEMLOAD data and write the static element load pattern to the .ELD file (see .ELD file in Chapter 8).
   b) *NODALOAD - call INSPAT to read the *NODALOAD data and write the static nodal load pattern to the .STA file (see .STA file in Chapter 8).

c) *NODALVEL - call INVPAT to read the *NODALVEL data and write the initial velocity pattern to the .VEL file (see .VEL file in Chapter 8).

d) *ACCNREC, *DISPREC or *FORCREC - call INAXL to read the *ACCNREC, *DISPREC or *FORCREC data and write the load record to the .ACC, .DIS or .FRC file (see these files in Chapter 8).

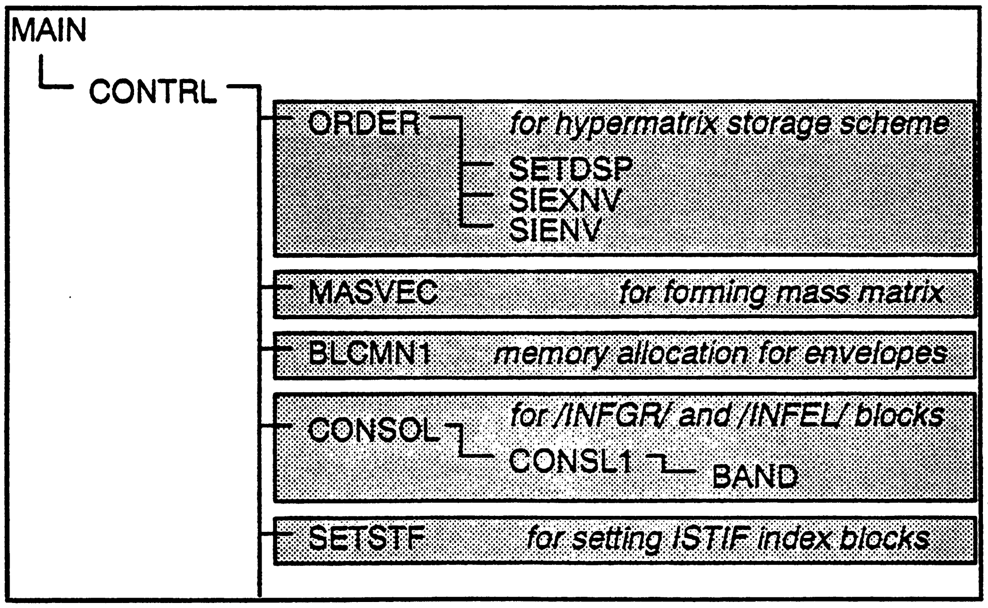
e) *SPECTRUM - call INSPEC to read the *SPECTRUM data and write the response spectrum to the .SPC file (see .SPC file in Chapter 8).

The processing of data for *ACCNREC, *DISPREC, *FORCREC and *SPECTRUM separators is identical with that in DRAIN-2DX and 3DX.

In DRAIN-BUILDING, the *ELEMLOAD, *NODALOAD and *NODALVEL data is specified for a floor or an interfloor type. The pattern name plus the ID of the floor or interfloor type must be unique. The processing of the data is otherwise very similar to that in DRAIN-2DX and 3DX.

7.9. ALLOCATE MEMORY FOR ANALYSIS PHASE

The flow-chart for this phase is as follows.


MAIN
CONTRL
ORDER
for hypermatrix storage scheme
SETDSP
SIEXNV
SIENV
MASVEC
for forming mass matrix
BLCMN1
memory allocation for envelopes
CONSOL
for /INFGR/ and /INFEL/ blocks
CONSL1
BAND
SETSTF
for setting ISTIF index blocks

The following main tasks are performed.

1. In ORDER, order the floor and interfloor instances for the hypermatrix and set arrays JFIF in /CNTFIF/ and IDORDR in /TITFIF/.
2. In ORDER, set the array NEQOF in /CNTFIF/; and variables NEQT, NEQQT, NEQIMX, NEQFMX and NEQQMX in /EQNS/.
3. In ORDER, set variable NDSPT (in /CONTR/) and allocate arrays IDSPT and SPDSP (see KIDSPT and KSPDSP in /STOR4/) in blank common.
4. In SETDSP, set the array IDSPT.
5. In ORDER, allocate arrays IADOF and IEXNV (see KIADOF and KIEXNV in /STOR4/) in blank common.
6. In SIEXNV, set the array IEXNV and set variable NZERO (in /FLRIFR/).
7. In ORDER, allocate array IENV (see KIENV in /STOR4/) in blank common.
8. In SIENV, set the array IENV and variable NNZERO (in /FLRIFR/).
9. In ORDER, set variables NSTFD and NSTFO in /EQNS/.
10. In MASVEC, allocate and set the arrays FMDOF and ALPHAM (see KFMDOF and KALPHA in /STOR4/) in blank common.
11. In BLCMN1, allocate the envelope arrays DENP, ISTP, DENN, ISTN, SECENV, ISECEN, RDSENV and IRDSEN for each floor and interfloor instance (see /STOR5/) in blank common.
12. In CONTRL, allocate array WKSPC (see KWKSPC in /STOR6/) in blank common. WKSPC is used for storing the following.
    a) Load pattern and dynamic load record input for *RESTART.
    b). Applied load patterns when setting up loads for analysis.
    c) Array MSTIF(neqqt) storing the row numbers of the first nonzero terms for columns of the stiffness matrix in subroutines CONSOL and SETSTF.

c) Element stiffness matrix during tangent stiffness update.

d) Temporary vector of length NEQQT for calculation of midstep equilibrium error or equivalent impulse error.

e) Nodal displacements DISP(6,ntndsf(ift)) for the current floor or interfloor instance for output during analysis and for updating nodal displacement envelopes.

f) Section forces SEFOR (see KSEFOR in /STOR6/) for the current floor or interfloor instance for output during analysis and for updating section envelopes.

g) Generalized displacements, RDS (see KRDS in /STOR6/) for the current floor or interfloor instance for output during analysis and for updating generalized displacement envelopes.

13. In CONTRL, set KSEFOR and KRDS (see /STOR6/) so that SEFOR and RDS may safely use parts of the WKSPC array.

14. In CONTRL, allocate the following arrays in blank common (see first word addresses from KREC to KDRU in /STOR6/).

    a) output buffer REC.

    b) vectors XLOD, EXTS, EXT, DEXT, RINTS, RINT, RU, DISI, VELI, ACCI, DDIS, CVEL, CACC, DIS, VEL, ACC, ENRI and DRU.

15. In CONTRL, calculate blank common available, NAVST, for storing /INFGR/ and /INFEL/ data. NAVST is the provisional length of INFB (see KINFB in /STOR6/).

16. In CONSOL, initialize the temporary array MSTIF(neqqt) so that MSTIF(i) = I for I=1,NEQQT. MSTIF is updated in subroutine BAND, so that it finally stores the row numbers of the first nonzero terms for columns of the stiffness matrix (see Section 5.6 for use of MSTIF).

17. In CONSOL, open scratch file NFBEG; and call CONSL1 for each floor and interfloor instance.

18. In CONSL1, for the instance read the /INFGR/ and /INFEL/ blocks from the .TYP file (unit NFFIFT) in to INFB; set the variable NBLOKF in /CNTFIF/; set the array IAD for the floor or interfloor type; set the variable IADOF (see KIADOF in /STOR4/); and call BAND for each element.
19. In BAND, update the MSTIF array, assuming that all element DOFs are coupled.
20. In BAND, re-code the element LM array for the element DOF, as follows (see Section 5.7).

    a) For an independent DOF or rotational slaved DOF:

       n = IFIF + LEQ*3

       where,

          'n' is the value in the LM array.

          IFIF = 0 if the DOF belongs to the floor or interfloor instance,

          IFIF = 1 if the DOF belongs to the floor 1 of the interfloor, and

          IFIF = 2 if the DOF belongs to the floor 2 of the interfloor.

          LEQ = local equation number for the DOF or for the rotational DOF of the master node.

       The data is recovered as follows.

          IFIF = MOD(n,3)

          LEQ = n/3

    b) For a translational slaved DOF:

       -n = IFIF + NDISP*3 + NLOC*9

       where,

          '-n' is the value in the LM array.

          IFIF = 0, 1 or 2 as for (a).

          NDISP = 1 for translational X or 2 for translational Y DOF.

          NLOC = sequence number of the slaved node.

The data is recovered as follows.

IFIF = MOD(n,3)

NDISP = MOD(n,9)/3

NLOC = n/9

21. In CONSOL, set the variable NTBLOK in /STOR1/.
22. In CONSOL, set the actual length, NINFB, of INFB as follows.
    a) NINFB=NAVST if NTBLOK > 1.
    b) NINFB=LTSING if NTBLOK = 1, where LTSING is the length required to accommodate /INFGR/ and /INFEL/ blocks for all elements.
23. In CONSOL, calculate LENK (in /EQNS/) using array MSTIF.
24. In CONSOL, set the variable INCOR in /CONTR/ depending on the remaining memory in blank common and requirements for storing stiffnesses.
25. In CONSOL, if INCOR $\leq$ 0 allocate arrays DIAG, STIF1-6 and ISTIF1-6 (see /STOR7/) in blank common.
26. In CONSOL, if INCOR $\geq$ 1 allocate arrays TANK, EFFK and LSTIF (see /STOR8/) in blank common.
27. In CONSOL, if INCOR $\geq$ 3 allocate array BETAK (see /STOR8/) in blank common.
28. In CONSOL, if INCOR $\geq$ 4 allocate array DTAN (see /STOR8/) in blank common.
29. In CONSOL, if INCOR = 5 allocate array DINFB (see KDINFB in /STOR6/) and close the scratch file NFBEG.
30. In CONSOL, set the variables NOFEK, NOFTK, NOFBK and NOFDK in /EQNS/ depending on value of INCOR (see Section 5.6.4).
31. In CONTRL, open scratch files NFCUR and NFUPD if NTBLOK $\neq$ 1 for storing /INFGR/ and /INFEL/ blocks.
32. In SETSTF, if INCOR $\leq$ 0

a) Open the scratch file NFSTFA with record length NEQQMX (in /EQNS/) in 4-byte units to store the address blocks, ISTIF, of the hypermatrix.

b) Open the scratch file NFSTFB with record length NSTFO (in /EQNS/) in 8-byte units to store the data blocks, STIF, of the hypermatrix.

c) For each stiffness block in the hypermatrix, set the ISTIF array (by using the data in MSTIF array) and write ISTIF to NFSTFA.

d) Initialize array STIF to zero, and write dummy records to NFSTFB corresponding to each stiffness block.

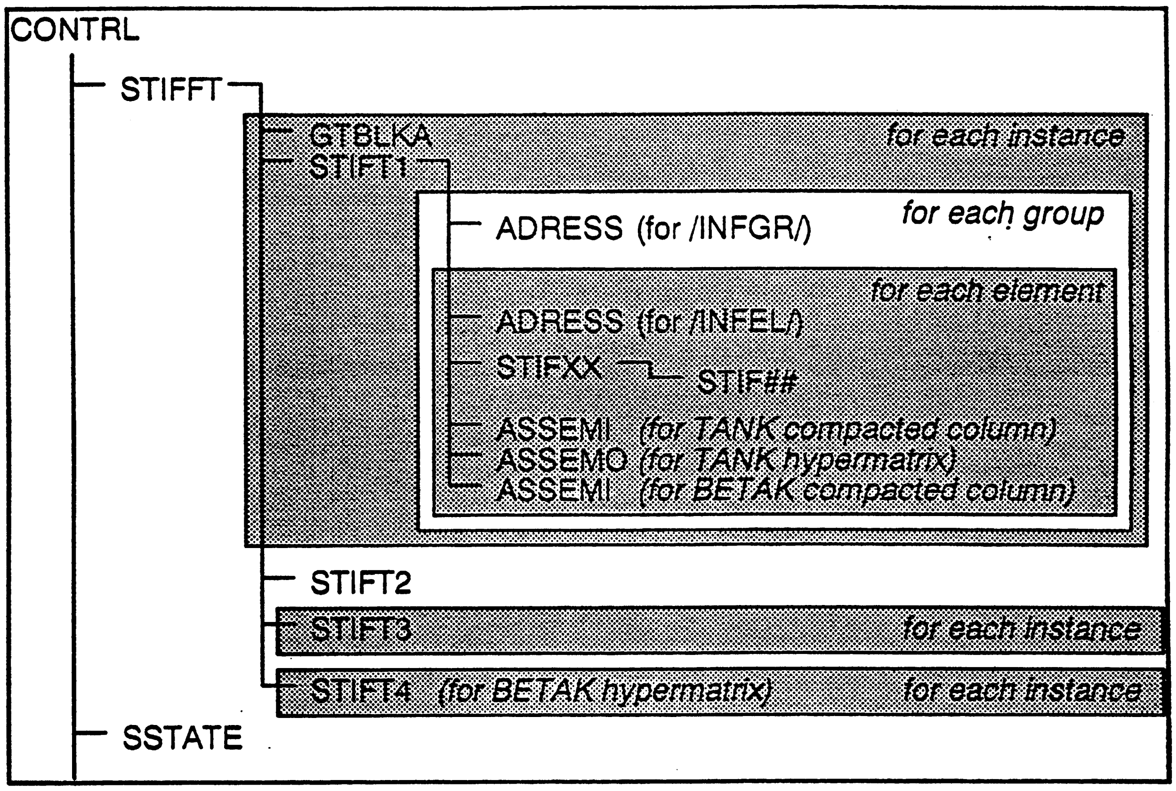
33. In SETSTF, if INCOR ≥ 1

a) Open the scratch file NFSTFB if INCOR ≤ 3 with a record length equal to LENK (in /EQNS/) in 8-byte units to store the compacted column stiffnesses and/or beta-k damping.

b) Set the LSTIF array (by using the data in MSTIF array).

7.10. FORM INITIAL TANGENT STIFFNESS AND BETA-K DAMPING

The flow chart for this phase is as follows.



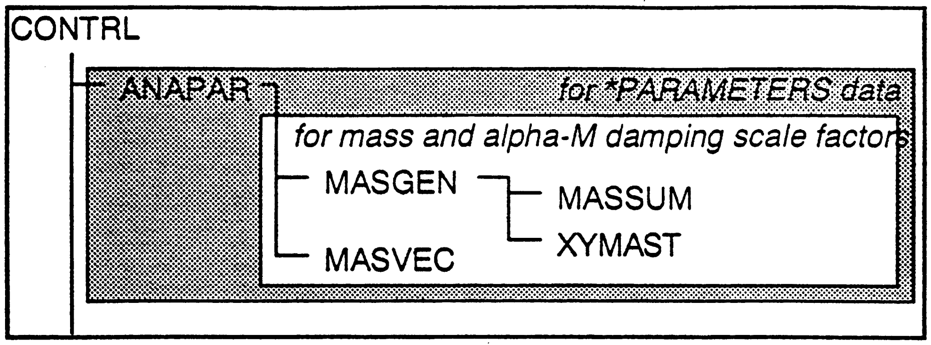
The following main tasks are performed.

1. In CONTRL, call STIFFT after detecting the first analysis separator (*GRAV, *STAT, *REST, *ACCN, *DISN, *FORN, *MODE or *SPEC).
2. In STIFFT, to form TANK and BETAK, for each instance
   a) If INCOR ≤ 0, call GTBLKA to get the addresses of the required stiffness blocks of TANK hypermatrix (see Section 5.7).
   b) Call STIFT1 for each instance.
3. In ADRESS, restore the /INFGR/ or /INFEL/ block from INFB. If required block is not in INFB, then write INFB to NFUPD and refill INFB from NFCUR.
4. In STIFXX call the element subroutine STIF##, where ## is the element type number for the current group (see KELEM in /GENINF/).
5. In STIF##, form the element stiffness matrix.

6. If INCOR > 0, then in ASSEMI, assemble the element stiffness into TANK (see Section 5.10.
7. If INCOR ≤ 0, then in ASSEMO, assemble the element stiffness into TANK hypermatrix (see KTANK in /STOR8/ and Section 5.7).
8. In STIFT1, scale element stiffness by BETAO (in /GENINF/) factor for the current element group to obtain the element beta-K damping matrix.
9. If INCOR > 0, then in ASSEMI, assemble the element beta-K damping into BETAK (see KBETAK in /STOR8/).
10. If INCOR ≤ 0, then in STIFT1 write the element beta-K damping and LM matrices to the scratch file, INPX.
11. If INCOR > 0, then in STIFT2 assemble the stiffness contributions of the support springs.
12. If INCOR ≤ 0, then in STIFFT call STIFT3 for each instance; and in STIFT3 assemble the stiffness contributions of the support springs.
13. In STIFFT, verify that TANK is not singular.
14. If INCOR ≤ 0, then in STIFFT call STIFT4 for each instance; in STIFT4, read the element beta-K damping and LM matrices from INPX; and in ASSEMO, assemble element beta-K in to the BETAK hypermatrix.
15. In CONTRL, open the .SXX file (unit NFPERM), where XX is the segment number 0.
16. In SSTATE, save on the .SXX file the blank common block; labeled common blocks; INFB blocks from NFBEG; and stiffness blocks from the NFSTFA and NFSTFB files.

7.11. PROCESS ANALYSIS PARAMETERS

The flow chart for this phase is as follows.



ANAPAR reads the *PARAMETERS data and modifies the variables and arrays in labeled common blocks, as follows.

Type of Analysis Parameters	**Modified Variables or Arrays**
Mass and Alpha-M Damping Scale Factors	ALPHAF in /CNTFIF/; FMNOD, FMDOF and ALPHAM in blank common (see KFMNOD in /STOR3/; KFMDOF and KALPHA in /STOR4/).
Viscous Damping Scale Factors	ALPHA and BETA in /DAMPG/
Collapse Displacements	DISMAX and RTNMAX in /COLPSE/
Event Overshoot Scale Factors	OVFAC in /GENINF/
Element Parameters	IELPR and RELPR in /GENINF/
Output Intervals for Static Analysis	all variables in /OUTS/
Output Intervals for Dynamic Analysis	all variables in /OUTD/
Control Parameters for Dynamic Analysis	KEVEN, KENRC, KEQBC and MAXEVD in /INDIC/
Time Step Parameters for Dynamic Analysis	DTAUTO, DTCONS, DTMAX and DTMIN in /AUTO/
Parameters for Variable Time Step Scheme	DTRED, DTINC, TOLHII, TOLHIS, TOLLOI, TOLLOS, TOLMX and NSINC in /AUTO/

7.12. IDENTIFY ANALYSIS TYPE

The flow chart for this phase is as follows.

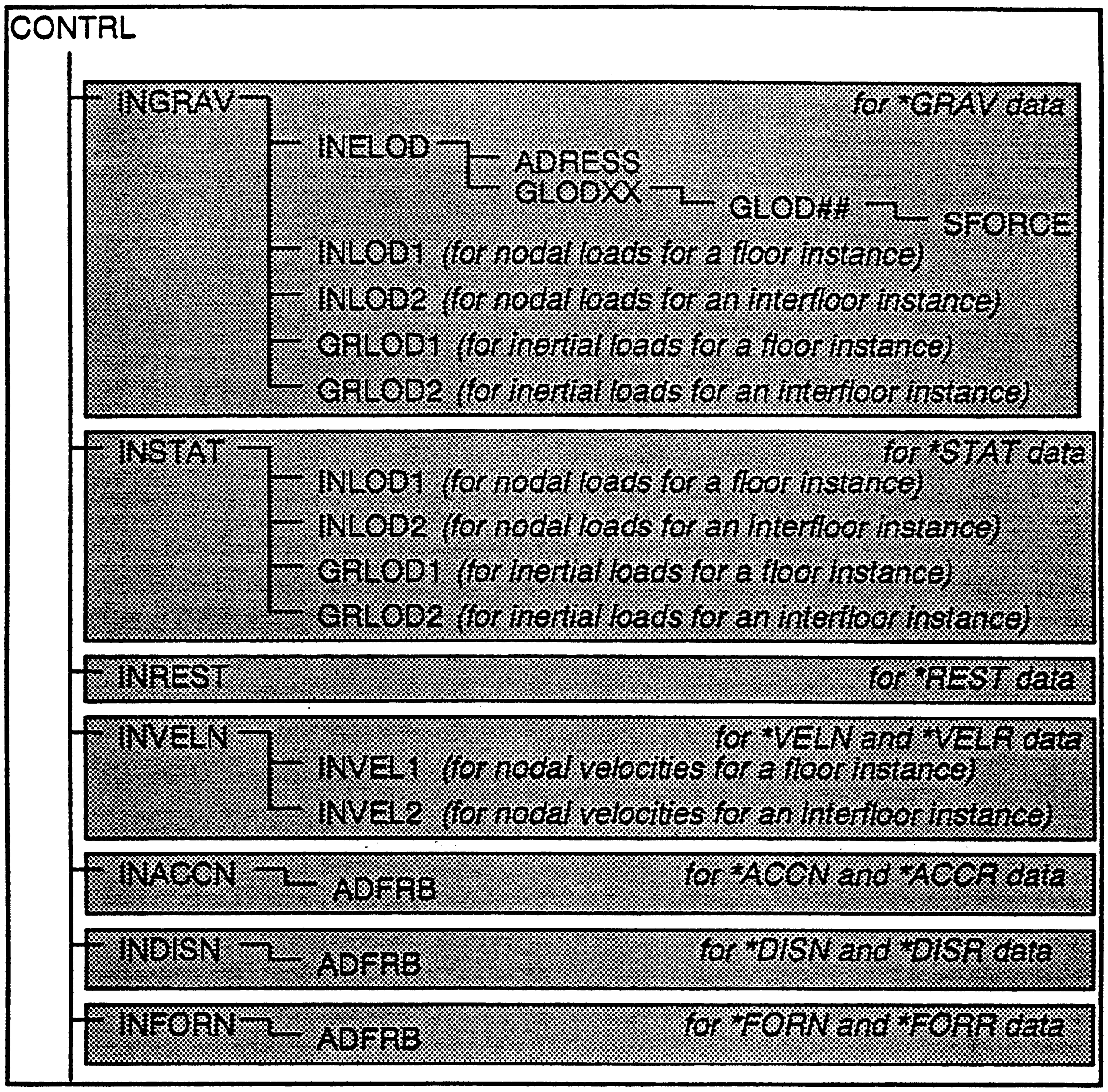
```
MAIN
 ├─ CONTRL
```

The following main tasks are performed in CONTRL.

1. Check that the separator is valid (i.e., one of *GRAV, *STAT, *REST, *MODE, *SPEC, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN, *FORR).
2. For *GRAV, check that the structure is in the unstressed state or the preceding analysis was also *GRAV.
3. For *STAT, *ACCN, *VELN, *DISN and *FORN, check that the structure is in static state (i.e., preceding analysis was *GRAV, *STAT or *REST).
4. For *ACCR check that the preceding analysis was *ACCN or *ACCR.
5. For *VELR check that the preceding analysis was *VELN or *VELR.
6. For *DISR check that the preceding analysis was *DISN or *DISR.
7. For *FORR check that the preceding analysis was *FORN or *FORR.

7.13. SET UP LOADS FOR ANALYSIS SEGMENTS

The flow chart for this phase is as follows.


CONTRL
INGRAV
for *GRAV data
INELOD
ADRESS
GLODXX
GLOD##
SFORCE
INLOD1 (for nodal loads for a floor instance)
INLOD2 (for nodal loads for an interfloor instance)
GRLOD1 (for inertial loads for a floor instance)
GRLOD2 (for inertial loads for an interfloor instance)
INSTAT
for *STAT data
INLOD1 (for nodal loads for a floor instance)
INLOD2 (for nodal loads for an interfloor instance)
GRLOD1 (for inertial loads for a floor instance)
GRLOD2 (for inertial loads for an interfloor instance)
INREST
for *REST data
INVELN
for *VELN and *VELR data
INVEL1 (for nodal velocities for a floor instance)
INVEL2 (for nodal velocities for an interfloor instance)
INACCN
ADFRB
for *ACCN and *ACCR data
INDISN
ADFRB
for *DISN and *DISR data
INFORN
ADFRB
for *FORN and *FORR data

The following main task are performed.

1. In CONTRL, call INGRAV for *GRAV; INSTAT for *STAT; INREST for *REST; INACCN for *ACCN or *ACCR; INVELN for *VELN or *VELR; INDISN for *DISN or *DISR; or INFORN for *FORN or *FORR.
2. In INGRAV, read the *GRAV data.
3. In INSTAT, read the *STAT data.
4. In INREST, read the *REST data.
5. In INACCN, read the *ACCN or *ACCR data.

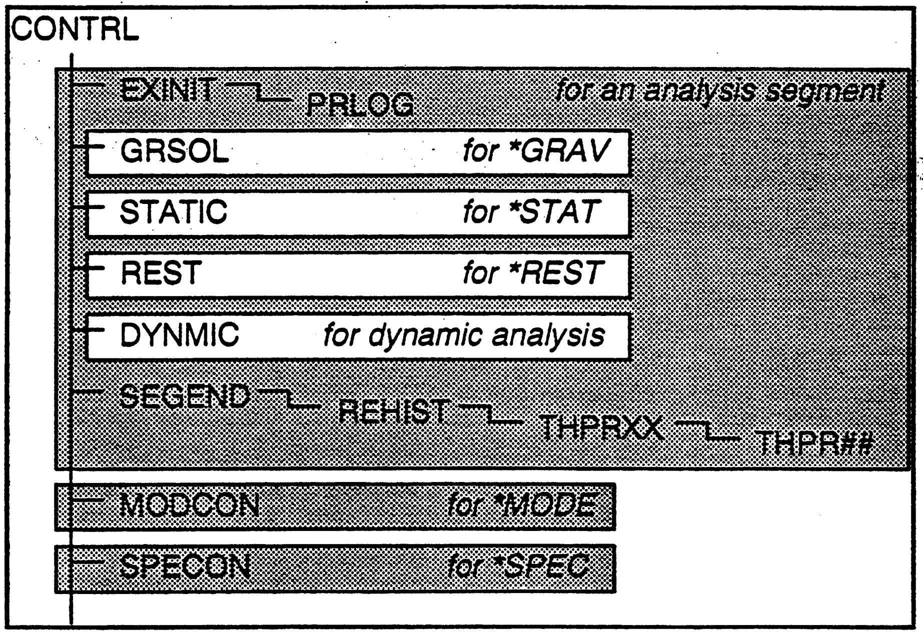
6. In INVELN, read the *VELN or *VELR data.
7. In INDISN, read the *DISN or *DISR data.
8. In INFORN, read the *FORN or *FORR data.

For DRAIN-BUILDING, the tasks performed in subroutines INGRAV, INSTAT, INREST, INVELN, INACCN, INDISN and INFORN are similar to those done in these subroutines for DRAIN-2DX and 3DX.

7.14. PERFORM ANALYSIS

7.14.1. Main Tasks

The flow chart for this phase is as follows.



The following main task are performed.

1. In CONTRL, to perform a *GRAV, *STAT, *REST, *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN or *FORR analysis
    a) Call EXINIT.
    b) Call GRSOL for *GRAV.

   c) Call STATIC for *STAT.
   d) Call REST for *REST.
   e) Call DYNMIC for *ACCN, *ACCR, *VELN, *VELR, *DISN, *DISR, *FORN or *FORR.
   f) call SEGEND.

2. In CONTRL, to perform *MODE analysis:
   a) Store blank common from L(KENP(1)) to L(KDIAG-1) for hypermatrix storage scheme and from L(KENP(1)) to L(KEFFK-1) for compacted column storage scheme on scratch file, NFRES. This makes space available for storing mode shapes, a flexibility matrix, and other data. EFFK, TANK and some arrays before DENP in blank common are used in *MODE analysis.
   b) Call MODCON to perform the analysis.
   c) On completion of the analysis restore blank common from NFRES.
3. In CONTRL, to perform *SPEC analysis:
   a) Store blank common from L(KENP(1)) to L(KINFB-1) on scratch file, NFRES. This makes space available for *SPEC analysis for storing response results, and other data. INFB and some arrays before DENP in blank common are used in the analysis.
   b) Call SPECON to perform the analysis.
   c) On completion of the analysis, restore blank common from NFRES.
4. In EXINIT:
   a) Initialize variables KENR, KRESIS and KTIT in /INDIC/; all variables in /OUTP/; NTIME and NPTIME in /RHIST/; and IQUIT and NEVEN in /EVENT/.
   b) Increment the analysis segment number, NSEG (in /CONTR/).
   c) Write the heading for the new segment on the .OUT file (unit NFOUTP in /TAPES/).

d) Call PRLOG to write the heading and starting energy log for the segment on the .SLO file (unit NFLOG). The heading and starting energy log for *REST is not written here, but later in REST.

e) Update the .LST file.

f) Open and write initial data to .RXX file (unit NFRES), where XX is the analysis segment number.

g) Open and write the initial data to .EXX file (unit NFENVP), where XX is the analysis segment number.

h) Open scratch file, unit NFPRNT, to save time-history results for printout.

i) Write analysis title on the .ECH file.

5. In GRSOL, STATIC and REST perform *GRAV, *STAT and *REST analysis, respectively, and write solution log for each substep to .ECH file (unit IOU); write energy log for each substep to .SLO file (unit NFLOG); write load-history results for printout to unit NFPRNT; write load-history results for post-processing to .RXX file (unit NFRES); write envelope results for printout to .OUT file (unit NFOUTP); write static and P-Δ work done for each element group (see ENEG and EEXT in /ENRGY/) to .OUT file (unit NFOUTP); write envelope results for post-processing to .EXX file (unit NFENVP); and write structure state data to .SXX file (unit NFPERM).

6. In DYNMIC perform the given dynamic analysis, and write solution log for each substep to .ECH file (unit IOU); write energy log for each time step to .SLO file (unit NFLOG); write time-history results for printout to unit NFPRNT; write time-history results for post-processing to .RXX file (unit NFRES); write envelope results for printout to .OUT file (unit NFOUTP); write static, damping and P-Δ work done for each element group (see ENEG, ENRD and EEXT in /ENRGY/) to .OUT file (unit

NFOUTP); write envelope results for post-processing to .EXX file (unit NFENVP); and write structure state data to .SXX file (unit NFPERM).

7. In SEGEND, close the .RXX and .EXX files.
8. In REHIST, read results for printout from unit NFPRNT, and write the results for each output item (see /RHIST/) to .OUT file (unit NFOUTP). For element results output, REHIST copies the element results to /THELM/ block and calls THPRXX.
9. In THPRXX, call the element subroutine THPR##, where ## is the element type number for the element (see KELEM in /GENINF/).
10. In THPR##, write heading and element results from /THELM/ to the .OUT file.
11. In MODCON read the *MODE data; perform *MODE analysis; and write periods, mass participation factors and mode shapes to .OUT file (unit NFOUTP) and .MXX file (unit NFMODE), where XX is the analysis segment number.
12. In SPECON read the *SPEC data; read specified response spectra from .SPC file; read periods and mass participation factors from .MXX file (unit NFMODE) and write to .UXX file (unit NFMRSL); read mode shapes from .MXX file; for each mode shape, calculate nodal displacements, element results, section forces and generalized displacements and write to .UXX file; calculate modal amplitudes; write response for each mode to .OUT file; form SRSS combination and write to .OUT file.
13. On completion of the analysis return to CONTRL and read the next separator line.
    a) For *PARAMETERS proceed to Process Analysis Parameters.
    b) For *STOP proceed to End Analysis Session.
    c) For an analysis separator proceed to Identify Analysis Type.

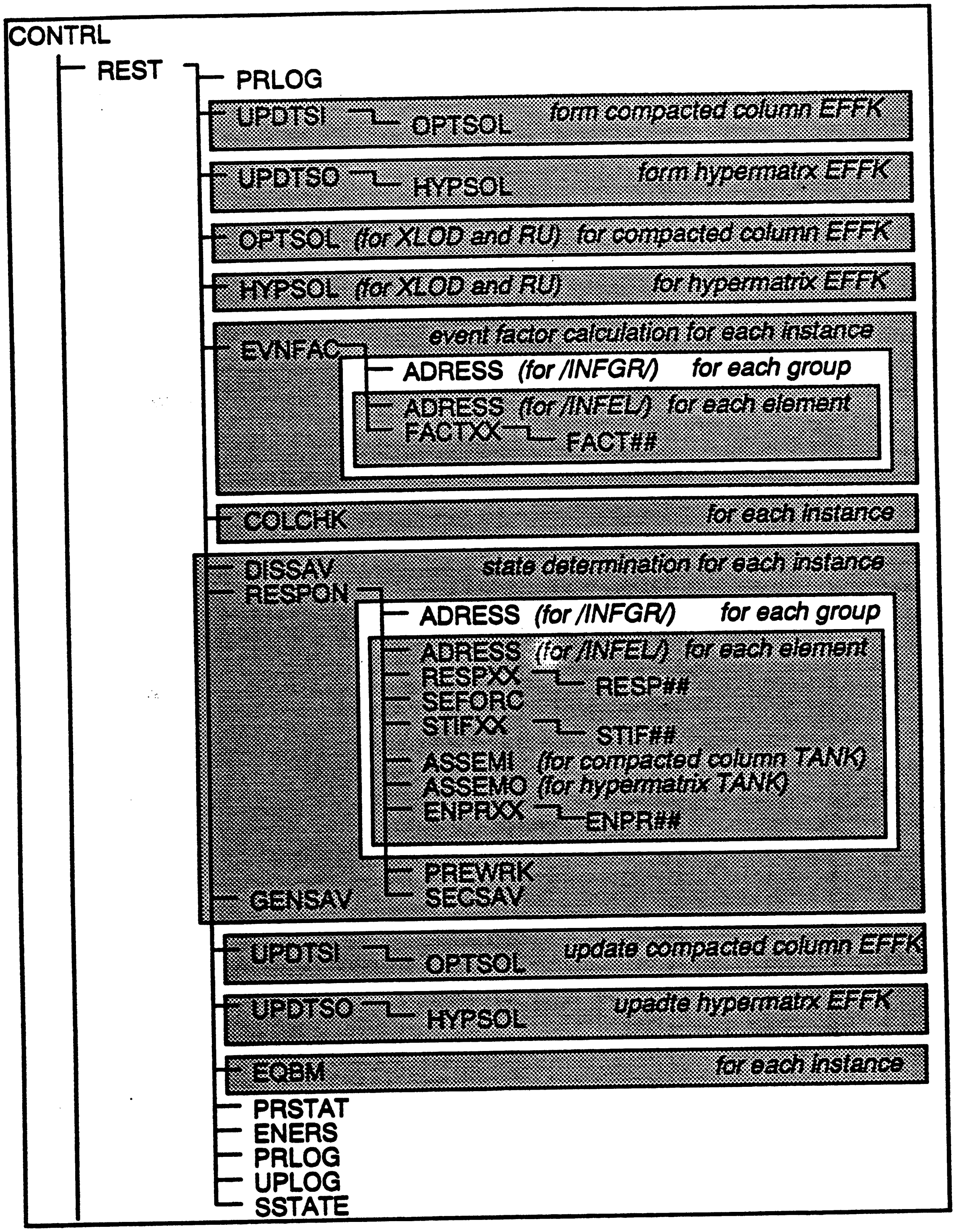
In DRAIN-BUILDING, the tasks done in GRSOL, STAT, REST, DYNMIC, MODCON and SPECON are essentially identical to those done for programs DRAIN-2DX and 3DX.

More detailed flow charts for these subroutines are presented in the following sections. Note the following differences in DRAIN-BUILDING as compared to DRAIN-2DX and 3DX.

1. For any element task (e.g., event factor calculation, state determination, stiffness update etc.) the elements are processed one floor or interfloor instance at a time. Thus subroutines EVNFAC and RESPON are called once for each instance.
2. In DRAIN-BUILDING, many nodal tasks are also carried out for one floor or interfloor instance at a time. Thus subroutines COLCHK, DISSAV, VELSAV, GENSAV and EQBM are called once for each instance.
3. For compacted column stiffnesses, subroutines UPDTSI, UPDTDI, OPTSOL, ASSEMI and MULTCO are used. These are similar or identical to subroutines UPDATS, UPDATE, OPTSOL, ASSEM and MULTCO.
4. For stiffnesses hypermatrices, subroutines UPDTSO, UPDTDO, HYPSOL, ASSEMO and HYPMUL are used.

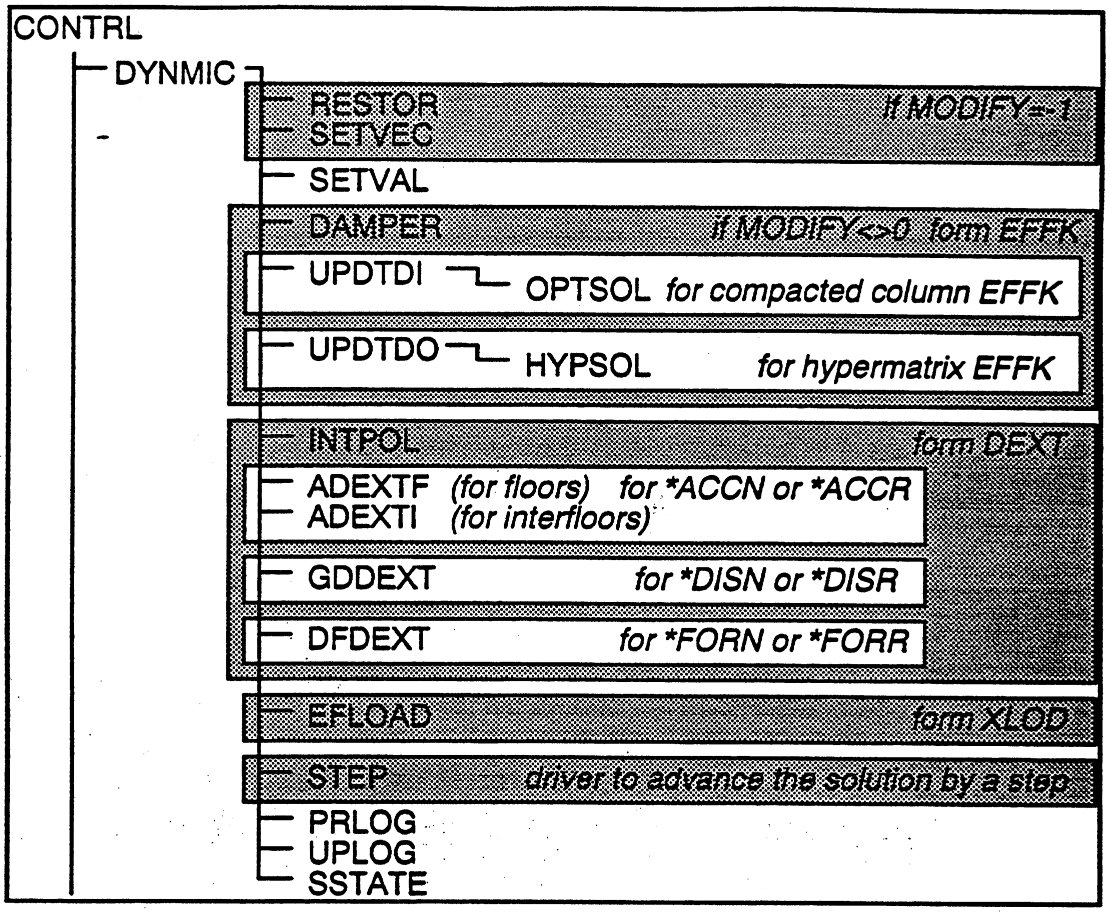
7.14.2. *GRAV Analysis - Subroutine GRSOL

The flow chart for this subroutine is as follows.


CONTRL
REST
PRLOG
UPDTSI
OPTSOL
form compacted column EFFK
UPDTSO
HYPSOL
form hypermatrx EFFK
OPTSOL (for XLOD and RU) for compacted column EFFK
HYPSOL (for XLOD and RU) for hypermatrix EFFK
EVNFAC
event factor calculation for each instance
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/) for each element
FACTXX
FACT##
COLCHK
for each instance
DISSAV
state determination for each instance
RESPON
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/) for each element
RESPXX
RESP##
SEFORC
STIFXX
STIF##
ASSEMI (for compacted column TANK)
ASSEMO (for hypermatrix TANK)
ENPRXX
ENPR##
PREWRK
SECSAV
GENSAV
UPDTSI
OPTSOL
update compacted column EFFK
UPDTSO
HYPSOL
upadte hypermatrx EFFK
EQBM
for each instance
PRSTAT
ENERS
PRLOG
UPLOG
SSTATE

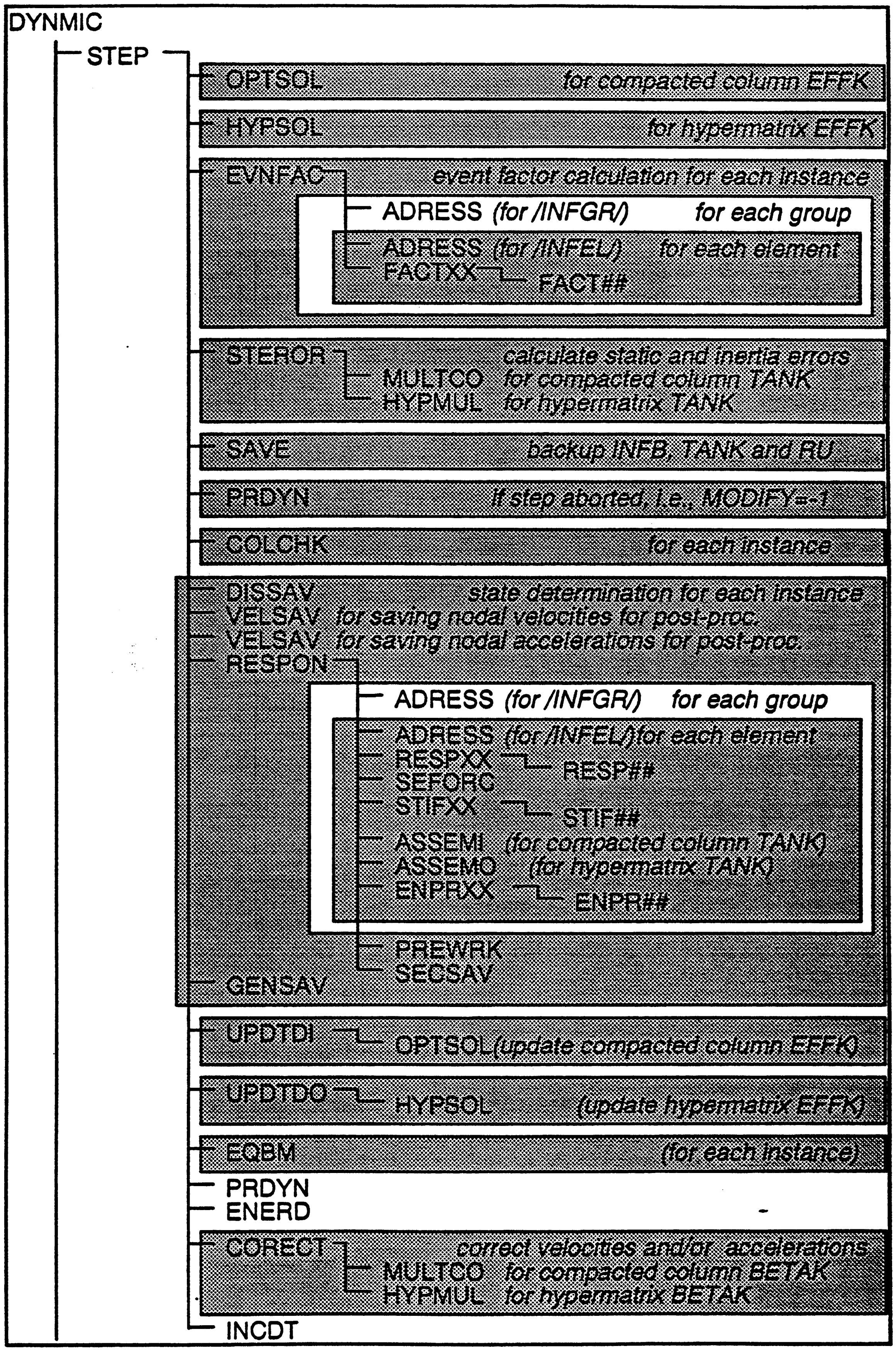
7.14.5. Dynamic Analysis - Subroutine DYNMIC

The flow chart for this subroutine is as follows.


CONTRL
DYNMIC
RESTOR
SETVEC
if MODIFY=-1
SETVAL
DAMPER
if MODIFY<>0 form EFFK
UPDTDI
OPTSOL for compacted column EFFK
UPDTDO
HYPSOL for hypermatrix EFFK
INTPOL
form DEXT
ADEXTF (for floors) for *ACCN or *ACCR
ADEXTI (for interfloors)
GDDEXT for *DISN or *DISR
DFDEXT for *FORN or *FORR
EFLOAD
form XLOD
STEP
driver to advance the solution by a step
PRLOG
UPLOG
SSTATE

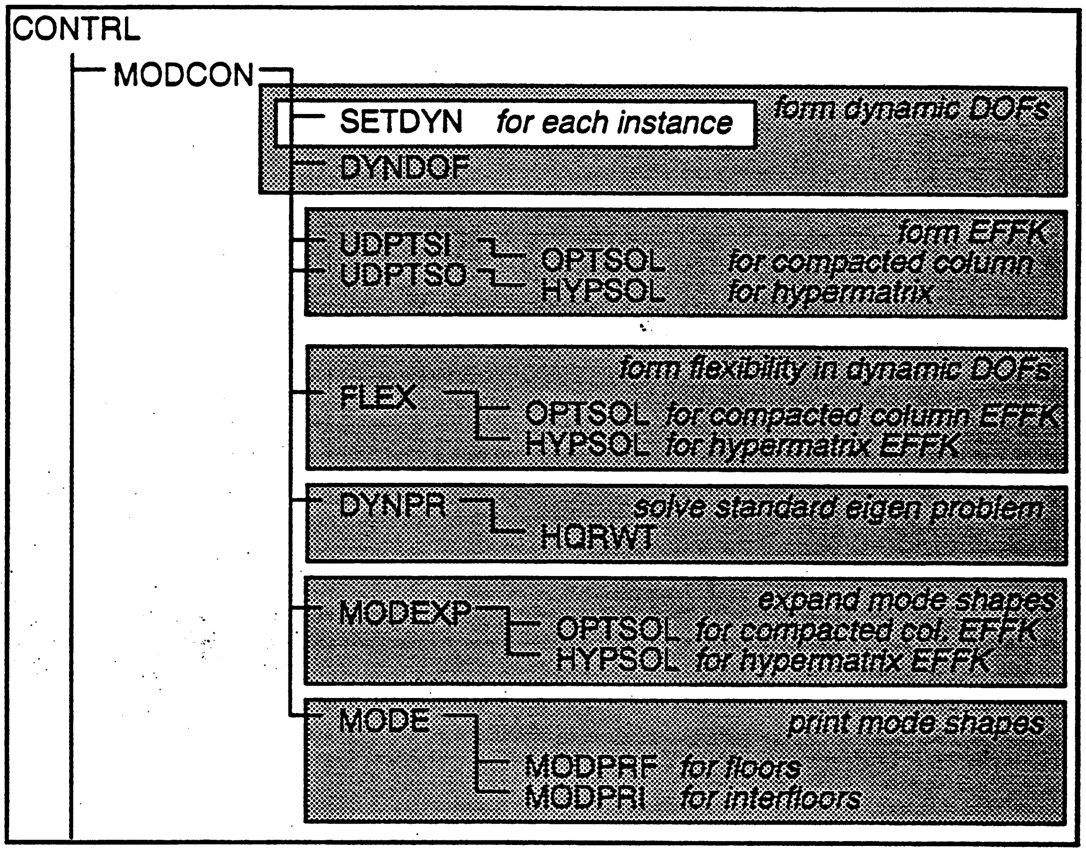
Note that in DRAIN-BUILDING, subroutines ADEXF and ADEDTI replace subroutine GADEXT of DRAIN-2DX and 3DX.

The flow chart for subroutine STEP is as follows.


DYNMIC
STEP
OPTSOL for compacted column EFFK
HYPSOL for hypermatrix EFFK
EVNFAC event factor calculation for each instance
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/) for each element
FACTXX
FACT##
STEROR calculate static and inertia errors
MULTCO for compacted column TANK
HYPMUL for hypermatrix TANK
SAVE backup INFB, TANK and RU
PRDYN if step aborted, i.e., MODIFY=-1
COLCHK for each instance
DISSAV state determination for each instance
VELSAV for saving nodal velocities for post-proc.
VELSAV for saving nodal accelerations for post-proc.
RESPON
ADRESS (for /INFGR/) for each group
ADRESS (for /INFEL/)for each element
RESPXX
RESP##
SEFORC
STIFXX
STIF##
ASSEMI (for compacted column TANK)
ASSEMO (for hypermatrix TANK)
ENPRXX
ENPR##
PREWRK
SECSAV
GENSAV
UPDTDI
OPTSOL(update compacted column EFFK)
UPDTDO
HYPSOL (update hypermatrix EFFK)
EQBM (for each instance)
PRDYN
ENERD
CORECT correct velocities and/or accelerations
MULTCO for compacted column BETAK
HYPMUL for hypermatrix BETAK
INCDT

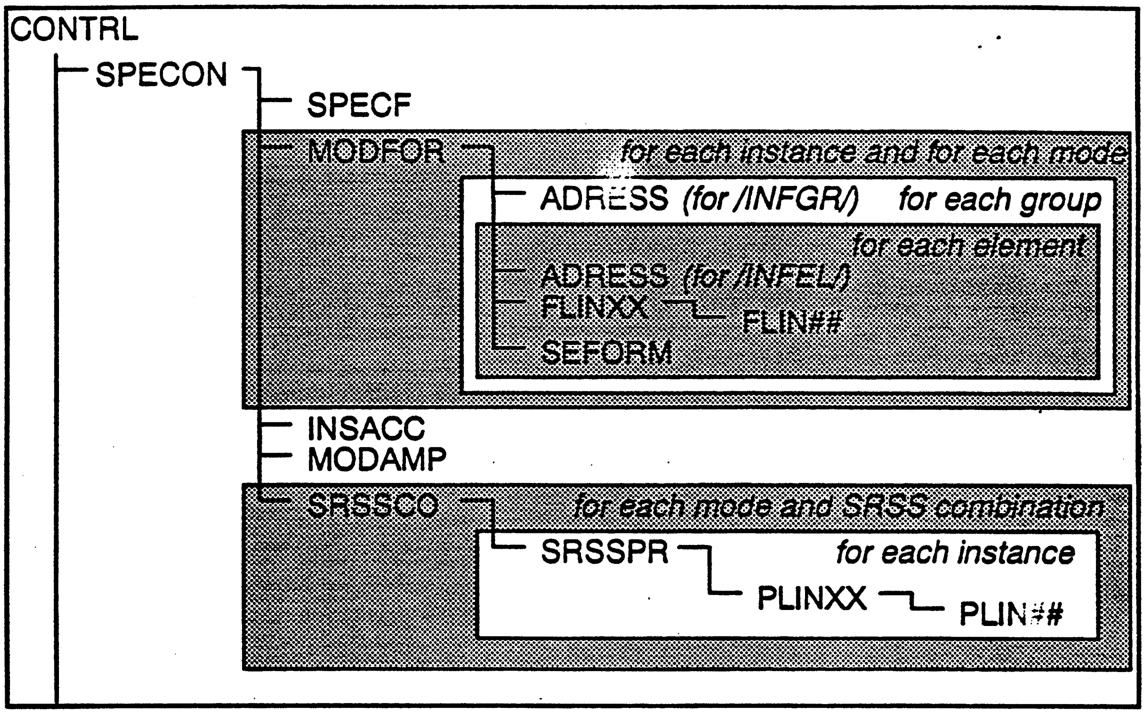
7.14.6. *MODE Analysis - Subroutine MODCON

The flew chart for this subroutine is as follows.


CONTRL
MODCON
SETDYN for each instance
DYNDOF
form dynamic DOFs
UDPTSI
UDPTSO
OPTSOL for compacted column
HYPSOL for hypermatrix
form EFFK
FLEX
form flexibility in dynamic DOFs
OPTSOL for compacted column EFFK
HYPSOL for hypermatrix EFFK
DYNPR
HQRWT
solve standard eigen problem
MODEXP
expand mode shapes
OPTSOL for compacted col. EFFK
HYPSOL for hypermatrix EFFK
MODE
print mode shapes
MODPRF for floors
MODPRI for interfloors

7.14.7. *SPEC Analysis - Subroutine SPECON

The flow chart for this subroutine is as follows.

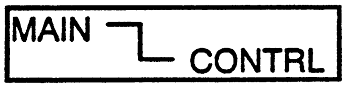


7.15. END ANALYSIS SESSION

The analysis session ends with one of the following conditions.

1. If IQUIT = -1 : last analysis segment could not be completed.
2. *STOP separator is read from input file.
3. Program stops because of errors in the input file or insufficient memory.

The flow chart for this phase is as follows.



The following tasks are performed.

1. In CONTRL, write the farewell message to the .ECH file (unit IOU).
2. In MAIN, close files and stop program execution.

8. STRUCTURE OF PERMANENT FILES -- DRAIN-BUILDING

8.1. INTRODUCTION

In section 6.4, the permanent files created by DRAIN-BUILDING were listed. Some of these files are binary files that may be used by other programs, particularly for post-processing. In this chapter, the structures of these files are described in detail.

8.2. FILES FOR LOAD PATTERNS AND LOAD RECORDS

8.2.1. .ELD File - Element Load Patterns

The .ELD file contains static element load patterns, each specified by *ELEMLOAD data. Each new pattern is appended to the .ELD file in subroutine INGPAT. The applied patterns are retrieved for *GRAV analysis in subroutine INELOD.

Each pattern consists of a number of FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
FIFTYP	character*4	ID of affected floor type or interfloor type.
PATIT	character*40	Pattern title.
NGRPL	integer	Number of element groups loaded by the pattern.

2. For each loaded element group, two records as follows.

   a) First record.

Variable	Type	Description
IGRC	integer	Element group number.
NLOD	integer	Number of element load sets for this group.

b) Second record.

Array	Description
SETLOD(ninl,nlod)	Element load sets. Each column of SETLOD contains a load set. NINL is the length of a load set from array NINFL in /GENINF/. NLOD is the number of element load sets.
ELFACT(nlod,nmem)	Load set scale factors for elements of the group. Each column of ELFACT contains the load set scale factors for one element. NMEM is the number of elements from array NELEM in /GENINF/.

8.2.2. .STA File - Static Load Patterns

The .STA file contains static nodal load patterns, each specified by *NODALOAD data. Each new pattern is appended to the .STA file in subroutine INSPAT. The applied patterns are retrieved for *GRAV or *STAT analysis in subroutine INLOD1 and INLOD2.

Each pattern consists of two FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
FIFTYP	character*4	ID of affected floor type or interfloor type.
PATIT	character*40	Pattern title.

2. Second record.

Array	Type	Description
XPAT(6,nnods+1) for a floor type. XPAT(6,nnods) for an interfloor type.	real	Nodal loads. NNODS is number of nodes from NNODSF in /CNTFIF/. For a floor type, the last column of XPAT contains loads on the diaphragm center, as follows. XPAT(1,nnods+1) = X force. XPAT(2,nnods+1) = Y force. XPAT(6,nnods+1) = Z moment.

8.2.4. .ACC, .DIS and .FRC Files - Dynamic Load Records

The .ACC, .DIS and .FRC files contain ground acceleration, ground displacement and dynamic force records, respectively. The structure of these files is identical to that for DRAIN-2DX and 3DX.

8.2.5. .SPC File - Response Spectra

The .SPC file contains response spectra, each specified by *SPECTRUM data. The structure of this file is identical to that for DRAIN-2DX and 3DX.

8.2.6. .VEL File - Initial Velocity Patterns

The .VEL file contains initial velocity patterns, each specified by *NODALVEL data. Each new pattern is appended to the .VEL file in subroutine INVPAT. The applied patterns are retrieved for *VELN or *VELR analysis in subroutine INVEL1 and INVEL2.

Each pattern consists of two FORTRAN records as follows.

1. First record.

Variable	Type	Description
PATID	character*4	Pattern name.
FIFTYP	character*4	ID of affected floor type or interfloor type.
PATIT	character*40	Pattern title.

2. Second record.

Array	Type	Description
XPAT(6,nnods+1) for a floor type. XPAT(6,nnods) for an interfloor type.	real	Nodal velocities. NNODS is number of nodes from NNODSF in /CNTFIF/. For a floor type, the last column of XPAT contains velocities of the diaphragm center, as follows. XPAT(1,nnods+1) = translational X. XPAT(2,nnods+1) = translational Y. XPAT(6,nnods+1) = rotational Z.

8.3. FILES FOR POST-PROCESSING

8.3.1. .GEO File - Structure Geometry

The .GEO file contains structure geometry data. The .GEO file is opened in subroutine CONTRL before model definition and closed after it.

The data consists of FORTRAN records, as follows.

1. First record is written in CONTRL.

Variable	Type	Description
FNAME	character*8	Problem name (in /TITLE/).
IHED	character*40	Problem title (in /TITLE/).

2. One record is written in subroutine INCNDS or INDTP2.

Variable	Type	Description
NDTP	integer	Number of compound node types +1 (in /CONTR/).
NSNDS	integer	Total number of subnodes in all compound node types (in /CONTR/).

3. If NDTP >1, one record is written in INDTP2, as follows.

Array	Description
NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type NT is equal to NSB(nt+1)-NSB(nt).
NDFSB(6,nsnds)	DOF codes for all subnodes of each compound node type, as follows. 0 : Absolute displacement. 1 : Restrained (not a DOF). 2 : Relative displacement w.r.t. main node. 3 : Special degree of freedom (i.e., not a conventional translation or rotation).
COSB(3,nsnds)	Coordinate offsets from main node for subnodes of each compound node type.

4. The data for each floor type consists of the following records.

a) First record written in subroutine FLRTYP.

Variable	Type	Description
IND	character*2	"FT" to indicate a floor type.
FTYP	character*4	Floor type ID (see IDFIFT in /TITFIF/).

b) One record written in subroutine FLRTYP.

Variable	Type	Description
IRFD	integer	Rigid floor diaphragm code (see JIRFD in /CNTFIF/).
FLRDC(2)	real	X and Y coordinates of diaphragm center (see FLRTDC in /CNTFIF/).

c) One record written in subroutine INFLRT.

Variable	Type	Description
NNODS	integer	Number of floor nodes from NNODSF in /CNTFIF/.
NTNDS	integer	Number of nodes and subnodes in the floor from NTNDSF in /CNTFIF/.

d) If NNODS > 0, one record written in INFLRT.

Array	Description
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

e) If NNODS > 0, one record written in INFLRT.

Array	Description
COORD(3,nnods)	Nodal coordinates, in ascending node number order.

f) If NNODS > 0, one record written in INFLRT.

Array	Description
ID(6,ntnds)	Equation numbers for displacements at each node and subnode, coded as follows: 0 : Restrained displacement. +n : Displacement is unrestrained and unslaved. 'n' = local equation number. -n : Displacement is slaved to the floor master node. 'n' = NTNDS.

g) Two records for each element group are written in subroutine INELEM. The sequence is terminated by a dummy first record.

i) First record.

Variable	Type	Description
IGRC	integer	Element group number.
KEL	integer	Element type number.
NMEM	integer	Number of elements.
NELNOD	integer	Number of nodes per element.
IEHD	character*40	Group title.

ii) If IGRC > 0, one record as follows.

Array	Type	Description
IELNOD(nelnod,nmem)	integer	Nodes for each element, coded as follows. n : 'n'= IFIF + 3 × NLOC where, IFIF = 0. NLOC = sequence number of the node.

5. The data for each interfloor type consists of the following records.

a) First record written in subroutine IFRTYP.

Variable	Type	Description
IND	character*2	"IT" to indicate an interfloor type.
FTYP	character*4	Interfloor type ID (see IDFIFT in /TITFIF/).

b) One record written in subroutine IFRTYP.

Variable	Type	Description
JIFRGT(2)	integer	Number of floor 1 and floor 2 of the interfloor (see /CNTFIF/).
XYZIFR(3)	real	X, Y and Z coordinates of origin of floor 2 relative to the origin of floor 1 (see /CNTFIF/).

c) Next record written in subroutine INIFRT.

Variable	Type	Description
NNODS	integer	Number of interfloor nodes (see NNODSF in /CNTFIF/).
NTNDS	integer	Number of nodes and subnodes in the interfloor (see NTNDSF in /CNTFIF/).

d) If NNODS > 0, one record written in INIFRT.

Array	Description
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

e) If NNODS > 0, one record written in INFLRT.

Array	Description
COORD(3,nnods)	Nodal coordinates, in ascending node number order.

f) If NNODS > 0, one record written in INFLRT.

Array	Description
ID(6,ntnds)	Equation numbers for displacements at each node and subnode, coded as follows. 0 : Restrained displacement. n : Displacement is unrestrained and unslaved. 'n' = local equation number.

g) Two records for each element group are written in subroutine INELEM. The sequence is terminated by a dummy first record.

i) First record.

Variable	Type	Description
IGRC	integer	Element group number.
KEL	integer	Element type number.
NMEM	integer	Number of elements.
NELNOD	integer	Number of nodes per element.
IEHD	character*40	Group title.

ii) If IGRC > 0, one record as follows.

Array	Type	Description
IELNOD(nelnod,nmem)	integer	Nodes for each element, coded as follows. n : 'n'= IFIF + 3 × NLOC where, IFIF = 0 for an interfloor node. IFIF = 1 for floor 1 node. IFIF = 2 for floor 2 node. NLOC = sequence number of the node.

6. Two records for each floor instance, written in subroutine FLRINS.

a) First record.

Variable	Type	Description
IND	character*2	"FI" to indicate a floor instance.
FTYP	character*4	Floor instance ID (see IDFIF in /TITFIF/).

b) Second record.

Variable	Type	Description
JFIFT	integer	Type number for the floor instance (see /CNTFIF/).
XYZFIF(3)	real	X, Y and Z coordinates of the floor origin (see /CNTFIF/).

7. Two records for each interfloor instance, written in subroutine IFRINS.

   a) First record.

Variable	Type	Description
IND	character*2	"II" to indicate an interfloor instance.
FTYP	character*4	Interfloor instance ID (see IDFIF in /TITFIF/).

   b) Second record.

Variable	Type	Description
JFIFT	integer	Type number for the interfloor instance (see /CNTFIF/).
JIFR(2)	integer	Number of floor 1 and floor 2.

8.3.2. .EXX File - Result Envelopes

The .EXX files contain result envelopes for post-processing, organized in FORTRAN records as follows.

1. First record written in subroutine EXINIT.

Variable	Type	Description
IHED	character*40	Problem title (in /TITLE/).
FNAME	character*8	Problem name (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).
NSEG	integer	Analysis segment number (in /CONTR/).

2. Second record written in EXINIT.

Variable	Type	Description
NDTP	integer	Number of compound node types +1 (in /CONTR/).
NFLR	integer	Number of floor instances (in /FLRIFR/).
NIFR	integer	Number of interfloor instances (in /FLRIFR/).

3. If NDTP > 1, one record written in EXINIT.

Array	Description
NSB(ndtp)	Location of first subnode for each compound node type in arrays NDFSB and COSB. The number of subnodes in compound node type NT is equal to NSB(nt+1)-NSB(nt).

4. One record written in EXINIT.

Array	Type	Description
IDFIF(nflr)	character*4	ID's of floor instances (in /TITFIF/).

5. One record written in EXINIT.

Array	Type	Description
IDFIF(nifr)	character*4	ID's of interfloor instances (in /TITFIF/).

6. One record containing control information for each floor instance is written in EXINIT.

Variable	Type	Description
IRFD	integer	Rigid floor diaphragm code (see JIRFD in /CNTFIF/).
NNODS	integer	Number of nodes (see NNODS in /CNTFIF/).
NTNDS	integer	Number of nodes and subnodes (see NTNDSF in /CNTFIF/).
NELGR	integer	Number of element groups (see NELGRF in /CNTFIF/)..
NELTOT	integer	Number of elements (see NELTTF in /CNTFIF/).
NRDS	integer	Number of generalized displacements (see NRDSF in /CNTFIF/).

7. One record containing control information for each interfloor instance is written in EXINIT.

Variable	Type	Description
NNODS	integer	Number of nodes (see NNODS in /CNTFIF/).
NTNDS	integer	Number of nodes and subnodes (see NTNDSF in /CNTFIF/).
NELGR	integer	Number of element groups (see NELGRF in /CNTFIF/)..
NELTOT	integer	Number of elements (see NELTTF in /CNTFIF/).
NSSEC	integer	Number of sections (see NSSECF in /CNTFIF/).
NRDS	integer	Number of generalized displacements (see NRDSF in /CNTFIF/).

8. The following records for each floor instance are written in EXINIT.

a) If NNODS > 0, one record.

Array	Description
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

b) If NELGR > 0, one record.

Array	Description
NELEM(nelgr)	Number of elements in each group.
KELEM(nelgr)	Element type number for each group.
NINFT(nelgr)	Number of output items per element.

9. The following records for each interfloor instance are written in EXINIT.

   a) If NNODS > 0, one record.

Array	Description
NDID(3,nnods)	Node identification array. 1 : Node number. 2 : Compound node type (0 = not a compound node). 3 : Location of nodal DOFs in ID array. For compound nodes, the subnode DOFs immediately follow the main node DOFs in the ID array.

   b) If NELGR > 0, one record.

Array	Description
NELEM(nelgr)	Number of elements in each group.
KELEM(nelgr)	Element type number for each group.
NINFT(nelgr)	Number of output items per element.

10. The subsequent records consist of result envelopes. Data for each envelope consists of up to 8 records per floor or interfloor instance. If NNODS > 0, records 1 to 4 are written in subroutine DISSAV; if NSSEC > 0, records 5 and 6 are written in subroutine SECSAV; and if NRDS > 0, records 7 and 8 are written in subroutine GENSAV. Presently there is no provision to save element envelopes on the .EXX file.

R.No.	Array	Type	Description
1.	DENP(6,ntnds)	real*4	Positive nodal displacement envelopes.
2.	ISTP(6,ntnds)	integer	Step numbers for DENP.
3.	DENN(6,ntnds)	real*4	Negative nodal displacement envelopes.
4.	ISTN(6,ntnds)	integer	Step numbers for DENN
5.	SECENV(6,6,nssec)	real*4	Section force envelopes. 2nd index indicates the envelope type as follows. 1 : Total positive. 2 : Total negative. 3 : Static positive. 4 : Static negative. 5 : Damping positive. 6 : Damping negative.
6.	ISECEN(6,6,nssec)	integer	Step numbers for SECENV.
7.	RDSENV(2,nrds)	real*4	Positive and negative generalized displacement envelopes.
8.	IRDSEN(2,nrds)	integer	Step numbers for RDSENV.

8.3.2. .RXX File - Result Histories

The .RXX files contain time or load history results for post-processing. The .RXX file is opened in subroutine EXINIT for each analysis segment and closed in subroutine SEGEND. The data consists of a number of FORTRAN records, as follows.

1. First record written in EXINIT.

Variable	Type	Description
IHED	character*40	Problem title (in /TITLE/).
FNAME	character*8	Problem name (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).
NSEG	integer	Analysis segment number (in /CONTR/).

2. Second record is written in EXINIT.

Variable	Type	Description
NFLR	integer	Number of floor instances (in /FLRIFR/).
NIFR	integer	Number of interfloor instances (in /FLRIFR/).

3. Third record written in EXINIT.

Array	Type	Description
IDFIF(nflr)	character*4	ID's of floor instances (in /TITFIF/).

4. Fourth record written in EXINIT.

Array	Type	Description
IDFIF(nifr)	character*4	ID's of interfloor instances (in /TITFIF/).

5. One record containing control information for each floor instance is written in EXINIT.

Variable	Type	Description
JFLR	integer	Output code for diaphragm center displacements (see JFLR in /CNTFIF/) for the floor.
NDPOST	integer	Number of floor nodes and subnodes in a post-processing set (see /RHIST/).
NELTHP	integer	Number of floor elements in a post-processing set (in /RHIST/).
NNRDSP	integer	Number of floor generalized displacements in a post-processing set (in /RHIST/).

6. One record containing control information for each interfloor instance is written in EXINIT.

Variable	Type	Description
NDPOST	integer	Number of interfloor nodes and subnodes in a post-processing set (in /RHIST/).
NELTHP	integer	Number of interfloor elements in a post-processing set (in /RHIST/).
NPSECP	integer	Number of interfloor structure sections in a post-processing set (in /RHIST/).
NNRDSP	integer	Number of interfloor generalized displacements in a post-processing set (in /RHIST/).

7. For each floor instance, the following records are written in EXINIT.

   a) If NDPOST > 0, one record.

Array	Description
INFNOD(3,ndpost)	Information for each node and/or subnode in a post-processing set, as follows. 1 : Node number. 2 : Compound node type number (0 = not a compound node). 3 : Subnode number (0 = main node).

   b) If NELTHP > 0, one record.

Array	Description
INFELM(4,nelthp)	Information for each element in a post-processing set, as follows. 1 : Element group number. 2 : Element type number. 3 : Element number. 4 : Number of output items per element (length of element results).

c) If NNRDSP > 0, one record.

Array	Description
INFRDS(nnrdsp)	Numbers of generalized displacements in a post-processing set.

8. For each interfloor instance, the following records are written in EXINIT.

a) If NDPOST > 0, one record.

Array	Description
INFNOD(3,ndpost)	Information for each node and/or subnode in a post-processing, as follows. 1 : Node number. 2 : Compound node type number (0 = not a compound node). 3 : Subnode number (0 = main node).

b) If NELTHP > 0, one record.

Array	Description
INFELM(4,nelthp)	Information for each element in a post-processing set, as follows. 1 : Element group number. 2 : Element type number. 3 : Element number. 4 : Number of output items per element (length of element results).

c) If NPSECP > 0, one record.

Array	Description
INFSEC(npsecp)	Numbers of structure sections in a post-processing set.

d) If NNRDSP > 0, one record.

Array	Description
INFRDS(nnrdsp)	Numbers of generalized displacements in a post-processing set.

9. The subsequent records consist of results for post-processing. The results at each step consists of the following records.

a) First record, written in subroutine GRSOL, STATIC, REST or STEP.

Variable	Type	Description
KSTEP	integer	Step number.
TIME	real*4	Current time for dynamic analysis segment. Current load factor for static analysis segment.

b) Following records for each floor and interfloor instance.

i) If NDPOST > 0, one record, written in subroutine DISSAV.

Array	Type	Description
DISP(6,ndpost)	real*4	Node and subnode displacements.

ii) If NDPOST > 0, two records for a dynamic analysis segment (ANAL='ACCN' or 'ACCR' or 'VELN' or 'VELR' or 'DISN' or 'DISR' or 'FORN' or 'FORR'), written in subroutine VELSAV.

R.No.	Array	Type	Description
1.	VELO(6,ndpost)	real*4	Node and subnode velocities.
2.	ACCE(6,ndpost)	real*4	Node and subnode accelerations.

iii) NELTHP records, written in subroutine RESPON.

Array	Type	Description
THOUT(*)	real*4	Element results. The length of element results depends on the element.

iv) If NPSECP > 0, one record, written in subroutine SECSAV.

Array	Type	Description
SECFRC(12,npsecp)	real*4	Static and damping section forces. Static followed by damping for each section.

v) If NNRDSP > 0, one record, written in subroutine GENSAV.

Array	Type	Description
GEDISP(nnrdsp)	real*4	Generalized displacements.

8.4. MODAL ANALYSIS FILES

8.4.1. .MXX File - Mode Shapes

The .MXX files contain results from mode shapes and periods analysis. The results consist of the following FORTRAN records, written in subroutine MODE.

1. First record.

Variable	Type	Description
FNAME	character*8	Problem name (in /TITLE/).
IHED	character*40	Problem title (in /TITLE/).
ANAL	character*4	Analysis segment type (in /TITLE/).
IHEDA	character*40	Analysis title (in /TITLE/).

2. Second record.

Variable	Type	Description
NEQT	integer	Length of each mode shape (in /EQNS/).
NVEC	integer	Number of mode shapes.

3. Third record.

Array	Description
EVAL(nvec)	Mode periods.

4. Fourth record.

Array	Description
XLM(3,nvec)	Mass participation factors in translational directions for each mode.

5. NVEC records, one for each mode shape.

Array	Description
EVEC(neqt)	Mode shape = $\underline{\phi}/\omega^2$. Where, $\underline{\phi}$ is mass normalized mode shape; $\omega = 2\pi/T$; and T = mode period.

8.4.2. .UXX File - Modal Responses

The .UXX files contain response results for unit modal amplitudes. The results consist of the following FORTRAN records, written in subroutine SPECON.

1. First record.

Variable	Type	Description
NVEC	integer	Number of mode shapes.
LENGTH	integer	Minimum length of record buffer required to read the response results.

2. Second record.

Array	Description
EVAL(nvec)	Mode periods.

3. Third record.

Array	Description
XLM(3,nvec)	Mass participation factors in translational directions for each mode.

4. NVEC sets of records, one set per mode shape. Each set consists of one record per (floor or interfloor) instance. Each record contains the following arrays consecutively.

Array	Type	Description
RESNDS(6,ntnds)	real*4	Response nodal displacements
RESELM(nlin,nmem)	real*4	One array per element group, containing element response results. NLIN is the number of result items per element for the group (NLINF in /GENINF/). NMEM is the number of elements for the group (NELEM in /GENINF/).
RESSEC(6,nssec)	real*4	Response static section forces.
RESRDS(nrds)	real*4	Response generalized displacements.

9. ELEMENT INTERFACE AND SUBROUTINES

9.1. ELEMENT SUBROUTINES

Each DRAIN program consists of a base program and a library of element subroutines. The base program is responsible for memory and file management, the nonlinear solution strategy, and the processing of nodal data. The element subroutines are responsible for processing element data.

A principal goal in designing the programs has been to allow new elements of a wide variety of types to be added without changing the base program.

To add a new element type it is necessary to develop ten element subroutines. Each subroutine performs a well-defined series of tasks, as shown in the following table. The "##" at the end of the subroutine name is a two digit number which identifies the element type. For example, for element type 2, the subroutines are INEL02, ELOD02, etc.

Subroutine	Element Tasks
INEL##	Input and initialize element data.
ELOD##	Input element load patterns. If there is no provision for element loads, this subroutine and subroutine GLOD## are dummy subroutines.
GLOD##	Set up actual element loads and initialize element forces, using element load patterns.
STIF##	Calculate element stiffness or change in stiffness. This is the "linearization" phase.
FACT##	Calculate element event factor for event-to-event analysis.
RESP##	Update element state. This is the "state determination" phase.
ENPR##	Print element envelope results.
THPR##	Print element time history results.
FLIN##	Calculate linear element actions and deformations for response spectrum analysis.
PLIN##	Print element actions and deformations for response spectrum analysis.

These subroutines may call other element subroutines and utility subroutines. As a matter of good practice, the names of all element subroutines should end with the element type number.

The base program and element subroutines communicate through the ***element interface***, which consists of the following.

1. Argument lists of the element subroutines.
2. Labeled common blocks /CLINE/, /ELMPAR/, /INFEL/, /INFGR/, /PTOP/, /TAPES/, and /THELM/. See Chapter 2 for a description of these common blocks.
3. Argument lists of the base program subroutines ECONTR, COORDS, ELNODE, LOCMAT and FINISH, which are called by element subroutine INEL##.
4. Argument list of the base program subroutine SFORCE, which is called by element subroutine GLOD##.

The element interface and the specific tasks performed in each element subroutine are described in detail in the following sections.

9.2. KEY VARIABLES

9.2.1. General

A number of key variables control the computations in the element subroutines. These can be grouped into three categories, as follows.

1. Overall structure variables.
2. Element group variables.
3. Individual element variables.

The variables are defined in this section. Specific procedures for using them are described later. At present these variable are all passed through argument lists.

9.2.2. Overall Structure Variables

Structure variables are defined for the structure as a whole, and have the same value for all element groups. The values of these variables are set by the base program.

The variables are as follows.

Variable	Description
KENR	Code for calculating element energies. 0 : Omit calculations. 1 : Calculate elastic-plastic (static) energy only. 2 : Calculate both elastic-plastic and damping (dynamic) energies.
KRESIS	Code for calculating element resisting forces. 1 : Calculate elastic-plastic (static) resisting force only. 2 : Calculate elastic-plastic and damping (dynamic) resisting forces.
KSTAT	Code for type of analysis. 1 : Static. 2 : Dynamic.
KSTEP	Current analysis step number.
KSTT	Code for forming element stiffness. 1 : Form total stiffness. 0 : Form change in stiffness.
KTYPE	Code for type of element stiffness to be formed. 1 : Elastic-plastic stiffness only. 0 : Elastic-plastic plus geometric stiffness. -1 : Geometric stiffness only.
TIM	Current time for dynamic analyses. Current load factor for static analyses.

9.2.3. Element Group Variables

Group variables are defined for each element group, and have the same value for all elements in the group. Except as noted, the values of these variables are set by the base program.

The variables are as follows.

Variable	**Description**
BETO	Initial stiffness damping factor. Each element has a damping matrix equal to BETO*K0 , where K0 = stiffness of element in its initial state.
ELFACT(nlod,nmem)	Load set scale factors for elements of the group. Set up for each loaded element group of an element load pattern. Each column contains the load set scale factors (see array SETLOD) for one element. Term I,J stores the scale factor by which load set I is to be multiplied for element J, to define the load pattern. ELFACT is set in subroutine ELOD##, and used only if element loads are permitted for the element type.
KEVE	Event calculation code. 1 : Event factors are to be calculated for elements in the current group. 0 : Event factors are not required.
KGEM	P-Δ analysis code. Note that P-Δ effects can add a great deal of complexity. 0 : Ignore P-Δ effects. 1 : Consider P-Δ effects and allow geometric stiffness to change for static analyses only. 2 : Consider P-Δ effects and allow geometric stiffness to change for both static and dynamic analyses.
NDOF	Number of element DOFs (rows in element stiffness matrix) for elements in current group. Set in subroutine INEL##.
NELNOD	Number of element nodes for elements in current group. Set in subroutine INEL##.

NFLIN	Number of output items per element for response spectrum analysis. Set in subroutine INEL##.
NINFL	Number of words required to store an element load set. Set in subroutine INEL##.
NINFT	Number of words in element output block (/THELM/) for each element in current group. Set in subroutine INEL##.
NLOD	Number of element load sets for current group and current element load pattern.
NMEM	Number of elements in current group.
SETLOD(ninfl,nlod)	Element load sets. Each column defines one load set. Set in subroutine ELOD##.

9.2.4. Individual Element Variables

Element variables are defined for each element individually and can have different values for each element. Except as noted, the values of these variables are set by the base program.

The variables are as follows.

Variable	Description
DDISE(ndof)	Increments in element end displacements in current analysis step. Used in subroutine FACT## to calculate element event factor and in subroutine RESP## to update element state.
DISE(ndof)	Total element end displacements at end of current step. Used in subroutine RESP## for calculating P-Δ effects.
EFAC	Element event factor, i.e., the proportion of DDISE required to reach the next significant nonlinear event. If EFAC $\geq$ 1.0, no event occurs within the current analysis step. If EFAC < 1.0, an event occurs and the base program must subdivide the step. EFAC is calculated only for those element groups for which KEVE = 1. Set in subroutine FACT##.
ENED	Increment in element beta-K damping work for current displacement increment. Set in subroutine RESP## if KENR > 1.

ENER	Increment in element elastic-plastic (static) work for current displacement increment. Set in subroutine RESP## if KENR > 0.
ENSO	Increment of element P-Δ work for current displacement increment. Set in subroutine RESP## if KENR > 0 and KTYPE ≤ 0. Because of P-Δ effects, the effective work done on an element is not equal to the simple elastic-plastic work. ENSO is needed to correct for this.
FK(ndof,ndof)	Element stiffness matrix (depends on the structure variables KSTT and KTYPE). Set in subroutine STIF##.
IMEM	Number of current element within its group. Set in subroutine INEL##.
KDATA	Data error counter. Initialized to zero before calling subroutine INEL## or ELOD##. Incremented by 1 for each input data error.
KSAVE	Code for saving element results for printout or post-processing. 1 : Set up results for current load or time step in /THELM/. The results are saved by the base program. 2 : Do not set up.
KST	Stiffness change code. Set in subroutine RESP## as follows. 1: Stiffness for current element has changed in the current step. Subroutine STIF## will be called to update the stiffness for this element. 0: No stiffness change has occurred. Subroutine STIF## will not be called.
RDAMP(ndof)	Element end damping forces. Set in subroutine RESP##.
RELAS(ndof)	Element end elastic-plastic (static) forces. Set in subroutine RESP##.
VELE(ndof)	Element end velocities at end of current step in dynamic analysis. Used in subroutine RESP## to calculate element damping forces.

9.3. LABELED COMMON BLOCKS

9.3.1. General

Data must be stored for each element, to monitor its nonlinear behavior. This data is transferred to and from the element subroutines by means of the labeled common blocks /INFGR/ and /INFEL/. There is one /INFGR/ data block for each element group (containing data common to all elements in the group) plus one /INFEL/ data block for each element (containing data unique to each element). The length of /INFGR/ may be zero.

In addition, element results for printout or post-processing are transferred by means of the labeled common block /THELM/.

9.3.2. Group Information Block, /INFGR/

The /INFGR/ block contains data common to all elements in the current group. The values in /INFGR/ must be initialized in subroutine INEL##, and once initialized must not be modified. The block may be divided in any way the programmer wishes, but care must be taken to ensure that real variables start on 8-byte boundaries. The length of /INFGR/ is conveyed to the base program in subroutine INEL##. The base program stores the block and makes it available for the current element when needed.

9.3.3. Element Information Block, /INFEL/

The /INFEL/ block contains data unique to each element. It must be initialized in subroutine INEL##, and then progressively updated in RESP## and (if necessary) in STIF##. All data to be retained for any element must be contained either in /INFEL/ or in /INFGR/. The block may be divided in any way the programmer wishes, except that the first variable must be IMEM (the element number in its group), and care must be taken to ensure that real variables start on 8-byte boundaries. The length of /INFEL/ is conveyed to the base program in subroutine INEL##. The base program stores the block and makes it available for the current element when needed.

The block may be updated in subroutines GLOD##, RESP## and STIF##. After calls to these subroutines the base program saves the updated block. The block is not saved after calls to FACT##, ENPR##, or FLIN##. Hence, any changes made to the /INFEL/ block in these subroutines will not be saved. As a matter of good practice, changes should not be made to /INFEL/ in these subroutines.

During the execution of INEL##, first the /INFGR/ block for the element group (if it exists) and then the /INFEL/ blocks for all elements are written to disk. At the end of the input phase, the blocks are brought back into memory and stored compactly in the available memory space. If memory becomes filled, the elements stored constitute an "element block". This block is written to disk and a new block is begun. During subsequent calculations the element blocks are returned to memory as needed and if updated are rewritten to disk. The blocking and I/O operations are all carried out by the base program. If the data for all elements can be held in memory (i.e. only one element block), then there is no I/O of element blocks to disk storage, and considerable execution time may be saved. In order to reduce I/O cost, the analyst may specify that execution of an analysis should not begin unless there is only one element block. This type of blocking will normally be used only on fixed-memory computers. On virtual memory computers it is probably more efficient to request sufficient memory to execute with a single element block.

9.3.4. Element Results Block, /THELM/

For each element, the results at the current load or time step may be saved as the analysis proceeds. The data is transferred by means of the labeled common block /THELM/. The length of /THELM/ is conveyed to the base program in subroutine INEL##, and is the same for all elements in a group.

If the results for any element are to be saved (i.e., KSAVE = 1), these results (consisting of anything that the programmer wishes) must be placed in /THELM/ during execution of subroutine RESP##. The base program keeps track of whether element results are to be saved. The base program buffers the /THELM/ data, and when the buffer is full writes it to a results scratch file (unit NFPRNT) and/or the post processing file (.RXX). At the end of each analysis segment the results on the scratch file are recalled, rearranged and written to the .OUT file using subroutine THPR##, if a printout has been requested. It is recommended that double precision variables be converted to single precision before placing the results in /THELM/, in order to reduce buffer lengths and disk storage.

9.3.5. Work Block

Labeled common block /WORK/ provides a work area for use by the programmer. It provides a means of grouping local variables, and with some compilers may save memory by reducing the number of local variables.

The /WORK/ block should generally not be used to transfer data between element subroutines, because it can be used differently for different elements, and the data may therefore be continually changing.

9.3.6. Tapes Block

Labeled common block /TAPES/ contains the I/O unit numbers. The variables in this block must not be modified. Units INP (DRAIN.INP file) and IOU (.ECH file) may be used in element subroutines for read and write statements. Unit IOU may be used for temporary debugging output if desired.

9.3.7. Element Parameters Block, /ELMPAR/

Labeled common block /ELMPAR/ contains element control parameters which can be changed by means of the *PARAMETERS input data. These parameters can be used to vary the behavior of elements in a group without re-entering the element input data (i.e., without starting a complete new problem). For example, element parameters could be used to turn on debug prints for a particular element. This feature has been added for future use. None of the current elements uses these parameters.

9.4. RESULTS ENVELOPES

Maximum and minimum envelope values for various element response quantities may be stored in the /INFEL/ block, and progressively updated. These quantities can then be printed at specified intervals (in subroutine ENPR##). Although it is acceptable to store only the envelope value, it is usual to store also the load or time step number at which the value was reached. The step number begins with zero in each analysis segment (a zero step number thus indicates that the envelope value was reached in some earlier segment). Hence, at the beginning of each new segment, the step number must be re-initialized to zero. See subroutine RESP## for the procedure.

9.5. ARGUMENT TYPES IN ARGUMENT LISTS

The variables transferred through any argument list are of three types, as follows.

Input: These are set by the calling subroutine and must not be modified in the called subroutine.

Output: These must be set in the called subroutine and returned for use by the calling subroutine. They are not set before entry to the called subroutine.

Modify: These are set by the calling subroutine, and may either be left unchanged or modified in the called subroutine, depending on circumstances.

9.6. SUBROUTINE INEL##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine INEL## are as follows.

```
SUBROUTINE INEL## (KDATA)
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
COMMON /PTOP/ . . .
COMMON /WORK/ . . .
COMMON /TAPES/ INP, IOU
```

(b) Argument type

The argument KDATA is of modify type.

(c) Purpose

INEL## is called once for each element group, in the input phase. Its purpose is to read the input data for all elements in the group, to convey values of several control variables to the base program, to initialize the /INFGR/ and /INFEL/ blocks, to send data enabling the base program to construct the element "location matrices", and to instruct the base program to save the /INFGR/ and /INFEL/ blocks.

(d) Tasks Performed Before Entry

The base program reads and echo prints the Group Information line of the *ELEMENTGROUP data.. The line contains the following data.

i) Element type number.

ii) Event calculation code (KEVE).

iii) P-Δ analysis code (KGEM).

iv) Initial stiffness damping factor (BETO).

v) Group title.

The base program initializes KDATA to zero, then calls INEL## for the corresponding element type.

(e) Tasks to be Performed in INEL##

Subroutine INEL## must read and echo-print the input data for elements in the group, and initialize the /INFGR/ and /INFEL/ blocks. See the subroutines for existing elements, such as INEL01 and INEL04, for examples. Note that the input data is not read directly, but is obtained by calls to the subroutine GETLIN.

The following tasks must be performed.

1. Print appropriate headings for the element type.
2. Convey NDOF, NINFL, NINFT, NFLIN and NELNOD to the base program by calling subroutine ECONTR. The call also initiializes /INFGR/ and /INFEL/ to zero.
3. If desired, establish tables of stiffnesses, strengths, etc., for subsequent use in specifying individual elements. The arrays for these tables may be defined in /WORK/ to avoid creating local variables.
4. Store data common to all elements in /INFGR/. This block must be completed before the call to subroutine FINISH (see below) for the first element in the group. This is because /INFGR/ is saved immediately preceding /INFEL/ for the first element.
5. Read the data for each element in the group, including node numbers, stiffnesses, strengths, etc., typically using the tables established in Step 4. Generation options may be included. The generation procedure used for the current elements is recommended.
6. For each element, carry out the following initialization operations.

   a) Set IMEM to the element number within the group (in sequence starting with 1).

   b) Establish, within /INFEL/, appropriate variables to permit the state of the element to be monitored. Initialize these variables to correspond to the unstressed state of the structure. Among other things these will include variables defining current element actions, which will be initialized to zero and then progressively updated as the analysis proceeds.

c) If the element type has an option for element loads, define variables in /INFEL/ to store initial element actions, in addition to the current values, and initialize to zero. Separate initial and current values are needed for calculation of the element elastic-plastic work (ENER) and resisting force vectors (RELAS and RINIT) in subroutine RESP##. If a gravity load analysis is performed, and if the element has element loads, both the initial and current actions are set, in subroutine GLOD##, at the beginning of the analysis. The initial actions then remain constant, while the current actions are progressively updated. This aspect of the elements is awkward, and an improved procedure may be implemented in the future. Recognize, however, that element loads on nonlinear elements cause major theoretical and computational complications.

d) Save data on the element length, orientation in space, etc., so that subsequent linearization and state determination calculations can be carried out. One way is to store a displacement transformation matrix relating nodal displacements to element deformations. Alternatively, the (X, Y and Z) coordinates of the element nodes may be stored, and the transformations recalculated as needed. The coordinates of nodes are determined by calls to base program subroutine COORDS, one call per node.

e) Initialize all other variables in /INFEL/ to appropriate values corresponding to the unstressed state of the structure.

f) Tell the base program the numbers of the nodes to which the element is connected, with a series of calls to base program subroutine ELNODE, one call per node.

g) Make a series of calls to base program subroutine LOCMAT, one call for each element DOF, in element DOF order (i.e., the order in the element stiffness matrix and resisting force vectors). This provides the data the base program needs to set up the element "location matrix" (i.e., the global equation number for each element DOF). The argument list contains the node number, subnode number (zero, since subnodes are not yet fully implemented), the displacement direction (1 = X translation; 2 = Y translation; etc.), and an error code. For DRAIN-BUILDING it also contains a floor/interfloor code (0 = node belongs to current floor or interfloor; 1 = node belongs to the floor 1 of current interfloor; 2 = node belongs to floor 2 of current interfloor). The base program places the location matrix for any element at the end of the element /INFEL/ block, and adds the required length to the block length NINFE.

h) Call subroutine FINISH with the statement

```
CALL FINISH
```

This call must be made after /INFEL/ for the element has been fully initialized, and must be made once for each element. This instructs the base program to save the /INFGR/ and /INFEL/ blocks.

i) Do appropriate input data checking. For each data error increment KDATA by 1 and echo print an appropriate error message. It will be usual to continue processing the input data if errors are detected. However, if the error is fatal, execution may be stopped by calling EXIT, as follows.

```
CALL EXIT
```

9.7. SUBROUTINE ELOD##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine ELOD## are as follows.

```
SUBROUTINE ELOD## (NMEM, NINFL, NLOD, SETLOD, ELFACT,
1                         KDATA)
COMMON /PTOP/ . . .
COMMON /TAPES/ INP, IOU
DIMENSION SETLOD(ninfl, nlod), ELFACT (nlod, nmem)
```

(b) Argument Types

The arguments are of following types.

Input: NMEM, NINFL, NLOD

Modify: SETLOD, ELFACT

(c) Purpose

Subroutine ELOD## is called once for each element group, each time an element load pattern is input. Its purpose is to store element load sets (SETLOD array) and load set scale factors (ELFACT array) for the load pattern. These arrays are used by subroutine GLOD## to construct load vectors for gravity load analysis.

(d) Tasks Performed Before Entry

For each element group which is loaded by the load pattern, the base program reads one line of input data. This line contains the number of element load sets (NLOD) for the group. The base program then initializes SETLOD and ELFACT to zero, and calls ELOD## for the corresponding element type.

(e) Tasks to be Performed in ELOD##

Each column of array SETLOD will store one load set (typically, but not necessarily, a set of element fixed end forces). It is suggested that the first term of each column be a key to identify the type of load set. For example this key could identify whether fixed end forces are in local (element) or global (structure) coordinates.

Each column of array ELFACT corresponds to an element in the current group. Term I, J in ELFACT is the scale factor by which load set I is to be multiplied for element J. If the factor is zero, the load set does not affect the element.

The following tasks must be performed.

1. Print appropriate headings for the load sets.

2. Read NLOD load sets, store in SETLOD, and echo print.
3. Print appropriate headings for the load set scale factors.
4. Read in all nonzero scale factors, identifying both the element and the load set, and using generation options as appropriate. Store in ELFACT and echo print.
5. If an element has no element loads (i.e. all scale factors are zero for that element), set ELFACT(1) for the element to 999999. This will save some time in later computation.
6. Do appropriate input checking. If there are errors increment KDATA and echo print an appropriate error message.

9.8. SUBROUTINE STIF##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine STIF## are as follows.

```
SUBROUTINE STIF## (KSTT, KTYPE, NDOF, FK)
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /WORK/ . . .
DIMENSION FK(ndof, ndof)
```

(b) Argument Types

The arguments are of the following types.

Input: KSTT, KTYPE, NDOF

Output: FK

(c) Purpose

Subroutine STIF## is called for an element whenever its stiffness changes. Only those elements with stiffness changes are affected (for initial formation of the stiffness this is all elements; for subsequent changes typically only a few elements). Its purpose is to form the element total stiffness or change in stiffness, and return it in array FK.

(c) Tasks Performed Before Entry

The base program sets up /INFGR/ for the current element group and /INFEL/ for the current element before calling STIF##. The variables KSTT and KTYPE are also set, to specify the required stiffness type.

(d) Tasks to be Performed in STIF##

STIF## must have the capability to form both the total stiffness and the change in stiffness since the last time the stiffness was formed. Depending on the values of the variables KSTT and KTYPE, it may be required to return the elastic-plastic stiffness only, the elastic-plastic plus geometric stiffness or the geometric stiffness only.

The following tasks must be performed.

1. KSTT = 0: Form the change in stiffness since the last time subroutine STIF## was called for the element.
   a) KTYPE = 1: Elastic-plastic stiffness only. This will be done if P-Δ effects are being ignored, or if they are being considered but the structure geometric stiffness is assumed to be constant.
   b) KTYPE = 0: Elastic-plastic plus geometric stiffness. This will be done if P-Δ effects are being considered and the structure geometric stiffness progressively changes.
   c) KTYPE = -1: Geometric stiffness only. This will be done if the structure geometric stiffness progressively changes but there is no change in the structure elastic-plastic stiffness.

2. KSTT = 1: Form the total stiffness, in the current element state.
   a) KTYPE = 1: Elastic-plastic stiffness only. This will be done when a complete structure stiffness is being formed and P-Δ effects are being ignored.
   b) KTYPE = 0: Elastic-plastic plus geometric stiffness. This will be done when a complete structure stiffness is being formed and P-Δ effects are being considered.
3. Update variables in the /INFEL/ block so that stiffness changes can be computed when STIF## is next called. For simple elements it may be possible to store indicators which identify the state when STIF## was last called, and to calculate stiffness changes by comparing these "previous" indicators with the "current" values (which must be progressively updated in subroutine RESP##). For more complex elements it may be necessary to store the total stiffness, and to get the stiffness change by subtraction.

9.9. SUBROUTINE GLOD##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine GLOD## are as follows.

```
SUBROUTINE GLOD## (NINFL, NLOD, SETLOD, ELFACT, PMULT)
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /WORK/ . . .
DIMENSION SETLOD(ninfl,nlod), ELFACT(nlod)
DIMENSION ELFINT(ndof)
```

(a) Argument Types

The arguments are of the following types.

Input: NINFL, NLOD, SETLOD, ELFACT, PMULT

Array ELFINT(ndof) is a local array.

(b) Argument Types

The arguments are of the following types.

Input: NDOF, DDISE

Modify: EFAC

Output: KEVT

(c) Purpose

Subroutine FACT## is called once for each element in each static analysis substep, and for each dynamic analysis substep if the event-to-event solution strategy is specified. The purpose of the subroutine is to calculate the element event factor, EFAC, based on the element displacement increment DDISE. A code (KEVT) indicating the type of event is also returned. This code depends on the element type (e.g., 1 = yielding in tension). It is printed as part of the solution log.

(d) Tasks Performed Before Entry

The following tasks are performed by the base program before subroutine FACT## is called.

1. Nodal displacement increments for the current element are set up in array DDISE.
2. EFAC is set to 1.0.
3. /INFGR/ and /INFEL/ are set up for for the current group and element.

(e) Task to be Performed in FACT##

Determine the proportion of DDISE required to reach the next significant nonlinear "event". This is the element event factor. If it is less than EFAC, set EFAC = element event factor and KEVT = a code indicating the event type. If there is no event, set KEVT = 0.

The coding for FACT## will usually be similar to that in RESP##, except that (a) FACT## can stop at the first event, and (b) whereas RESP## typically traces out the behavior from exact event to exact event, FACT## has an event overshoot provision. In INEL##, one or more overshoot tolerances are stored in /INFEL/. The event factor is then calculated as the multiple of DDISE required to reach not, say, the exact yield value, but this values plus the tolerance. The tolerance can be varied from the value input in INEL## by setting the value of an "event overshoot scale factor" (in the *PARAMETERS section of the input). The overshoot tolerance from INEL## is multiplied by the scale factor (FACOV in the FACT## argument list) before the event factor is calculated.

9.11. SUBROUTINE RESP##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine RESP## are as follows.

```
SUBROUTINE RESP## (KRESIS, KSAVE, KGEM, KSTEP, NDOF, KST,
1                   KENR, ENER, ENED, ENSO, BETO, RELAS,
2                   RDAMP, RINIT, DDISE, DISE, VELE)
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
 COMMON /THELM/ . . .
COMMON /WORK/ . . .
DIMENSION RELAS(ndof), RDAMP(ndof), RINIT(ndof), DDISE(ndof),
1          DISE(ndof), VELE(ndof)
```

(b) Argument Types

The arguments are of the following types.

Input: KRESIS, KSAVE, KGEM, NDOF, KENR, BETO, DDISE, DISE, VELE

Output: ENER, ENED, ENSO, RELAS, RDAMP, RINIT

Modify: KSTEP, KST

(c) Purpose

Subroutine RESP## is called once for each element in each analysis substep. Its purpose is to update the element state, form the static and damping resisting forces, perform energy calculations, update response quantities, and put element results in /THELM/ for saving or printing. This is the most complex of the element subroutines.

(d) Tasks Performed Before Entry

The following tasks are performed by the base program before subroutine RESP## is called.

1. KRESIS is set to define the type of resisting force to be calculated. The options are as follows.
   a) KRESIS = 1: elastic-plastic only (static).
   b) KRESIS = 2: elastic-plastic and damping (dynamic).
2. KST is set to zero.
3. KGEM is set to indicate whether P-Δ effects are to be considered.
4. KSAVE is set to indicate whether element results are to be saved for post-processing and/or printout.
5. Element end displacement increments, total displacements and velocities are set up in arrays DDISE, DISE and VELE, respectively.
6. /INFGR/ and /INFEL/ are set up for the current group and element.

(e) Tasks to be Performed in RESP##

The following tasks must be performed.

1. If KSTEP = -1, this is the first entry to RESP## for the current analysis segment. Set any envelope step numbers or times to 0, and reset KSTEP = 1.

2. Update the element state. Typically the following steps will be followed.
   a) Using DDISE, determine increments of element deformations. Hence, by tracing the nonlinear action-deformation relationship, determine increments of element actions. For a complex element this can be a lengthy computation. Update the element deformations, actions, etc., as appropriate.
   b) If KENR = 1, calculate the increment of element elastic-plastic work, ENER.
   c) If KENR > 0 and KGEM > 0, calculate the of element P-Δ (second order) work, ENSO.
   d) Compare the current state with the state when the stiffness matrix was last formed (i.e. when KST was last set to > 0). A commonly used procedure is to assign values to some state indicators, storing previous and current values. If the current value is different from the previous value the state of the element has changed, and its stiffness must be modified for the next step. To signal this set KST to 1. If KST is 1, subroutine STIF## will be called for the element. Do not replace previous state indicators with the current ones in RESP## - this must be done in STIF##.
   e) If KRESIS = 1, form elastic-plastic (static) resisting force vectors, in global coordinates, in arrays RELAS and RINIT. These are the forces acting on the element to satisfy element equilibrium. If there are no element loads, RELAS is the total static resisting force in the current state, and RINIT is zero. However, if there are element loads, RINIT is the resisting force in the initial state, and RELAS is the difference between the total and initial resisting forces (i.e. the change in resisting force since the initial state). RELAS must include P-Δ effects if they are present. Since RINIT is for the element clamped end state it will usually include no P-Δ effects.
   f) If KRESIS = 2, form the element damping force vector, RDAMP. Like RELAS, these are the forces, in global coordinates, which are in equilibrium with the viscous damping force in the element. If KENR = 2, calculate the increment of

element damping work (ENED), equal to the work done by the average damping forces moving through the deformation increments.

g) Update the values of all response quantities for the element.

h) Update the envelope values as necessary and the step numbers or times at which they occur. Note that the step number will be relative to the beginning of the current analysis segment.

i) If KSAVE = 1, put the results for the element at the end of the current load or time step in /THELM/.

9.12. SUBROUTINE ENPR##

(a) Beginning Statements

The SUBROUTINE and COMMON statements for subroutine ENPR## are as follows.

```
SUBROUTINE ENPR## (NFOUTP)
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /PTOP/ . . .
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
```

(b) Argument Types

The argument NFOUTP is of input type.

(c) Purpose

Subroutine ENPR## prints, on unit NFOUTP, envelope values for the current element. ENPR## is called from the base program for all elements at step intervals defined by the input data. In addition, the subroutine is automatically called for all elements at the end of each analysis segment.

(d) Tasks Performed Before Entry

The base program performs the following tasks.

1. Before calling ENPR## for the first element of any element group, the heading containing analysis segment number and element group number is printed to NFOUTP.
2. /INFGR/ and /INFEL/ are set up for the current group and element.

(e) Tasks to be Performed in ENPR##

The following tasks must be performed.

1. If IMEM = 1 print an appropriate heading.
2. Print the envelope values.

9.13. SUBROUTINE THPR##

(a) Beginning Statements

The SUBROUTINE and COMMON statements for the subroutine THPR## are as follows.

```
SUBROUTINE THPR## (KSTAT, KHD, KSTEP, TIM, NFOUTP)
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /PTOP/ . . .
COMMON /THELM/ . . .
```

(b) Argument Types

The arguments are of following types.

Input: KSTAT, KHD, KSTEP, TIM, NFOUTP

(c) Purpose

Subroutine THPR## is called by the base program to print, on .OUT file (unit NFOUTP), element results at the end of an analysis segment.

(d) Tasks Performed Before Entry

The base program performs the following tasks.

1. Before calling THPR## for the first step of any element, a heading containing the analysis segment number, element group number and element number is printed to NFOUTP; and KHD is set to 1. After calling THPR## for the first step KHD is reset to 0.
2. The results for the current element and analysis step are set up in the /THELM/ block.

The /INFGR/ and /INFEL/ blocks are not set up. Hence, all data to be printed must be in the /THELM/ block (or in the argument list).

(e) Tasks to be Performed in THPR##

The following tasks must be performed.

1. If KHD = 1, this is the first set of results for the element. Print appropriate heading.
   a) KSTAT = 1 : for static analysis
   b) KSTAT = 2 : for dynamic anlysis.
2. Print element results.
   a) KSTAT = 1 : for static analysis
   b) KSTAT = 2 : for dynamic anlysis.

9.14. SUBROUTINE FLIN##

(a) Beginning Statements

The SUBROUTINE, COMMON, and DIMENSION statements for subroutine FLIN## are as follows.

```
SUBROUTINE FLIN## (NDOF, NFLIN, RELAS, EFLIN, DDISE)
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /INFGR/ . . .
COMMON /INFEL/ IMEM, . . .
COMMON /WORK/ . . .
DIMENSION RELAS(ndof), DDISE(ndof)
REAL EFLIN(nflin)
```

(b) Argument Types

The arguments are of following types.

Input: NDOF, NFLIN, DDISE

Output: RELAS, EFLIN

(c) Purpose

Subroutine FLIN## is called once for each mode and each element during response spectrum analysis. Its purpose is to form element results and static resisting forces, assuming linear behavior. The element results are used to set up the SRSS results output for the analysis. The resisting forces are used to calculate the forces on structure sections.

(d) Tasks Performed Before Entry

The following two tasks are performed by the base program before subroutine FLIN## is called.

1. Displacements (corresponding to one mode shape) are set up in array DDISE for the current element.
2. /INFGR/ and /INFEL/ are set up for the current group and element.

(e) Tasks to be Performed in FLIN##

The following tasks must be performed.

1. Determine element deformations using DDISE.
2. Calculate linear element actions corresponding to these deformations.

3. Set up the static resisting force vector, RELAS, in global coordinates.
4. Set up, in EFLIN, the element results which are to be combined (using SRSS combination) and output. Note that EFLIN is a single precision array.

9.15. SUBROUTINE PLIN##

(a) Beginning Statements

The SUBROUTINE, and DIMENSION statements for subroutine PLIN## are as follows.

```
SUBROUTINE PLIN## (KHD, IMEM, NFLIN, EFLIN, NFOUTP)
COMMON /ELMPAR/ RELPAR(2), IELPAR(2)
COMMON /PTOP/ . . .
REAL EFLIN(nflin)
```

(b) Argument Types

The arguments are of the following types.

Input: KHD, IMEM, NFLIN, EFLIN, NFOUTP

(c) Purpose

Subroutine PLIN## is called to print, on unit NFOUTP, the element results for response spectrum analyses. It is called once for each element.

(d) Tasks Performed Before Entry

The base program performs the following tasks.

1. Before calling PLIN## for the first element of any element group, a headind with the element group number is written to NFOUTP; and KHD is set to 1. After calling PLIN## for the first element KHD is reset to 0.
2. The response results for the current element are set up in array EFLIN.

(e) Tasks to be Performed in PLIN##

The following tasks must be performed.

1. If KHD =1 print a heading for the response results.
2. Print the response results.

10. CONCLUSIONS AND RECOMMENDATIONS

10.1 CONCLUSIONS

Does the world need another nonlinear structural analysis code? Yes, we believe that there is a need for a relatively simple program that can be used without a great deal of training, has the features needed for nonlinear seismic analysis, and can be maintained and extended with relatively modest effort. The DRAIN family of programs can fill this need. Increasingly it is being recognized that rational seismic-resistant design requires consideration of inelastic effects, and may require nonlinear analysis. We believe that the programs can be valuable tools not only for research but also for practical analysis. Compared with the commercially available nonlinear analysis codes, which have been developed mainly for finite element analysis, the DRAIN programs are relatively simple, both to use and to modify. They also have features that are specifically for civil engineering structures.

DRAIN-2DX has received a great deal of use, and seems to be reliable. DRAIN-3DX is virtually identical to DRAIN-2DX, and DRAIN-BUILDING is very similar. Hence, they can be maintained with little duplication of effort. Additional work is needed to complete the post-processing and geometry plotting programs, and to add to the element library. It is hoped that other researchers will find that the DRAIN programs are suitable platforms for developing new elements.

There are many improvements that can be made to the DRAIN programs, in the following broad areas.

1. Post-processing.
2. New elements.
3. Element interface.
4. Efficiency and portability.

5. New features.
6. Documentation.

Some specific improvements are listed in the following sections.

10.2. POST-PROCESSING

10.2.1. General Purpose Post-Processors

The DRAIN programs write files containing the model geometry (.GEO file), current states (.SXX files), time-history results (.RXX files) and result envelopes (.EXX files). At the present time, however, there are no programs available to perform post-processing operations on these files. Programs are needed (a) to plot the model geometry to help in checking the input data, (b) to plot deformed shapes and element states, (c) to plot result time-histories, and (d) to process result envelopes. Preliminary versions of programs for geometry and time-history plotting have been developed, but additional work is needed before they can be released for use.

10.2.2. User-Defined Output

Presently the user is able to control the items and step or time intervals for output of result histories and/or envelopes. The results for selected items are written to files for post-processing and/or printout. Some user organizations (e.g., CALTRANS) have indicated the need for further customization, so that the outputs can be used by their in-house programs.

One approach would be to produce customized versions of these programs. This is not recommended, because (a) it would require detailed knowledge of the program, and would therefore be expensive; and (b) it would be difficult to update the customized versions as the programs are updated due to coding changes, addition of new features and/or elements.

Another approach would be to provide sufficient flexibility in the program so that the user can add customized subroutines within a well-defined structured protocol. One possibility is as follows.

1. Provide two labeled common blocks, one for lists of items and one for step and/or time intervals for customized output.
2. On receiving instructions from the input file, the base program calls a customized subroutine to open files for receiving the customized output, and to set up the two labeled common blocks.
3. At each step, the base program calls a customized subroutine for customized output of each of the following.
   a) nodal displacements and envelopes.
   b) element results and envelopes for each element.
   c) structure section forces and envelopes.
   d) generalized displacements and envelopes.

   Before calling the customized subroutine, the base program will set up the necessary data in their argument list. For element results the labeled common blocks /THELM/ and /ENVELM/ would be provided.

10.2.3. Addition to .EXX files

The results envelope (.EXX) files currently contain envelopes for nodal displacements, structure section forces and relative displacements but do not contain element result envelopes. This feature must be added.

10.2.4. Frame Definition in DRAIN-BUILDING

DRAIN-BUILDING does not currently include the concept of a frame. A feature to allow frame results to be output is needed, presumably as a post-processing option.

10.3. NEW ELEMENTS

The element library for DRAIN-2DX currently consists mainly of simple elements that have been adapted from the original DRAIN-2D program. The element libraries for DRAIN-3DX and DRAIN-BUILDING currently consist of only a truss bar element and two beam-column elements that are not fully developed (a lumped plasticity fiber-hinge element and a distributed plasticity fiber-section element). Additional elements must be developed. It should be noted, however, that the development of a new element involves a great deal of effort.

10.4. ELEMENT INTERFACE

10.4.1. Weaknesses

In the development of a general purpose program for nonlinear analysis, the most challenging task is the design of the element interface. The interface in the DRAIN programs is basically sound, but it can be improved in a number of ways. The main weaknesses in the current design are as follows.

1. In the static gravity (*GRAV) analysis, when element loads are applied, the behavior must remain linear. The element interface must be extended to account for nonlinear behavior due to element loads.
2. In the state determination phase, the element resisting forces in the initial state must be returned, plus the change in the resisting forces from the initial state to the current state. This is an awkward procedure which should be improved.
3. Elements which have internal inertia can not be considered, because inertia forces and stiffness contributions are not included in the current interface. Also elements with internal viscous damping which is not of simple beta-K type can not be considered. Changes in the interface are needed to accommodate more general element types.

Some possible improvements are described in the following sections.

10.4.2. Storage of Element Location Matrix (LM array)

In earlier versions of DRAIN, the second variable in /INFEL/ had to be the "location matrix," LM(ndof). This is no longer the case and the base program now sets up the LM array through calls to LOCMAT in subroutine INEL##. Currently, the LM array for each element is stored following the /INFEL/ data in the /INFEL/ block, and is not visible or available to the element subroutines. Since the LM array is actually present in the /INFEL/ block, it is currently a part of the element interface. This storage scheme has caused difficulties.

A separate labeled common block, /LOCELM/, is proposed for storage of the LM array. The /LOCELM/ data will be stored following the /INFEL/ data in the base program buffer, INFB (see KINFB in /STOR/ or /STOR6/). This data will be restored to the /LOCELM/ block whenever it is required to process the current element.

/LOCELM/ will not be a part of the element interface, and will be used by the base program only.

10.4.3. Storage of Element Envelopes

Currently, the element envelopes are stored in the /INFEL/ block, and are not visible to the base program. For this reason the base program is unable to write element envelopes to the .EXX file for post-processing.

A new labeled common block, /ENVELM/, is proposed to store element envelopes. The length of /ENVELM/ will be conveyed to the base program as an argument in the call to ECONTR in INEL##. Subroutine RESP## will set up element envelopes in the /ENVELM/ block in much the same way as it presently sets up /THELM/.

10.4.4. Element End Displacements and Forces

Currently, arrays storing element end quantities (e.g., DDISE, DISE, VELE, RINIT, RELAS and RDAMP) are in the argument lists of subroutines RESP##, FACT## and FLIN##.

The existing labeled common block, /DISVEL/, could be used for such arrays, to reduce the argument list lengths. /DISVEL/ will then be a part of the element interface.

The declaration for /DISVEL/ will be as follows.

```
PARAMETER (MXDOFP=30)
COMMON /DISVEL/ ddise(MXDOFP), vele(MXDOFP), dise(MXDOFP),
1                         relas(MXDOFP), rdamp(MXDOFP), rinit(MXDOFP)
```

10.5. EFFICIENCY AND PORTABILITY

10.5.1. Windows Version

One major concern is that the programs compile without errors but do not work reliably on PCs under OS2 and Microsoft Windows. It is not clear whether this is due to coding errors or problems in the operating systems.

10.5.2. Large Capacity Equation Solver for DRAIN-3DX

In DRAIN-3DX the stiffness matrices are stored in memory. This limits the size of the problems that can be solved. A large capacity out-of-core equation solver is needed.

10.5.3. Large Capacity Eigensolver

In all DRAIN programs the eigenproblem is converted to flexibility form and solved using Tridiagonal QR iteration. This is efficient for up to about 300 mass DOFs, but is inefficient for large problems. Also the present eigensolver requires that the flexibility matrix be entirely in memory.

It is proposed to use Lanczos or subspace iteration eigensolver in future versions. This would allow very large problems to be solved, and would not require that the flexibility matrix be formed. The hypermatrix structure in DRAIN-BUILDING could also be exploited.

10.5.4. Display of Progress During Analysis

The programs do not currently inform the user how far the analysis has progressed. A change is needed to display the current time or load step as the analysis progresses.

10.5.5. Detection of Data Errors

The programs detect many errors in the input data. However, it is still possible for the programs to "crash", and additional checks are needed.

10.5.6. Execution Speed

Input-output to scratch files are done using FORTRAN READ and WRITE statements. It is possible to save significant time on PCs by using more efficient assembly language routines. The argument lists for some element subroutines are long, and it may be more efficient to use labeled commons. Some calls to element subroutines use IF-THEN constructs rather than CASE-OF (computed go-to), which is inefficient. It may be possible to reduce the maximum number of file units that are open at any time. These and other efficiency improvements could be made.

10.6. NEW FEATURES

10.6.1. Coupled Mass

In present programs the masses are lumped at the nodes, which results in a diagonal mass and alpha-M damping matrices. The structural mass actually arises from the masses of the structural members, and if a consistent mass theory is used at the element level, coupled matrices result. Coupled mass matrices also result when masses at slaved nodes are transferred to the master node (the programs currently ignore off-diagonal terms in this case).

A consistent mass matrix requires more memory and some mass related computations are more lengthy, but otherwise there is no reason why consistent mass could not be considered in nonlinear dynamic analysis. However, the current eigensolver requires a diagonal mass matrix. The element interface must also be extended to allow element masses to be returned to the base program.

The advantages of having coupled mass and damping matrices are that inertia forces can be modeled at the element level and a more accurate representation of the mass can be obtained. This feature becomes more important if complex substructured elements are developed, because such elements can have substantial internal inertia effects.

10.6.2. Free-Form and/or Spreadsheet Input

The programs all require a formatted input. This is admittedly antiquated, although the input format is much more flexible than the original DRAIN-2D format (sections of the data are separated by key words; it is no longer necessary to specify the numbers of nodes, elements, etc.; comments can be added to the data; and the nodes need not be numbered sequentially)

There is a need for a free-form input option for those users who prefer it. A spreadsheet-type input option would be particularly convenient.

10.6.3. Iteration on Unbalance

By using the event-to-event strategy, the programs seek to prevent significant unbalanced forces from developing. However, the strategy does not always achieve this goal, especially when there are substantial P-delta effects. An option to iterate on the unbalance would be useful in such cases.

10.6.4 Improved Response Spectrum Analysis

The complete quadratic combination (CQC) rule could be added as an option for combining modal responses in response spectrum analysis.

10.6.5 Improved Options for Damping Losses

The beta-K damping provision is included to account for miscellaneous energy losses which are not included in the nonlinear model. Unfortunately this is not a very rational method, and it does not give the analyst much flexibility. Improvements should be possible. One possibility is to provide for miscellaneous hysteretic, rather than viscous, losses.

10.6.6. Integration of Analyses

Currently, three distinct static analyses (i.e., static gravity, static and rest) are supported and there are separate driver subroutines to carry out these analyses. It is possible to combine these subroutines into a single driver.

The four distinct dynamic analyses (i.e., ground acceleration, ground displacement, initial velocity and dynamic force) are already supported by an integrated driver.

10.6.7. Simpler Structure Section Definition

When a structure section is defined it is currently necessary to input the location where each cut-element intersects the section plane, and also to input a force transformation matrix. Some or all of this input data could be calculated by the program. Note, however, that this is not an easy task, since the program must include many element types, including zero length elements.

10.6.8. Extension to Bridge Analysis

A new program, DRAIN-BRIDGE, with an option for linear substructuring could be developed from DRAIN-BUILDING. In this program a bridge could be modeled as an assemblage of *support structures* and *spans*. This is similar to the modeling of a building by *floors* and *interfloors* in DRAIN-BUILDING. Additional efficiency could be obtained if the spans are assumed to be linear (as will often be the case), by treating each span as a linear substructure and condensing out its internal degrees of freedom.

10.7. DOCUMENTATION

This report documents the program design. Other reports include (a) user guides for the three programs, (b) examples to guide users in developing nonlinear models and performing analyses, (c) programming instructions for developers of new elements, (d) static and dynamic analysis theory, and (e) modeling guidelines for developers of new elements. Some of these reports are not yet complete. There is also a need for additional examples to guide users in the effective use of nonlinear analysis.

REFERENCES

1. Allahabadi, R., "DRAIN-2DX -- Seismic Response and Damage Assessment for 2D Structures," Ph.D. dissertation, University of California, Berkeley, 1987.
2. Allahabadi, R., and G. H. Powell, "DRAIN-2DX User Guide," Report No. UCB / EERC-88/06, March 1988.
3. Bathe, K. J., and E. L. Wilson, " Numerical Methods in Finite Element Analysis," Prentice Hall, Inc., NJ, 1976.
4. Clough, R. W., and J. Penzien, "Dynamics of Structures," McGraw-Hill, Inc., 1975.
5. Fuchs, G. F., J. R. Roy and E. Schrem, "Hypermatrix Solution of Large Sets of Symmetric Positive Definite Linear Equations," Computer Methods in Applied Mechanics and Engineering, pp. 197-206, 1972.
6. George, A., and Liu, J. W-H, "Computer Solution of Large Sparse Positive Definite Systems," Prentice Hall, Inc., Englewood Cliffs, NJ 07632, 1981.
7. Golafshani, A., "A Computer Program for Inelastic Seismic Response of Structures," Ph.D. dissertation, University of California, Berkeley, 1982.
8. Golub, G. H., and C. F. Van Loan, "Matrix Computations", Second Edition, Johns Hopkins University Press, 1989.
9. Jennings, A., "A Compact Storage Scheme for the Solution of Symmetric Linear Simultaneous Equations," Comput. Journal 9, pp. 281-285, 1966.
10. Powell, G. H., "DRAIN-2D User Guide," Report No. UCB / EERC 73-22, Earthquake Engineering Research Center, University of California, Berkeley, CA, April 1973.
11. Prakash, V., and G. H. Powell, "DRAIN-2DX Version 1.02 User Guide," a Computer Program distributed by NISEE / Computer Applications, Department of Civil Engineering, University of California, Berkeley, CA, July 1992.

12. Prakash, V., and G. H. Powell, "DRAIN-3DX User Guide," a Computer Program to be distributed through NISEE / Computer Applications, Department of Civil Engineering, University of California, Berkeley, CA.

13. Prakash, V., and G. H. Powell, "DRAIN-BUILDING User Guide," a Computer Program to be distributed through NISEE / Computer Applications, Department of Civil Engineering, University of California, Berkeley, CA.